Artificial Intelligence 2
Summer Semester 2025

– Lecture Notes –

Prof. Dr. Michael Kohlhase

Professur für Wissensrepräsentation und -verarbeitung
Informatik, FAU Erlangen-Nürnberg
`Michael.Kohlhase@FAU.de`

2025-05-14

SOME RIGHTS RESERVED

Chapter 1
Preliminaries

SOME RIGHTS RESERVED

1.1 TL;DR: Goals and Links

SOME RIGHTS RESERVED

What you should learn here...

- **What you should learn in AI-2:**
  - In the broadest sense: *A bunch of tools for your toolchest* (i.e. various (quasi-mathematical) models, first and foremost)

SOME RIGHTS RESERVED

What you should learn here...

- **What you should learn in AI-2:**
  - In the broadest sense: *A bunch of tools for your toolchest* (i.e. various (quasi-mathematical) models, first and foremost)
  - the underlying *principles* of these models (assumptions, limitations, the math behind them ...)

What you should learn here...

- **What you should learn in AI-2:**
  - In the broadest sense: *A bunch of tools for your toolchest* (i.e. various (quasi-mathematical) models, first and foremost)
  - the underlying *principles* of these models (assumptions, limitations, the math behind them ...)
  - the ability to describe real-world problems in terms of these models, **where adequate** (...and knowing **when** they are adequate!),

SOME RIGHTS RESERVED

What you should learn here...

- **What you should learn in AI-2:**
  - In the broadest sense: *A bunch of tools for your toolchest* (i.e. various (quasi-mathematical) models, first and foremost)
  - the underlying *principles* of these models (assumptions, limitations, the math behind them ...)
  - the ability to describe real-world problems in terms of these models, **where adequate** (...and knowing **when** they are adequate!), and
  - the ideas behind effective *algorithms* that solve these problems (and to understand them well enough to implement them)
- **Note:** You will likely never get payed to implement an algorithm that e.g. solves Bayesian networks. (They already exist)
  - *But* you might get payed to *recognize* that some given problem *can be* represented as a Bayesian network!
  - **Or:** you can recognize that it is *similar to* a Bayesian network, and reuse the underlying principles to develop new specialized tools.

SOME RIGHTS RESERVED

Compare two employees

- **“We have the following problem and we need a solution: . . . ”**

Compare two employees

- **“We have the following problem and we need a solution: ...”**
- **Employee 1 – Deep Learning can do everything:** “I just need ≈1.5 million labeled examples of potentially sensitive data, a GPU cluster for training, and a few weeks to train, tweak and finetune the model.
  But *then* I can solve the problem... with a confidence of 95%, within 40 seconds of inference per input. Oh, as long as the input isn't longer than 15`unit`, or I will need to retrain on a bigger input layer...”

SOME RIGHTS RESERVED

Compare two employees

- **“We have the following problem and we need a solution: . . . ”**
- **Employee 1 – Deep Learning can do everything:** “I just need ≈1.5 million labeled examples of potentially sensitive data, a GPU cluster for training, and a few weeks to train, tweak and finetune the model.
  But *then* I can solve the problem… with a confidence of 95%, within 40 seconds of inference per input. Oh, as long as the input isn't longer than 15`unit`, or I will need to retrain on a bigger input layer…”
- **Employee 2 – AI-2 Alumna:** “…while you were talking, I quickly built a custom UI for an off-the-shelve `<problem>` solver that runs on a medium-sized potato and returns a *provably correct* result in a few milliseconds. For inputs longer than 1000`unit`, you might need a slightly bigger potato though…”

SOME RIGHTS RESERVED

Compare two employees

- **“We have the following problem and we need a solution: . . . ”**
- **Employee 1 – Deep Learning can do everything:** “I just need ≈1.5 million labeled examples of potentially sensitive data, a GPU cluster for training, and a few weeks to train, tweak and finetune the model.
  But *then* I can solve the problem... with a confidence of 95%, within 40 seconds of inference per input. Oh, as long as the input isn’t longer than 15`unit`, or I will need to retrain on a bigger input layer...”
- **Employee 2 – AI-2 Alumna:** “...while you were talking, I quickly built a custom UI for an off-the-shelve `<problem>` solver that runs on a medium-sized potato and returns a *provably correct* result in a few milliseconds. For inputs longer than 1000`unit`, you might need a slightly bigger potato though...”
- **Moral of the story:** Know your *tools* well enough to select the right one for the job.

SOME RIGHTS RESERVED

Dates, Links, Materials

- **Lectures**: Tuesday 16:15 – 17:45 **H9**, Thursday 10:15 – 11:45 **H8**
- **Tutorials**:
  - Friday 10:15 – 11:45 *Room 11501.02.019*
  - Friday 14:15 – 15:45 *Zoom:* `https://fau.zoom.us/j/97169402146`
  - Monday 12:15 – 13:45 *Room H4*
  - Tuesday 08:15 – 09:45 *Room 11302.02.134-113*

(Starting thursday in week 2 (25.04.2024))

SOME RIGHTS RESERVED

Dates, Links, Materials

- **Lectures**: Tuesday 16:15 – 17:45 **H9**, Thursday 10:15 – 11:45 **H8**
- **Tutorials**:
  - Friday 10:15 – 11:45 *Room 11501.02.019*
  - Friday 14:15 – 15:45 *Zoom:* `https://fau.zoom.us/j/97169402146`
  - Monday 12:15 – 13:45 *Room H4*
  - Tuesday 08:15 – 09:45 *Room 11302.02.134-113*

  (Starting thursday in week 2 (25.04.2024))
- **studon**: `https://www.studon.fau.de/studon/goto.php?target=crs_5645530` (Used for announcements, e.g. homeworks, and homework submissions)
- **Video streams / recordings**: `https://www.fau.tv/course/id/3816`
- **Lecture notes / slides / exercises:** `https://kwarc.info/teaching/AI/` (Most importantly: `notes2.pdf` and `slides2.pdf`)
- **ALeA**: `https://courses.voll-ki.fau.de/course-home/ai-2`: Lecture notes, forum, **tuesday quizzes**, flashcards,...

Textbook: *Russel/Norvig: Artificial Intelligence, A modern Approach* [**RusNor:AIMA09**]. Make sure that you read the edition ≥ 3 ⇝ vastly improved over ≤ 2.

SOME RIGHTS RESERVED

1.2 Administrative Ground Rules

SOME RIGHTS RESERVED

Prerequisites for AI-2

- **Content Prerequisites:** The mandatory courses in CS@FAU; Sem. 1-4, in particular:
    - Course "Algorithmen und Datenstrukturen". (Algorithms & Data Structures)
    - Course "Grundlagen der Logik in der Informatik" (GLOIN). (Logic in CS)
    - Course "Berechenbarkeit und Formale Sprachen". (Theoretical CS)

SOME RIGHTS RESERVED

Prerequisites for AI-2

- **Content Prerequisites:** The mandatory courses in CS@FAU; Sem. 1-4, in particular:
  - Course “Algorithmen und Datenstrukturen”. (Algorithms & Data Structures)
  - Course “Grundlagen der Logik in der Informatik” (GLOIN). (Logic in CS)
  - Course “Berechenbarkeit und Formale Sprachen”. (Theoretical CS)
- **Skillset Prerequisite:** Coping with mathematical formulation of the structures
  - Mathematics is the language of science (in particular CS)
  - It allows us to be very precise about what we mean. (good for you)

SOME RIGHTS RESERVED

Prerequisites for AI-2

- **Content Prerequisites:** The mandatory courses in CS@FAU; Sem. 1-4, in particular:
  - Course “Algorithmen und Datenstrukturen”. (Algorithms & Data Structures)
  - Course “Grundlagen der Logik in der Informatik” (GLOIN). (Logic in CS)
  - Course “Berechenbarkeit und Formale Sprachen”. (Theoretical CS)
- **Skillset Prerequisite:** Coping with mathematical formulation of the structures
  - Mathematics is the language of science (in particular CS)
  - It allows us to be very precise about what we mean. (good for you)
- **Intuition:** (take them with a kilo of salt)
  - This is what I assume you know! (I have to assume something)
  - In most cases, the dependency on these is partial and “in spirit”.
  - If you have not taken these (or do not remember), read up on them as needed!

SOME RIGHTS RESERVED

Prerequisites for AI-2

- **Content Prerequisites:** The mandatory courses in CS@FAU; Sem. 1-4, in particular:
  - Course “Algorithmen und Datenstrukturen”. (Algorithms & Data Structures)
  - Course “Grundlagen der Logik in der Informatik” (GLOIN). (Logic in CS)
  - Course “Berechenbarkeit und Formale Sprachen”. (Theoretical CS)
- **Skillset Prerequisite:** Coping with mathematical formulation of the structures
  - Mathematics is the language of science (in particular CS)
  - It allows us to be very precise about what we mean. (good for you)
- **Intuition:** (take them with a kilo of salt)
  - This is what I assume you know! (I have to assume something)
  - In most cases, the dependency on these is partial and “in spirit”.
  - If you have not taken these (or do not remember), read up on them as needed!
- **Real Prerequisites:** Motivation, interest, curiosity, hard work. (AI-2 is non-trivial)
- You can do this course if you want! (and I hope you are successful)

SOME RIGHTS RESERVED

Assessment, Grades

- **Overall (Module) Grade:**
  - Grade via the exam (Klausur) $\leadsto$ 100% of the grade.
  - Up to 10% bonus on-top for an exam with $\geq$ 50% points. (< 50% $\leadsto$ no bonus)
  - Bonus points $\widehat{=}$ percentage sum of the best 10 prepquizzes divided by 100.

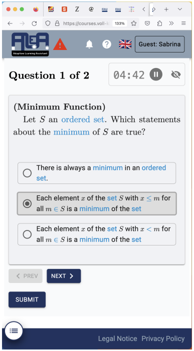
SOME RIGHTS RESERVED

Assessment, Grades

- **Overall (Module) Grade:**
  - Grade via the exam (Klausur) $\rightsquigarrow$ 100% of the grade.
  - Up to 10% bonus on-top for an exam with $\geq 50\%$ points. ($< 50\% \rightsquigarrow$ no bonus)
  - Bonus points $\widehat{=}$ percentage sum of the best 10 prepquizzes divided by 100.
- **Exam:** exam conducted in presence on paper! ($\sim$ Oct. 10. 2025)
- **Retake Exam:** 90 minutes exam six months later. ($\sim$ April 10. 2026)
- ⚠ You have to register for exams in `https://campo.fau.de` in the first month of classes.
- **Note:** You can de-register from an exam on `https://campo.fau.de` up to three working days before exam. (do not miss that if you are not prepared)

Preparedness Quizzes

- **PrepQuizzes:** Before every lecture we offer a 10 min online quiz – the PrepQuiz – about the material from the previous week. (16:15-16:25; starts in week 2)
- **Motivations:** We do this to
  - keep you prepared and working continuously. (primary)
  - bonus points if the exam has $\geq 50\%$ points (potential part of your grade)
  - update the ALeA learner model. (fringe benefit)
- The prepquizes will be given in the ALeA system



Question 1 of 2 04:42

(Minimum Function)
Let S an ordered set. Which statements about the minimum of S are true?

○ There is always a minimum in an ordered set.

● Each element x of the set S with $x \leq m$ for all $m \in S$ is a minimum of the set

○ Each element x of the set S with $x < m$ for all $m \in S$ is a minimum of the set

- `https://courses.voll-ki.fau.de/quiz-dash/ai-2`
- You have to be logged into ALeA! (via FAU IDM)
- You can take the prepquiz on your laptop or phone, …
- …in the lecture or at home …
- …via WLAN or 4G Network. (do not overload)
- Prepquizzes will only be available 16:15-16:25!

SOME RIGHTS RESERVED

Next Week: Pretest

- Next week we will try out the prepquiz infrastructure with a pretest!
  - **Presence**: bring your laptop or cellphone.
  - **Online**: you can and should take the pretest as well.
  - Have a recent firefox or chrome (chrome: younger than March 2023)
  - Make sure that you are logged into ALeA (via FAU IDM; see below)
- **Definition 2.1.** A pretest is an assessment for evaluating the preparedness of learners for further studies.
- **Concretely:** This pretest
  - establishes a baseline for the competency expectations in and
  - tests the ALeA quiz infrastructure for the prepquizzes.
- Participation in the pretest is optional; it will not influence grades in any way.
- The pretest covers the prerequisites of AI-2 and some of the material that may have been covered in other courses.
- The test will be also used to refine the ALeA learner model, which may make learning experience in ALeA better. (see below)

SOME RIGHTS RESERVED

Special Admin Conditions

- Some degree programs do not "import" the course Artificial Intelligence 1, and thus you may not be able to register for the exam via `https://campo.fau.de`.
  - Just send me an e-mail and come to the exam, (we do the necessary admin)
  - Tell your program coordinator about AI-1/2 so that they remedy this situation
- In "Wirtschafts-Informatik" you can only take AI-1 and AI-2 together in the "Wahlpflichtbereich".
  - ECTS credits need to be divisible by five $\leadsto 7.5 + 7.5 = 15$.

SOME RIGHTS RESERVED

1.3 Getting Most out of AI-2

SOME RIGHTS RESERVED

AI-2 Homework Assignments

- **Goal:** Homework assignments reinforce what was taught in lectures.
- **Homework Assignments:** Small individual problem/programming/proof task
  - but take time to solve (at least read them directly ⇝ questions)
- **Didactic Intuition:** Homework assignments give you material to test your understanding and show you how to apply it.
- ⚠ Homeworks give no points, but without trying you are unlikely to pass the exam.
- **Our Experience:** Doing your homework is probably even *more* important (and predictive of exam success) than attending the lecture in person!

SOME RIGHTS RESERVED

AI-2 Homework Assignments

- **Goal:** Homework assignments reinforce what was taught in lectures.
- **Homework Assignments:** Small individual problem/programming/proof task
  - but take time to solve (at least read them directly ⇝ questions)
- **Didactic Intuition:** Homework assignments give you material to test your understanding and show you how to apply it.
- ⚠ Homeworks give no points, but without trying you are unlikely to pass the exam.
- **Our Experience:** Doing your homework is probably even *more* important (and predictive of exam success) than attending the lecture in person!
- Homeworks will be mainly peer-graded in the ALeA system.
- **Didactic Motivation:** Through peer grading students are able to see mistakes in their thinking and can correct any problems in future assignments. By grading assignments, students may learn how to complete assignments more accurately and how to improve their future results. (not just us being lazy)

SOME RIGHTS RESERVED

AI-2 Homework Assignments – Howto

- **Homework Workflow:** in ALeA (see below)
  - Homework assignments will be published on thursdays: see `https://courses.voll-ki.fau.de/hw/ai-1`
  - Submission of solutions via the ALeA system in the week after
  - Peer grading/feedback (and master solutions) via answer classes.
- **Quality Control:** TAs and instructors will monitor and supervise peer grading.

SOME RIGHTS RESERVED

AI-2 Homework Assignments – Howto

- **Homework Workflow:** in ALeA (see below)
  - Homework assignments will be published on thursdays: see `https://courses.voll-ki.fau.de/hw/ai-1`
  - Submission of solutions via the ALeA system in the week after
  - Peer grading/feedback (and master solutions) via answer classes.
- **Quality Control:** TAs and instructors will monitor and supervise peer grading.
- **Experiment:** Can we motivate enough of you to make peer assessment self-sustaining?
  - I am appealing to your sense of community responsibility here ...
  - You should only expect other's to grade your submission if you grade their's (cf. Kant's "Moral Imperative")
  - **Make no mistake**: The grader usually learns at least as much as the gradee.

SOME RIGHTS RESERVED

AI-2 Homework Assignments – Howto

- **Homework Workflow:** in ALeA (see below)
  - Homework assignments will be published on thursdays: see `https://courses.voll-ki.fau.de/hw/ai-1`
  - Submission of solutions via the ALeA system in the week after
  - Peer grading/feedback (and master solutions) via answer classes.
- **Quality Control:** TAs and instructors will monitor and supervise peer grading.
- **Experiment:** Can we motivate enough of you to make peer assessment self-sustaining?
  - I am appealing to your sense of community responsibility here …
  - You should only expect other's to grade your submission if you grade their's (cf. Kant's "Moral Imperative")
  - **Make no mistake**: The grader usually learns at least as much as the gradee.
- **Homework/Tutorial Discipline:**
  - Start early! (many assignments need more than one evening's work)
  - Don't start by sitting at a blank screen (talking & study groups help)
  - Humans will be trying to understand the text/code/math when grading it.
  - Go to the tutorials, discuss with your TA! (they are there for you!)

SOME RIGHTS RESERVED

Tutorials for Artificial Intelligence 1

- **Approach:** Weekly tutorials and homework assignments (first one in week two)
- **Goal 1:** Reinforce what was taught in the lectures. (you need practice)
- **Goal 2:** Allow you to ask any question you have in a protected environment.

SOME RIGHTS RESERVED

Tutorials for Artificial Intelligence 1

- **Approach:** Weekly tutorials and homework assignments (first one in week two)
- **Goal 1:** Reinforce what was taught in the lectures. (you need practice)
- **Goal 2:** Allow you to ask any question you have in a protected environment.
- **Instructor/Lead TA:** Florian Rabe (KWARC Postdoc, Privatdozent)
  - Room: 11.137 @ Händler building, `florian.rabe@fau.de`
- **Tutorials:** One each taught by Florian Rabe (lead); Primula Mukherjee, Ilhaam Shaikh, Praveen Kumar Vadlamani, and Shreya Rajesh More.
  - Tutorials will start in week 3. (before there is nothing to do)
  - Details (rooms, times, etc) will be announced in time (i.e. not now) on the forum and matrix channel.
- **Life-saving Advice:** Go to your tutorial, and prepare for it by having looked at the slides and the homework assignments!

SOME RIGHTS RESERVED

Collaboration

- **Definition 3.1.** Collaboration (or cooperation) is the process of groups of agents acting together for common, mutual benefit, as opposed to acting in competition for selfish benefit. In a collaboration, every agent contributes to the common goal and benefits from the contributions of others.
- In learning situations, the benefit is "better learning".
- **Observation:** In collaborative learning, the overall result can be significantly better than in competitive learning.
- **Good Practice:** Form study groups. (long- or short-term)
  1. ⚠ Those learners who work/help most, learn most!
  2. ⚠ Freeloaders – individuals who only watch – learn very little!
- It is OK to collaborate on homework assignments in AI-2! (no bonus points)
- Choose your study group well! (ALeA helps via the study buddy feature)

SOME RIGHTS RESERVED

Do I need to attend the AI-2 Lectures

- Attendance is not mandatory for the AI-2 course. (official version)
- **Note:** There are two ways of learning: (both are OK, your mileage may vary)
  - Approach B: Read a book/papers (here: lecture notes)
  - Approach I: come to the lectures, be involved, interrupt the instructor whenever you have a question.

  The only advantage of I over B is that books/papers do not answer questions
- Approach S: come to the lectures and sleep does not work!
- The closer you get to research, the more we need to discuss!

SOME RIGHTS RESERVED

1.4 Learning Resources for AI-2

SOME RIGHTS RESERVED

Textbooks and supplementary Literature

- **Textbook:** *Russel/Norvig: Artificial Intelligence, A modern Approach* [**RusNor:AIMA09**].
  - basically “broad but somewhat shallow”
  - great to get intuitions on the basics of AI

  Make sure that you read the edition ≥ 3 ⇜ vastly improved over ≤ 2.

SOME RIGHTS RESERVED

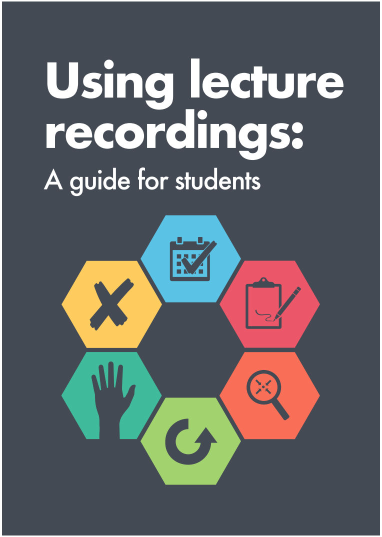
Course Notes, Forum, Matrix

- **Lecture notes** will be posted at `https://kwarc.info/teaching/AI`
  - We mostly prepare/update them as we go along (semantically preloaded ⇝ research resource)
  - Please report any errors/shortcomings you notice. (improve for the group/successors)
- **StudOn Forum:** For announcements – `https://www.studon.fau.de/studon/goto.php?target=lcode_70Bjcaxg`
- **Matrix Channel:** `https://matrix.to/#/#ai-12:fau.de` for questions, discussion with instructors and among your fellow students. (your channel, use it!)

  Login via FAU IDM ⇝ instructions
- **Course Videos** are at at `https://fau.tv/course/id/4225`.
- **Do not let the videos mislead you:** Coming to class is highly correlated with passing the exam!

Practical recommendations on Lecture Videos

- **Excellent Guide:** [**NorKueRob:lcprs18**] (German version at [**NorKueRob:vnas18**])



Using lecture recordings:

A guide for students

Attend lectures.

Take notes.

Be specific.

Catch up.

Ask for help.

Don't cut corners.

SOME RIGHTS RESERVED

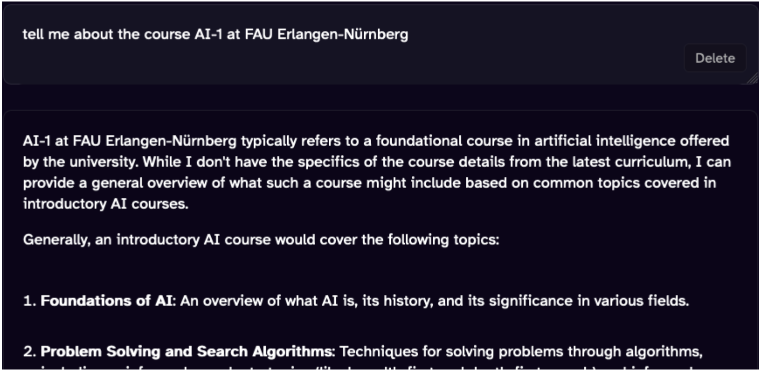
NOT a Resource for : LLMs – AI-based tools like ChatGPT

- **Definition 4.1.** A large language model (LLM) is a computational model capable of language generation or other natural language processing tasks.
- **Example 4.2.** OpenAI's GPT, Google's Bard, and Meta's Llama.

SOME RIGHTS RESERVED

NOT a Resource for : LLMs – AI-based tools like ChatGPT

- **Definition 4.6.** A large language model (LLM) is a computational model capable of language generation or other natural language processing tasks.
- **Example 4.7.** OpenAI's GPT, Google's Bard, and Meta's Llama.
- **Definition 4.8.** A chatbot is a software application or web interface that is designed to mimic human conversation through text or voice interactions. Modern chatbots are usually based on LLMs.
- **Example 4.9 (ChatGPT talks about AI-1).** (but remains vague)



tell me about the course AI-1 at FAU Erlangen-Nürnberg

AI-1 at FAU Erlangen-Nürnberg typically refers to a foundational course in artificial intelligence offered by the university. While I don't have the specifics of the course details from the latest curriculum, I can provide a general overview of what such a course might include based on common topics covered in introductory AI courses.

Generally, an introductory AI course would cover the following topics:

1. **Foundations of AI:** An overview of what AI is, its history, and its significance in various fields.

2. **Problem Solving and Search Algorithms:** Techniques for solving problems through algorithms,

SOME RIGHTS RESERVED

NOT a Resource for : LLMs – AI-based tools like ChatGPT

- **Definition 4.11.** A large language model (LLM) is a computational model capable of language generation or other natural language processing tasks.
- **Example 4.12.** OpenAI's GPT, Google's Bard, and Meta's Llama.
- **Definition 4.13.** A chatbot is a software application or web interface that is designed to mimic human conversation through text or voice interactions. Modern chatbots are usually based on LLMs.
- **Example 4.14 (ChatGPT talks about AI-1).** (but remains vague)
- **Note:** LLM-based chatbots invent *every word*! (surpisingly often correct)
- **Example 4.15 (In the AI-1 exam).** ChatGPT scores ca. 50% of the points.
  - ChatGPT can almost pass the exam … (We could award it a Master's degree)
  - But can you? (the AI-1 exams will be in person on paper)

  You will only pass the exam, if you can do AI-1 yourself!

SOME RIGHTS RESERVED

NOT a Resource for : LLMs – AI-based tools like ChatGPT

- **Definition 4.16.** A large language model (LLM) is a computational model capable of language generation or other natural language processing tasks.
- **Example 4.17.** OpenAI's GPT, Google's Bard, and Meta's Llama.
- **Definition 4.18.** A chatbot is a software application or web interface that is designed to mimic human conversation through text or voice interactions. Modern chatbots are usually based on LLMs.
- **Example 4.19 (ChatGPT talks about AI-1).** (but remains vague)
- **Note:** LLM-based chatbots invent *every word*! (surprisingly often correct)
- **Example 4.20 (In the AI-1 exam).** ChatGPT scores ca. 50% of the points.
  - ChatGPT can almost pass the exam . . . (We could award it a Master's degree)
  - But can you? (the AI-1 exams will be in person on paper)

  You will only pass the exam, if you can do AI-1 yourself!
- **Intuition:** AI tools like GhatGPT, CoPilot, etc. (see also [**Shein:iacse24**])
  - can help you solve problems, (valuable tools in production situations)
  - hinders learning if used for homeworks/quizzes, etc. (like driving instead of jogging)

SOME RIGHTS RESERVED

NOT a Resource for : LLMs – AI-based tools like ChatGPT

- **Definition 4.21.** A large language model (LLM) is a computational model capable of language generation or other natural language processing tasks.
- **Example 4.22.** OpenAI's GPT, Google's Bard, and Meta's Llama.
- **Definition 4.23.** A chatbot is a software application or web interface that is designed to mimic human conversation through text or voice interactions. Modern chatbots are usually based on LLMs.
- **Example 4.24 (ChatGPT talks about AI-1).** (but remains vague)
- **Note:** LLM-based chatbots invent *every word*! (surpisingly often correct)
- **Example 4.25 (In the AI-1 exam).** ChatGPT scores ca. 50% of the points.
  - ChatGPT can almost pass the exam … (We could award it a Master's degree)
  - But can you? (the AI-1 exams will be in person on paper)

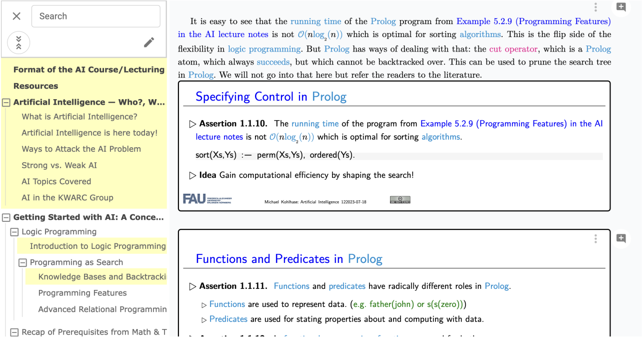
  You will only pass the exam, if you can do AI-1 yourself!
- **Intuition:** AI tools like GhatGPT, CoPilot, etc. (see also [**Shein:iacse24**])
  - can help you solve problems, (valuable tools in production situations)
  - hinders learning if used for homeworks/quizzes, etc. (like driving instead of jogging)
- **What (not) to do:** (to get most of the brave new AI-supported world)
  - try out these tools to get a first-hand intuition what they can/cannot do
  - challenge yourself while learning so that you can also do it (mind over matter!)

SOME RIGHTS RESERVED

1.5 ALeA – AI-Supported Learning

ALEA: Adaptive Learning Assistant

- **Idea:** Use AI methods to help teach/learn AI (AI4AI)
- **Concretely:** Provide HTML versions of the AI-2 slides/lecture notes and embed learning support services into them. (for pre/postparation of lectures)
- **Definition 5.1.** Call a document active, iff it is interactive and adapts to specific information needs of the readers. (lecture notes on steroids)
- **Intuition:** ALEA serves active course materials. (PDF mostly inactive)
- **Goal:** Make ALEA more like a instructor + study group than like a book!
- **Example 5.2 (Course Notes).** $\widehat{=}$ Slides + Comments



Search

Format of the AI Course/Lecturing

Resources

- **Artificial Intelligence — Who?, W...**
  - What is Artificial Intelligence?
  - Artificial Intelligence is here today!
  - Ways to Attack the AI Problem
  - Strong vs. Weak AI
  - AI Topics Covered
  - AI in the KWARC Group
- **Getting Started with AI: A Conce...**
  - Logic Programming
    - Introduction to Logic Programming
    - Programming as Search
      - Knowledge Bases and Backtracki
      - Programming Features
      - Advanced Relational Programmin
  - Recap of Prerequisites from Math & T

It is easy to see that the running time of the Prolog program from Example 5.2.9 (Programming Features) in the AI lecture notes is not $\mathcal{O}(n\log_2(n))$ which is optimal for sorting algorithms. This is the flip side of the flexibility in logic programming. But Prolog has ways of dealing with that: the cut operator, which is a Prolog atom, which always succeeds, but which cannot be backtracked over. This can be used to prune the search tree in Prolog. We will not go into that here but refer the readers to the literature.

Specifying Control in Prolog

▷ **Assertion 1.1.10.** The running time of the program from Example 5.2.9 (Programming Features) in the AI lecture notes is not $\mathcal{O}(n\log_2(n))$ which is optimal for sorting algorithms.

```
sort(Xs,Ys) :— perm(Xs,Ys), ordered(Ys).
```

▷ **Idea** Gain computational efficiency by shaping the search!

Michael Kohlhase: Artificial Intelligence 122023-07-18

Functions and Predicates in Prolog

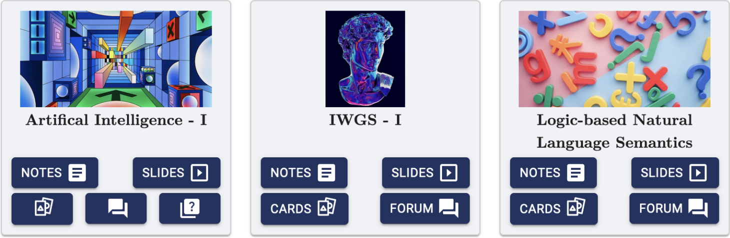
▷ **Assertion 1.1.11.** Functions and predicates have radically different roles in Prolog.

- ▷ Functions are used to represent data. (e.g. father(john) or s(s(zero)))
- ▷ Predicates are used for stating properties about and computing with data.

⇝ yellow parts in table of contents (left) already covered in lectures.

VoLL-KI Portal at `https://courses.voll-ki.fau.de`

- **Portal for ALeA Courses:** `https://courses.voll-ki.fau.de`


Artifical Intelligence - I
NOTES
SLIDES
IWGS - I
NOTES
SLIDES
CARDS
FORUM
Logic-based Natural Language Semantics
NOTES
SLIDES
CARDS
FORUM

- **AI-2 in ALeA:** `https://courses.voll-ki.fau.de/course-home/ai-2`
  - All details for the course.
  - recorded syllabus (keep track of material covered in course)
  - syllabus of the last semesters (for over/preview)
- **ALeA Status:** The ALEA system is deployed at FAU for over 1000 students taking eight courses
  - (some) students use the system actively (our logs tell us)
  - reviews are mostly positive/enthusiastic (error reports pour in)

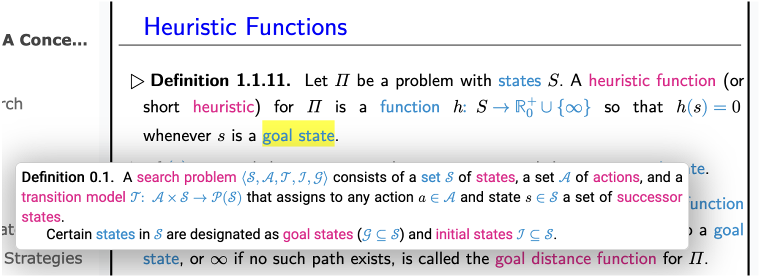
Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.

SOME RIGHTS RESERVED

Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 5.6 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of its definition. (even works recursively)



Heuristic Functions

▷ **Definition 1.1.11.** Let Π be a problem with states S. A heuristic function (or short heuristic) for Π is a function $h\colon S \to \mathbb{R}^+_0 \cup \{\infty\}$ so that $h(s) = 0$ whenever s is a goal state.

Definition 0.1. A search problem $\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$ consists of a set $\mathcal{S}$ of states, a set $\mathcal{A}$ of actions, and a transition model $\mathcal{T}\colon \mathcal{A}\times\mathcal{S}\to\mathcal{P}(\mathcal{S})$ that assigns to any action $a\in\mathcal{A}$ and state $s\in\mathcal{S}$ a set of successor states.

Certain states in $\mathcal{S}$ are designated as goal states ($\mathcal{G}\subseteq\mathcal{S}$) and initial states $\mathcal{I}\subseteq\mathcal{S}$.

te.

unction

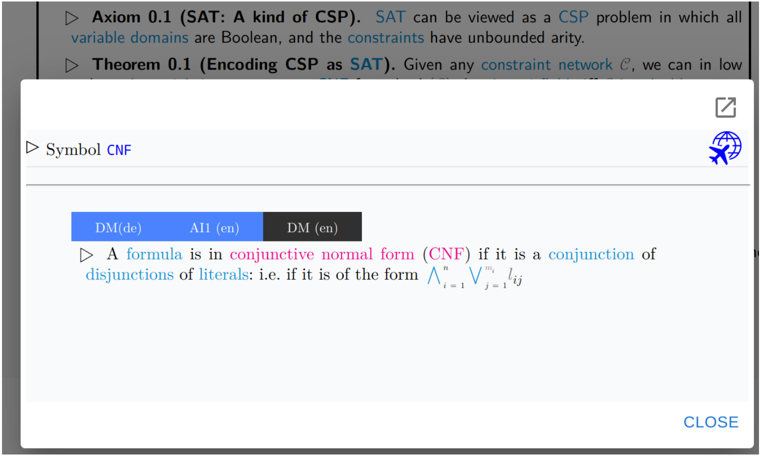
a goal

state, or ∞ if no such path exists, is called the goal distance function for Π.

SOME RIGHTS RESERVED

Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 5.9 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of its definition. (even works recursively)
- **Example 5.10 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.



▷ **Axiom 0.1 (SAT: A kind of CSP).** SAT can be viewed as a CSP problem in which all variable domains are Boolean, and the constraints have unbounded arity.

▷ **Theorem 0.1 (Encoding CSP as SAT).** Given any constraint network $\mathcal{C}$, we can in low

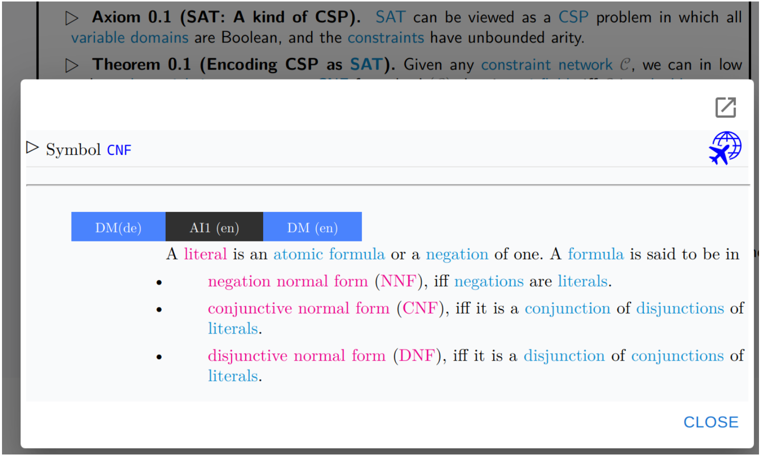
▷ Symbol CNF

▷ A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals: i.e. if it is of the form $\bigwedge_{i=1}^{n}\bigvee_{j=1}^{m_i} l_{ij}$

SOME RIGHTS RESERVED

Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 5.12 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of its definition. (even works recursively)
- **Example 5.13 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.



▷ **Axiom 0.1 (SAT: A kind of CSP).** SAT can be viewed as a CSP problem in which all variable domains are Boolean, and the constraints have unbounded arity.

▷ **Theorem 0.1 (Encoding CSP as SAT).** Given any constraint network $\mathcal{C}$, we can in low

▷ Symbol CNF

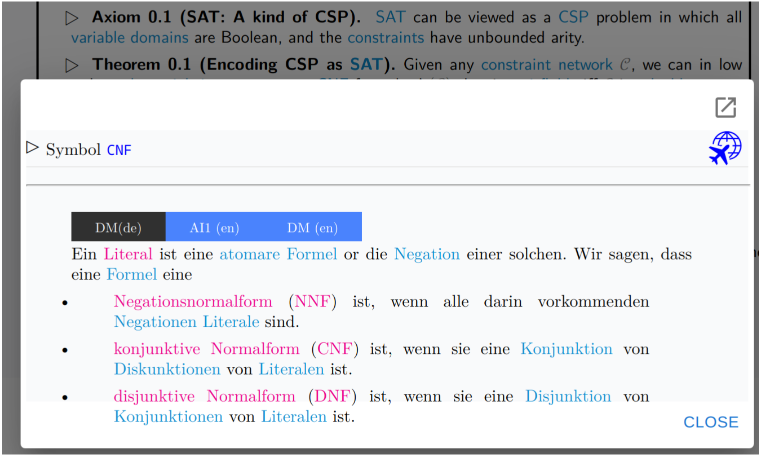
A literal is an atomic formula or a negation of one. A formula is said to be in

- negation normal form (NNF), iff negations are literals.
- conjunctive normal form (CNF), iff it is a conjunction of disjunctions of literals.
- disjunctive normal form (DNF), iff it is a disjunction of conjunctions of literals.

SOME RIGHTS RESERVED

Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 5.15 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of its definition. (even works recursively)
- **Example 5.16 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.



▷ **Axiom 0.1 (SAT: A kind of CSP).** SAT can be viewed as a CSP problem in which all variable domains are Boolean, and the constraints have unbounded arity.

▷ **Theorem 0.1 (Encoding CSP as SAT).** Given any constraint network 𝒞, we can in low

▷ Symbol CNF

Ein Literal ist eine atomare Formel or die Negation einer solchen. Wir sagen, dass eine Formel eine

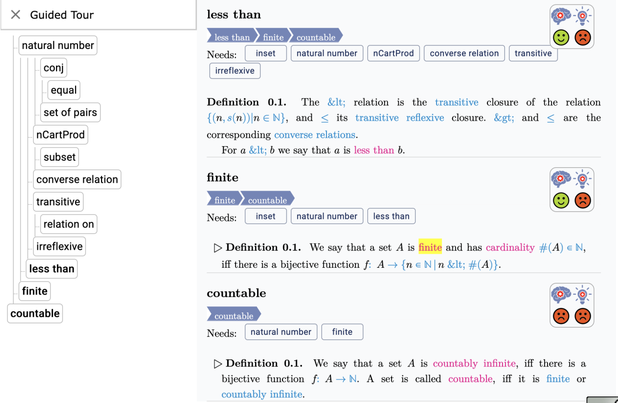
- Negationsnormalform (NNF) ist, wenn alle darin vorkommenden Negationen Literale sind.
- konjunktive Normalform (CNF) ist, wenn sie eine Konjunktion von Diskunktionen von Literalen ist.
- disjunktive Normalform (DNF) ist, wenn sie eine Disjunktion von Konjunktionen von Literalen ist.

SOME RIGHTS RESERVED

Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 5.18 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of its definition. (even works recursively)
- **Example 5.19 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.
- **Example 5.20 (Guided Tour).** A guided tour for a concept c assembles definitions/etc. into a self-contained mini-course culminating at c.

$c =$ countable $\leadsto$



less than

Definition 0.1. The &lt; relation is the transitive closure of the relation $\{(n,s(n))|n\in\mathbb{N}\}$, and $\leq$ its transitive reflexive closure. &gt; and $\leq$ are the corresponding converse relations.

For a &lt; b we say that a is less than b.

finite

▷ **Definition 0.1.** We say that a set A is finite and has cardinality $\#(A)\in\mathbb{N}$, iff there is a bijective function $f\colon A\to\{n\in\mathbb{N}\,|\,n$ &lt; $\#(A)\}$.

countable

▷ **Definition 0.1.** We say that a set A is countably infinite, iff there is a bijective function $f\colon A\to\mathbb{N}$. A set is called countable, iff it is finite or countably infinite.

Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 5.21 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of its definition. (even works recursively)
- **Example 5.22 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.
- **Example 5.23 (Guided Tour).** A guided tour for a concept c assembles definitions/etc. into a self-contained mini-course culminating at c.
- . . . your idea here . . . (the sky is the limit)

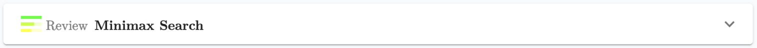
SOME RIGHTS RESERVED

(Practice/Remedial) Problems Everywhere

- **Problem:** Learning requires a mix of understanding and test-driven practice.
- **Idea:** ALeA supplies targeted practice problems everywhere.
- **Concretely:** Revision markers at the end of sections.

(Practice/Remedial) Problems Everywhere

- **Problem:** Learning requires a mix of understanding and test-driven practice.
- **Idea:** ALeA supplies targeted practice problems everywhere.
- **Concretely:** Revision markers at the end of sections.
  - A relatively non-intrusive overview over competency

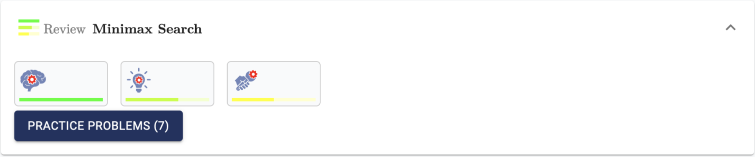


Review **Minimax Search**

SOME RIGHTS RESERVED

(Practice/Remedial) Problems Everywhere

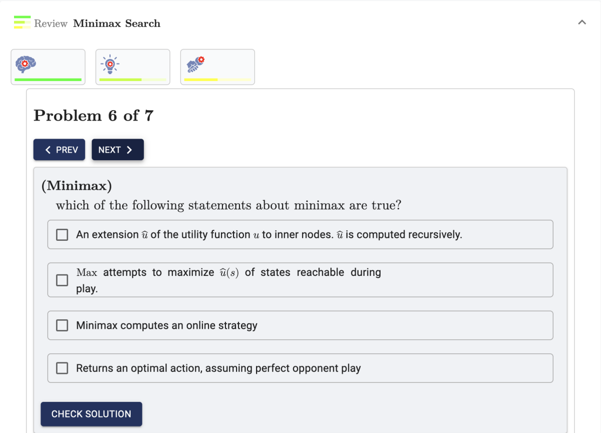
- **Problem:** Learning requires a mix of understanding and test-driven practice.
- **Idea:** ALeA supplies targeted practice problems everywhere.
- **Concretely:** Revision markers at the end of sections.
  - A relatively non-intrusive overview over competency
  - Click to extend it for details.



SOME RIGHTS RESERVED

(Practice/Remedial) Problems Everywhere

- **Problem:** Learning requires a mix of understanding and test-driven practice.
- **Idea:** ALeA supplies targeted practice problems everywhere.
- **Concretely:** Revision markers at the end of sections.
  - A relatively non-intrusive overview over competency
  - Click to extend it for details.
  - Practice problems as usual. (targeted to your specific competency)



Review **Minimax Search**

Problem 6 of 7

(Minimax)

which of the following statements about minimax are true?

☐ An extension $\hat{u}$ of the utility function u to inner nodes. $\hat{u}$ is computed recursively.

☐ Max attempts to maximize $\hat{u}(s)$ of states reachable during play.

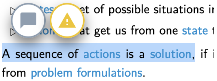
☐ Minimax computes an online strategy

☐ Returns an optimal action, assuming perfect opponent play

SOME RIGHTS RESERVED

Localized Interactions with the Community

- Selecting text brings up localized – i.e. anchored on the selection – interactions:



▷ tes et of possible situations i

▷ ion at get us from one state t

A sequence of actions is a solution, if i

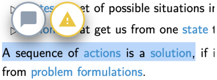
from problem formulations.

- post a (public) comment or take (private) note
- report an error to the course authors/instructors

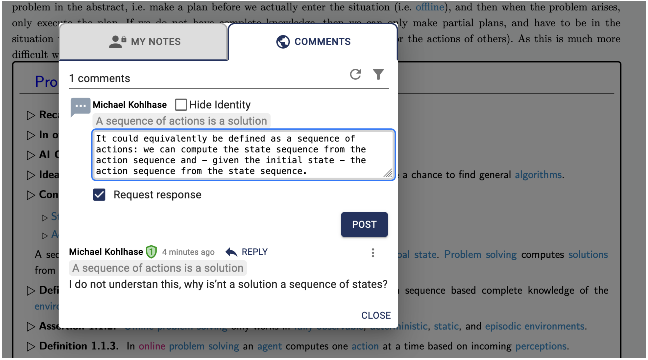
SOME RIGHTS RESERVED

Localized Interactions with the Community

- Selecting text brings up localized – i.e. anchored on the selection – interactions:


▷ ...tes ...et of possible situations ir
▷ ...ion ...at get us from one state t
A sequence of actions is a solution, if i
from problem formulations.

  - post a (public) comment or take (private) note
  - report an error to the course authors/instructors
- Localized comments induce a thread in the ALEA forum (like the StudOn Forum, but targeted towards specific learning objects.)


problem in the abstract, i.e. make a plan before we actually enter the situation (i.e. offline), and then when the problem arises, only execute the plan. If we do not have complete knowledge, then we can only make partial plans, and have to be in the situation ... r the actions of others). As this is much more difficult w...

Michael Kohlhase ☐ Hide Identity

A sequence of actions is a solution

```
It could equivalently be defined as a sequence of
actions: we can compute the state sequence from the
action sequence and – given the initial state – the
action sequence from the state sequence.
```

☑ Request response

Michael Kohlhase 4 minutes ago REPLY

A sequence of actions is a solution

I do not understan this, why is'nt a solution a sequence of states?

... a chance to find general algorithms.

A sec ... al state. Problem solving computes solutions from ...

▷ Defi ... sequence based complete knowledge of the envir...

▷ Ass... ...inistic, static, and episodic environments.

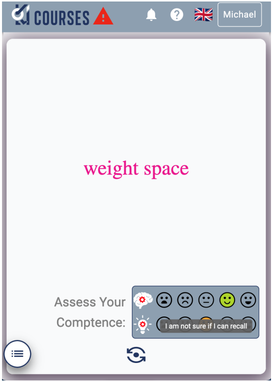
▷ **Definition 1.1.3.** In online problem solving an agent computes one action at a time based on incoming perceptions.

- Answering questions gives karma $\widehat{=}$ a public measure of user helpfulness.
- Notes can be anonymous (⇝ generate no karma)

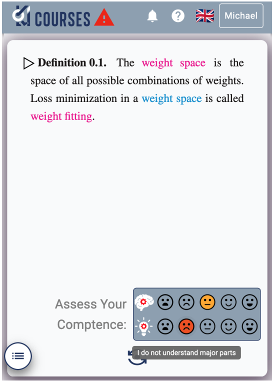
SOME RIGHTS RESERVED

New Feature: Drilling with Flashcards

- Flashcards challenge you with a task (term/problem) on the front...



weight space



▷ **Definition 0.1.** The weight space is the space of all possible combinations of weights. Loss minimization in a weight space is called weight fitting.

...and the definition/answer is on the back.

- Self-assessment updates the learner model (before/after)
- **Idea:** Challenge yourself to a card stack, keep drilling/assessing flashcards until the learner model eliminates all.
- **Bonus:** Flashcards can be generated from existing semantic markup (educational equivalent to free beer)

SOME RIGHTS RESERVED

Learner Data and Privacy in ALeA

- **Observation:** Learning support services in ALeA use the learner model; they
  - need the learner model data to adapt to the invidivual learner!
  - collect learner interaction data (to update the learner model)
- **Consequence:** You need to be logged in (via your FAU IDM credentials) for useful learning support services!

SOME RIGHTS RESERVED

Learner Data and Privacy in ALeA

- **Observation:** Learning support services in ALeA use the learner model; they
  - need the learner model data to adapt to the invidivual learner!
  - collect learner interaction data (to update the learner model)
- **Consequence:** You need to be logged in (via your FAU IDM credentials) for useful learning support services!
- **Problem:** Learner model data is highly sensitive personal data!
- **ALeA Promise:** The ALeA team does the utmost to keep your personal data safe. (SSO via FAU IDM/eduGAIN, ALeA trust zone)

SOME RIGHTS RESERVED

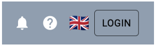
Learner Data and Privacy in ALeA

- **Observation:** Learning support services in ALeA use the learner model; they
  - need the learner model data to adapt to the invidivual learner!
  - collect learner interaction data (to update the learner model)
- **Consequence:** You need to be logged in (via your FAU IDM credentials) for useful learning support services!
- **Problem:** Learner model data is highly sensitive personal data!
- **ALeA Promise:** The ALeA team does the utmost to keep your personal data safe. (SSO via FAU IDM/eduGAIN, ALeA trust zone)
- **ALeA Privacy Axioms:**
  1. ALeA only collects learner models data about logged in users.
  2. Personally identifiable learner model data is only accessible to its subject (delegation possible)
  3. Learners can always query the learner model about its data.
  4. All learner model data can be purged without negative consequences (except usability deterioration)
  5. Logging into ALeA is completely optional.
- **Observation:** Authentication for bonus quizzes are somewhat less optional, but you can always purge the learner model later.

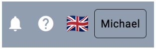
SOME RIGHTS RESERVED

Concrete Todos for ALeA

- **Recall:** You will use ALeA for the prepquizzes (or lose bonus points)
  All other use is optional. (but AI-supported pre/postparation can be helpful)
- To use the ALeA system, you will have to log in via SSO: (do it now)
  - go to `https://courses.voll-ki.fau.de/course-home/ai-2`,
  - in the upper right hand corner you see ,

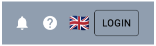
  - log in via your FAU IDM credentials. (you should have them by now)
  - You get access to your personal ALeA profile via
    (plus feature notifications, manual, and language chooser)

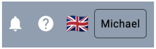
SOME RIGHTS RESERVED

Concrete Todos for ALeA

- **Recall:** You will use ALeA for the prepquizzes (or lose bonus points)
  All other use is optional. (but AI-supported pre/postparation can be helpful)
- To use the ALeA system, you will have to log in via SSO: (do it now)
  - go to `https://courses.voll-ki.fau.de/course-home/ai-2`,
  - in the upper right hand corner you see ,



  - log in via your FAU IDM credentials. (you should have them by now)
  - You get access to your personal ALeA profile via
    (plus feature notifications, manual, and language chooser)



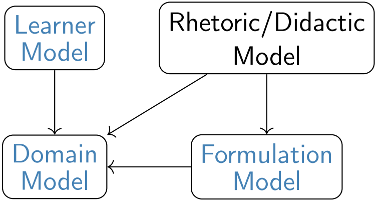
- **Problem:** Most ALeA services depend on the learner model. (to adapt to you)
- **Solution:** Initialize your learner model with your educational history!
  - **Concretely**: enter taken CS courses (FAU equivalents) and grades.
  - ALeA uses that to estimate your CS/AI competencies. (for your benefit)
  - then ALeA knows about you; I don't! (ALeA trust zone)

SOME RIGHTS RESERVED

1.6 AI-Supported Learning – How does it work?

ALeA$\widehat{=}$ Data-Driven & AI-enabled Learning Assistance

- **Idea:** Do what a teacher does!
  Use/maintain four models:



Learner Model

Rhetoric/Didactic Model

Domain Model

Formulation Model

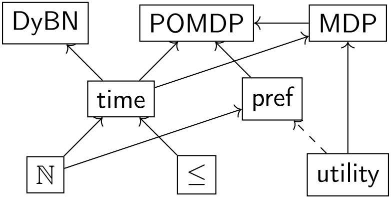
(Good) teachers

- understand the objects and their properties they are talking about
- have readimade formulations how to convey them best
- and understand how these best work together
- model what the learners already know/understand and adapts them accordingly

SOME RIGHTS RESERVED

ALeA $\hat{=}$ Data-Driven & AI-enabled Learning Assistance

- **Idea:** Do what a teacher does!
  Use/maintain four models:
- **Ingredient 1:** Domain model $\hat{=}$ knowledge/theory graph



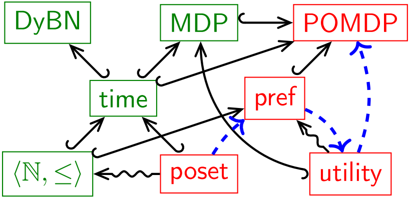
A theory graph provides (modular representation of the domain)

- symbols with URIs for all concepts, objects, and relations
- definitions, notations, and verbalizations for all symbols
- "object-oriented inheritance" and views between theories.

SOME RIGHTS RESERVED

ALeA $\widehat{=}$ Data-Driven & AI-enabled Learning Assistance

- **Idea:** Do what a teacher does!
  Use/maintain four models:
- **Ingredient 1:** Domain model $\widehat{=}$ knowledge/theory graph
- **Ingredient 2:** Learner model $\widehat{=}$ adding competency estimations

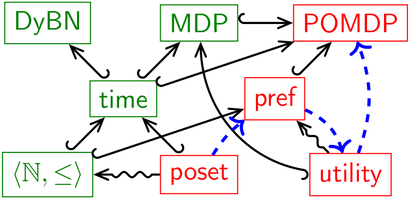


The learner model is a function from learner IDs × symbol URIs to competency values

- competency comes in six cognitive dimensions: `remember`, `understand`, `analyze`, `evaluate`, `apply`, and `create`.
- ALeA logs all learner interactions (keeps data learner-private)
- each interaction updates the learner model function.

ALeA $\widehat{=}$ Data-Driven & AI-enabled Learning Assistance

- **Idea:** Do what a teacher does! Use/maintain four models:
- **Ingredient 1:** Domain model $\widehat{=}$ knowledge/theory graph
- **Ingredient 2:** Learner model $\widehat{=}$ adding competency estimations
- **Ingredient 3:** A collection of ready-formulated learning objects



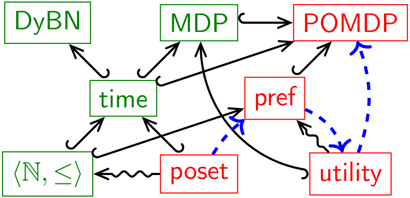
Learning objects are the text fragments learners see and interact with; they are structured by

- didactic relations, e.g. tasks have prerequisites and learning objectives
- rhetoric relations, e.g. introduction, elaboration, and transition

SOME RIGHTS RESERVED

ALeA $\widehat{=}$ Data-Driven & AI-enabled Learning Assistance

- **Idea:** Do what a teacher does! Use/maintain four models:
- **Ingredient 1:** Domain model $\widehat{=}$ knowledge/theory graph
- **Ingredient 2:** Learner model $\widehat{=}$ adding competency estimations
- **Ingredient 3:** A collection of ready-formulated learning objects
- **Ingredient 4:** Educational dialogue planner $\leadsto$ guided tours



The dialogue planner assembles learning objects into active course material using

- the domain model and didactic relations to determine the order of LOs
- the learner model to determine what to show
- the rhetoric relations to make the dialogue coherent

SOME RIGHTS RESERVED

Chapter 2
Artificial Intelligence – Who?, What?, When?, Where?, and Why?

the introduction of this chapter]21467

Plot for this chapter

- Motivation, overview, and finding out what you already know
  - What is artificial intelligence?
  - What has AI already achieved?
  - A (very) quick walk through the AI-1 topics.
  - How can you get involved with AI at KWARC?

SOME RIGHTS RESERVED

2.1 What is Artificial Intelligence?

SOME RIGHTS RESERVED

What is Artificial Intelligence? Definition

- **Definition 1.1 (According to Wikipedia).** Artificial Intelligence (AI) is intelligence exhibited by machines
- **Definition 1.2 (also).** Artificial Intelligence (AI) is a sub-field of CS that is concerned with the automation of intelligent behavior.
- **BUT:** it is already difficult to define intelligence precisely.
- **Definition 1.3 (Elaine Rich).** artificial intelligence (AI) studies how we can make the computer do things that humans can still do better at the moment.

SOME RIGHTS RESERVED

What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

the ability to learn

SOME RIGHTS RESERVED

What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

Inference

SOME RIGHTS RESERVED

What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

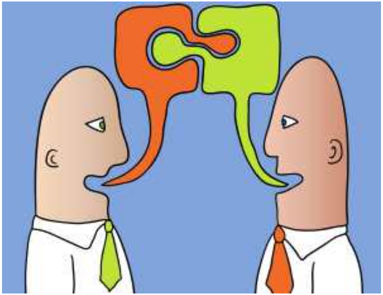
Perception

SOME RIGHTS RESERVED

What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

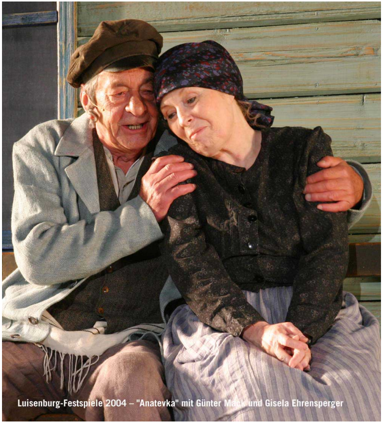
Language understanding



What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

Emotion


Luisenburg-Festspiele 2004 – "Anatevka" mit Günter Mack und Gisela Ehrensperger

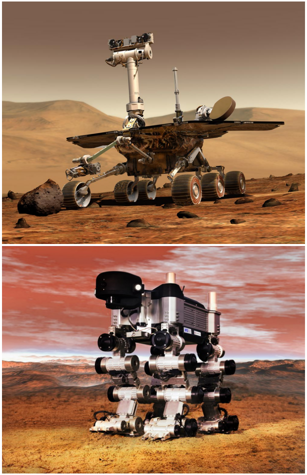
SOME RIGHTS RESERVED

2.2 Artificial Intelligence is here today!

SOME RIGHTS RESERVED

Artificial Intelligence is here today!

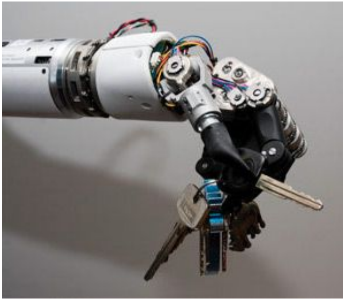
- in outer space
  - in outer space systems need autonomous control:
  - remote control impossible due to time lag
- in artificial limbs
- in household appliances
- in hospitals
- for safety/security



SOME RIGHTS RESERVED

Artificial Intelligence is here today!

- in outer space
- in artificial limbs
  - the user controls the prosthesis via existing nerves, can e.g. grip a sheet of paper.
- in household appliances
- in hospitals
- for safety/security



Artificial Intelligence is here today!

- in outer space
- in artificial limbs
- in household appliances
  - The iRobot Roomba vacuums, mops, and sweeps in corners, . . . , parks, charges, and discharges.
  - general robotic household help is on the horizon.
- in hospitals
- for safety/security

SOME RIGHTS RESERVED

Artificial Intelligence is here today!

- in outer space
- in artificial limbs
- in household appliances
- in hospitals
  - in the USA 90% of the prostate operations are carried out by RoboDoc
  - Paro is a cuddly robot that eases solitude in nursing homes.
- for safety/security

SOME RIGHTS RESERVED

Artificial Intelligence is here today!

- in outer space
- in artificial limbs
- in household appliances
- in hospitals
- for safety/security
  - e.g. Intel verifies correctness of all chips after the “Pentium 5 disaster”

© 1999 Randy Glasbergen. www.glasbergen.com

"It's the latest innovation in office safety.
When your computer crashes, an air bag is activated
so you won't bang your head in frustration."

SOME RIGHTS RESERVED

The AI Conundrum

- **Observation:** Reserving the term "artificial intelligence" has been quite a land grab!
- **But:** researchers at the Dartmouth Conference (1956) really thought they would solve/reach AI in two/three decades.
- **Consequence:** AI still asks the big questions. (and still promises answers soon)
- **Another Consequence:** AI as a field is an incubator for many innovative technologies.
- **AI Conundrum:** Once AI solves a subfield it is called "CS". (becomes a separate subfield of CS)
- **Example 2.1.** Functional/Logic Programming, automated theorem proving, Planning, machine learning, Knowledge Representation, . . .
- **Still Consequence:** AI research was alternatingly flooded with money and cut off brutally.

SOME RIGHTS RESERVED

The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.

The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.
- Funding levels are tied to public perception of success (especially for AI)

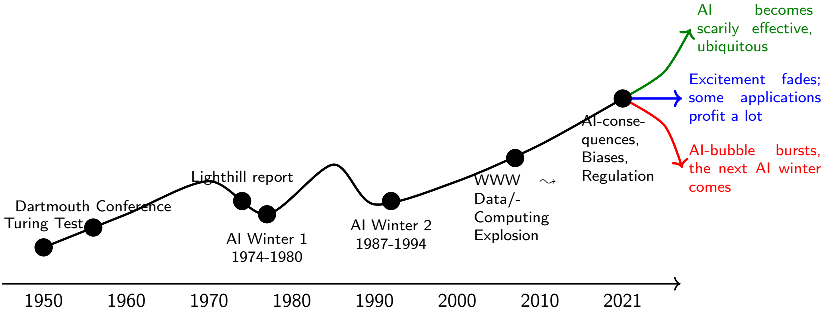
SOME RIGHTS RESERVED

The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.
- Funding levels are tied to public perception of success (especially for AI)
- **Definition 2.4.** An AI winter is a time period of low public perception and funding for AI,
  mostly because AI has failed to deliver on its – sometimes overblown – promises
  An AI summer is a time period of high public perception and funding for AI

The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.
- Funding levels are tied to public perception of success (especially for AI)
- **Definition 2.5.** An AI winter is a time period of low public perception and funding for AI,
  mostly because AI has failed to deliver on its – sometimes overblown – promises
  An AI summer is a time period of high public perception and funding for AI
- A potted history of AI (AI summers and summers)



SOME RIGHTS RESERVED

2.3 Ways to Attack the AI Problem

SOME RIGHTS RESERVED

Four Main Approaches to Artificial Intelligence

- **Definition 3.1.** Symbolic AI is a subfield of AI based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into meaning-carrying structures (expressions) and manipulating them (using processes) to produce new expressions.

SOME RIGHTS RESERVED

Four Main Approaches to Artificial Intelligence

- **Definition 3.5.** Symbolic AI is a subfield of AI based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into meaning-carrying structures (expressions) and manipulating them (using processes) to produce new expressions.
- **Definition 3.6.** Statistical AI remedies the two shortcomings of symbolic AI approaches: that all concepts represented by symbols are crisply defined, and that all aspects of the world are knowable/representable in principle. Statistical AI adopts sophisticated mathematical models of uncertainty and uses them to create more accurate world models and reason about them.

SOME RIGHTS RESERVED

Four Main Approaches to Artificial Intelligence

- **Definition 3.9.** Symbolic AI is a subfield of AI based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into meaning-carrying structures (expressions) and manipulating them (using processes) to produce new expressions.
- **Definition 3.10.** Statistical AI remedies the two shortcomings of symbolic AI approaches: that all concepts represented by symbols are crisply defined, and that all aspects of the world are knowable/representable in principle. Statistical AI adopts sophisticated mathematical models of uncertainty and uses them to create more accurate world models and reason about them.
- **Definition 3.11.** Subsymbolic AI (also called connectionism or neural AI) is a subfield of AI that posits that intelligence is inherently tied to brains, where information is represented by a simple sequence pulses that are processed in parallel via simple calculations realized by neurons, and thus concentrates on neural computing.

SOME RIGHTS RESERVED

Four Main Approaches to Artificial Intelligence

- **Definition 3.13.** Symbolic AI is a subfield of AI based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into meaning-carrying structures (expressions) and manipulating them (using processes) to produce new expressions.
- **Definition 3.14.** Statistical AI remedies the two shortcomings of symbolic AI approaches: that all concepts represented by symbols are crisply defined, and that all aspects of the world are knowable/representable in principle. Statistical AI adopts sophisticated mathematical models of uncertainty and uses them to create more accurate world models and reason about them.
- **Definition 3.15.** Subsymbolic AI (also called connectionism or neural AI) is a subfield of AI that posits that intelligence is inherently tied to brains, where information is represented by a simple sequence pulses that are processed in parallel via simple calculations realized by neurons, and thus concentrates on neural computing.
- **Definition 3.16.** Embodied AI posits that intelligence cannot be achieved by reasoning about the state of the world (symbolically, statistically, or connectivist), but must be embodied i.e. situated in the world, equipped with a "body" that can interact with it via sensors and actuators. Here, the main method for realizing intelligent behavior is by learning from the world.

SOME RIGHTS RESERVED

Two ways of reaching Artificial Intelligence?

- We can classify the AI approaches by their coverage and the analysis depth(they are complementary)

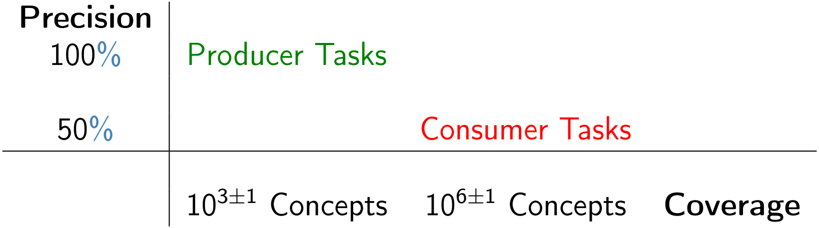
Deep	symbolic AI-1	not there yet cooperation?
Shallow	no-one wants this	statistical/sub symbolic AI-2
Analysis ↑ vs. Coverage →	Narrow	Wide

- **This semester** we will cover foundational aspects of symbolic AI (deep/narrow processing)
- **next semester** concentrate on statistical/subsymbolic AI. (shallow/wide-coverage)

SOME RIGHTS RESERVED

Environmental Niches for both Approaches to AI

- **Observation:** There are two kinds of applications/tasks in AI
  - Consumer tasks: consumer grade applications have tasks that must be fully generic and wide coverage. (e.g. machine translation like Google Translate)
  - Producer tasks: producer grade applications must be high-precision, but can be domain-specific (e.g. multilingual documentation, machinery-control, program verification, medical technology)



after Aarne Ranta [**Ranta:atcp17**].

- **General Rule:** Subsymbolic AI is well suited for consumer tasks, while symbolic AI is better suited for producer tasks.
- A domain of producer tasks I am interested in: mathematical/technical documents.

SOME RIGHTS RESERVED

2.4 Strong vs. Weak AI

SOME RIGHTS RESERVED

Strong AI vs. Narrow AI

- **Definition 4.1.** With the term narrow AI (also weak AI, instrumental AI, applied AI) we refer to the use of software to study or accomplish *specific* problem solving or reasoning tasks (e.g. playing chess/go, controlling elevators, composing music, . . .)
- **Definition 4.2.** With the term strong AI (also full AI, AGI) we denote the quest for software performing at the full range of human cognitive abilities.
- **Definition 4.3.** Problems requiring strong AI to solve are called AI hard, and AI complete, iff AGI should be able to solve them all.
- **In short:** We can characterize the difference intuitively:
  - narrow AI: What (most) computer scientists think AI is / should be.
  - strong AI: What Hollywood authors think AI is / should be.
- **Needless to say** we are only going to cover narrow AI in this course!

SOME RIGHTS RESERVED

A few words on AGI. . .

- The conceptual and mathematical framework (agents, environments is the same for strong AI and weak AI.
- AGI research focuses mostly on abstract aspects of machine learning (reinforcement learning, neural nets) and decision/game theory ("which goals should an AGI pursue?").
- Academic respectability of AGI fluctuates massively, recently increased (again). (correlates somewhat with AI winters and golden years)
- Public attention increasing due to talk of "existential risks of AI" (e.g. Hawking, Musk, Bostrom, Yudkowsky, Obama, . . .)
- **Kohlhase's View:** Weak AI is here, strong AI is very far off. (not in my lifetime)
- ⚠ **:** But even if that is true, weak AI will affect all of us deeply in everyday life.
- **Example 4.4.** You should not train to be an accountant or truck driver! (bots will replace you soon)

SOME RIGHTS RESERVED

AGI Research and Researchers

- "Famous" research(ers) / organizations
  - MIRI (Machine Intelligence Research Institute), Eliezer Yudkowsky (Formerly known as "Singularity Institute")
  - Future of Humanity Institute Oxford (Nick Bostrom),
  - Google (Ray Kurzweil),
  - AGIRI / OpenCog (Ben Goertzel),
  - `petrl.org` (People for the Ethical Treatment of Reinforcement Learners). (Obviously somewhat tongue-in-cheek)
- ⚠ Be highly skeptical about any claims with respect to AGI! (Kohlhase's View)

SOME RIGHTS RESERVED

2.5 AI Topics Covered

SOME RIGHTS RESERVED

Topics of AI-1 (Winter Semester)

- Getting Started
  - What is artificial intelligence? (situating ourselves)
  - Logic programming in Prolog (An influential paradigm)
  - Intelligent Agents (a unifying framework)
- Problem Solving
  - Problem Solving and search (Black Box World States and Actions)
  - Adversarial search (Game playing) (A nice application of search)
  - constraint satisfaction problems (Factored World States)
- Knowledge and Reasoning
  - Formal Logic as the mathematics of Meaning
  - Propositional logic and satisfiability (Atomic Propositions)
  - First-order logic and theorem proving (Quantification)
  - Logic programming (Logic + Search⇝ Programming)
  - Description logics and semantic web
- Planning
  - Planning Frameworks
  - Planning Algorithms
  - Planning and Acting in the real world

SOME RIGHTS RESERVED

Topics of AI-2 (Summer Semester)

- Uncertain Knowledge and Reasoning
  - Uncertainty
  - Probabilistic reasoning
  - Making Decisions in Episodic Environments
  - Problem Solving in Sequential Environments
- Foundations of machine learning
  - Learning from Observations
  - Knowledge in Learning
  - Statistical Learning Methods
- Communication (If there is time)
  - Natural Language Processing
  - Natural Language for Communication

SOME RIGHTS RESERVED

AI1SysProj: A Systems/Project Supplement to AI-1

- The AI-1 course concentrates on concepts, theory, and algorithms of symbolic AI.
- **Problem:** Engineering/Systems Aspects of AI are very important as well.
- **Partial Solution:** Getting your hands dirty in the homeworks and the Kalah Challenge

AI1SysProj: A Systems/Project Supplement to AI-1

- The AI-1 course concentrates on concepts, theory, and algorithms of symbolic AI.
- **Problem:** Engineering/Systems Aspects of AI are very important as well.
- **Partial Solution:** Getting your hands dirty in the homeworks and the Kalah Challenge
- **Full Solution:** AI1SysProj: AI-1 Systems Project (10 ECTS, 30-50places)
  - For each Topic of AI-1, where will be a mini-project in AI1SysProj
  - e.g. for game-play there will be Chinese Checkers (more difficult than Kalah)
  - e.g. for CSP we will schedule TechFak courses or exams (from real data)
  - solve challenges by implementing the AI-1 algorithms or use SoA systems

SOME RIGHTS RESERVED

AI1SysProj: A Systems/Project Supplement to AI-1

- The AI-1 course concentrates on concepts, theory, and algorithms of symbolic AI.
- **Problem:** Engineering/Systems Aspects of AI are very important as well.
- **Partial Solution:** Getting your hands dirty in the homeworks and the Kalah Challenge
- **Full Solution:** AI1SysProj: AI-1 Systems Project (10 ECTS, 30-50places)
  - For each Topic of AI-1, where will be a mini-project in AI1SysProj
  - e.g. for game-play there will be Chinese Checkers (more difficult than Kalah)
  - e.g. for CSP we will schedule TechFak courses or exams (from real data)
  - solve challenges by implementing the AI-1 algorithms or use SoA systems
- **Question:** Should I take AI1SysProj in my first semester? (i.e. now)

SOME RIGHTS RESERVED

AI1SysProj: A Systems/Project Supplement to AI-1

- The AI-1 course concentrates on concepts, theory, and algorithms of symbolic AI.
- **Problem:** Engineering/Systems Aspects of AI are very important as well.
- **Partial Solution:** Getting your hands dirty in the homeworks and the Kalah Challenge
- **Full Solution:** AI1SysProj: AI-1 Systems Project (10 ECTS, 30-50places)
  - For each Topic of AI-1, where will be a mini-project in AI1SysProj
  - e.g. for game-play there will be Chinese Checkers (more difficult than Kalah)
  - e.g. for CSP we will schedule TechFak courses or exams (from real data)
  - solve challenges by implementing the AI-1 algorithms or use SoA systems
- **Question:** Should I take AI1SysProj in my first semester? (i.e. now)
- **Answer:** It depends . . . (on your situation)
  - most master's programs require a 10-ECTS "Master's Project" (Master AI: two)
  - there will be a great pressure on project places (so reserve one early)
  - BUT 10 ECTS $\widehat{=}$ 250-300 hours involvement by definition (1/3 of your time/ECTS)

SOME RIGHTS RESERVED

AI1SysProj: A Systems/Project Supplement to AI-1

- The AI-1 course concentrates on concepts, theory, and algorithms of symbolic AI.
- **Problem:** Engineering/Systems Aspects of AI are very important as well.
- **Partial Solution:** Getting your hands dirty in the homeworks and the Kalah Challenge
- **Full Solution:** AI1SysProj: AI-1 Systems Project (10 ECTS, 30-50places)
    - For each Topic of AI-1, where will be a mini-project in AI1SysProj
    - e.g. for game-play there will be Chinese Checkers (more difficult than Kalah)
    - e.g. for CSP we will schedule TechFak courses or exams (from real data)
    - solve challenges by implementing the AI-1 algorithms or use SoA systems
- **Question:** Should I take AI1SysProj in my first semester? (i.e. now)
- **Answer:** It depends ... (on your situation)
    - most master's programs require a 10-ECTS "Master's Project" (Master AI: two)
    - there will be a great pressure on project places (so reserve one early)
    - BUT 10 ECTS $\widehat{=}$ 250-300 hours involvement by definition (1/3 of your time/ECTS)
- **BTW:** There will also be an AI2SysProj next semester! (another chance)

SOME RIGHTS RESERVED

2.6 AI in the KWARC Group

SOME RIGHTS RESERVED

The KWARC Research Group

- **Observation:** The ability to represent knowledge about the world and to draw logical inferences is one of the central components of intelligent behavior.
- **Thus:** reasoning components of some form are at the heart of many AI systems.
- **KWARC Angle:** Scaling up (web-coverage) without dumbing down (too much)
  - Content markup instead of full formalization (too tedious)
  - User support and quality control instead of "The Truth" (elusive anyway)
  - use Mathematics as a test tube (⚠ Mathematics $\widehat{=}$ Anything Formal ⚠)
  - care more about applications than about philosophy (we cannot help getting this right anyway as logicians)
- The KWARC group was established at Jacobs Univ. in 2004, moved to FAU Erlangen in 2016
- See `http://kwarc.info` for projects, publications, and links

SOME RIGHTS RESERVED

Overview: KWARC Research and Projects

Applications: eMath 3.0, Active Documents, Active Learning, Semantic Spreadsheets/CAD/CAM, Change Mangagement, Global Digital Math Library, Math Search Systems, SMGloM: Semantic Multilingual Math Glossary, Serious Games, . . .

Foundations of Math:

- MathML, *OpenMath*
- advanced Type Theories
- MMT: Meta Meta Theory
- Logic Morphisms/Atlas
- Theorem Prover/CAS Interoperability
- Mathematical Models/Simulation

KM & Interaction:

- Semantic Interpretation (aka. Framing)
- math-literate interaction
- MathHub: math archives & active docs
- Active documents: embedded semantic services
- Model-based Education

Semantization:

- LaTeXML: LaTeX $\leadsto$ XML
- sTeX: Semantic LaTeX
- invasive editors
- Context-Aware IDEs
- Mathematical Corpora
- Linguistics of Math
- ML for Math Semantics Extraction

Foundations: Computational Logic, Web Technologies, OMDoc/MMT

SOME RIGHTS RESERVED

Research Topics in the KWARC Group

- We are always looking for bright, motivated KWARCies.
- We have topics in for all levels! (Enthusiast, Bachelor, Master, Ph.D.)
- List of current topics: `https://gl.kwarc.info/kwarc/thesis-projects/`
  - Automated Reasoning: Maths Representation in the Large
  - Logics development, $(\text{Meta})^n$-Frameworks
  - Math Corpus Linguistics: Semantics Extraction
  - Serious Games, Cognitive Engineering, Math Information Retrieval, Legal Reasoning, …
  - …last but not least: KWARC is the home of ALeA!
- We always try to find a topic at the intersection of your and our interests.
- We also sometimes have positions!. (HiWi, Ph.D.: $\frac{1}{2}$ E-13, PostDoc: full E-13)

SOME RIGHTS RESERVED

Part 1

Getting Started with AI: A Conceptual Framework

SOME RIGHTS RESERVED

Enough philosophy about "Intelligence" (Artificial or Natural)

- So far we had a nice philosophical chat, about "intelligence" et al.
- As of today, we look at technical stuff!

SOME RIGHTS RESERVED

Enough philosophy about "Intelligence" (Artificial or Natural)

- So far we had a nice philosophical chat, about "intelligence" et al.
- As of today, we look at technical stuff!
- Before we go into the algorithms and data structures proper, we will
  1. introduce a programming language for AI-2
  2. prepare a conceptual framework in which we can think about "intelligence" (natural and artificial), and
  3. recap some methods and results from theoretical CS.

SOME RIGHTS RESERVED

Chapter 3
Logic Programming

SOME RIGHTS RESERVED

3.1 Introduction to Logic Programming and ProLog

SOME RIGHTS RESERVED

Logic Programming

- **Idea:** Use logic as a programming language!
- We state what we know about a problem (the program) and then ask for results (what the program would compute).
- **Example 1.1.**

Program	Leibniz is human Sokrates is human Sokrates is a greek Every human is fallible	$x + 0 = x$ If $x + y = z$ then $x + s(y) = s(z)$ 3 is prime
Query	Are there fallible greeks?	is there a z with $s(s(0)) + s(0) = z$
Answer	Yes, Sokrates!	yes $s(s(s(0)))$

- **How to achieve this?** Restrict a logic calculus sufficiently that it can be used as computational procedure.
- **Remark:** This idea leads a totally new programming paradigm: logic programming.
- **Slogan:** Computation = Logic + Control (Robert Kowalski 1973; [**Kowalski:alc79**])
- We will use the programming language Prolog as an example.

SOME RIGHTS RESERVED

Prolog Terms and Literals

- **Definition 1.2.** Prolog expresses knowledge about the world via
  - constants denoted by lowercase strings,
  - variables denoted by strings starting with an uppercase letter or _, and
  - functions and predicates (lowercase strings) applied to terms.
- **Definition 1.3.** A Prolog term is
  - a Prolog variable, or constant, or
  - a Prolog function applied to terms.

  A Prolog literal is a constant or a predicate applied to terms.
- **Example 1.4.** The following are
  - Prolog terms: john, X, _, father(john), . . .
  - Prolog literals: loves(john,mary), loves(john,_), loves(john,wife_of(john)),. . .

SOME RIGHTS RESERVED

Prolog Programs: Facts and Rules

- **Definition 1.5.** A Prolog program is a sequence of clauses, i.e.
  - facts of the form l., where l is a literal, (a literal and a dot)
  - rules of the form h:−b_1,. . .,b_n., where $n > 0$. h is called the head literal (or simply head) and the b_i are together called the body of the rule.

  A rule h:−b_1,. . .,b_n., should be read as "*h (is true) if b_1 and . . . and b_n are*".
- **Example 1.6.** Write "something is a car if it has a motor and four wheels" as car(X) :− has_motor(X),has_wheels(X,4). (variables are uppercase)
  This is just an ASCII notation for $m(x) \wedge w(x,4) \Rightarrow car(x)$.
- **Example 1.7.** The following is a Prolog program:

```
human(leibniz).
human(sokrates).
greek(sokrates).
fallible(X):−human(X).
```

The first three lines are Prolog facts and the last a rule.

SOME RIGHTS RESERVED

Prolog Programs: Knowledge bases

- **Intuition:** The knowledge base given by a Prolog program is the set of facts that can be derived from it under the if/and reading above.
- **Definition 1.8.** The knowledge base given by Prolog program is that set of facts that can be derived from it by Modus Ponens (MP), $\wedge I$ and instantiation.

$$\frac{A \quad A \Rightarrow B}{B}\,\mathrm{MP} \qquad \frac{A \quad B}{A \wedge B}\,\wedge I \qquad \frac{\mathsf{A}}{[\mathsf{B}/X](\mathsf{A})}\,\mathrm{Subst}$$

Querying the Knowledge Base: Size Matters

- **Idea:** We want to see whether a fact is in the knowledge base.
- **Definition 1.9.** A query is a list of Prolog literals called goal literals (also subgoals or simply goals). We write a query as ?−$A_1, \ldots, A_n$. where A_i are goals.
- **Problem:** Knowledge bases can be big and even infinite. (cannot pre-compute)
- **Example 1.10.** The knowledge base induced by the Prolog program

```
nat(zero).
nat(s(X)) :− nat(X).
```

  contains the facts nat(zero), nat(s(zero)), nat(s(s(zero))), . . .

Querying the Knowledge Base: Backchaining

- **Definition 1.11.** Given a query Q: $?\!-\ A_1, \ldots, A_n$. and rule R: $h\!:\!-\ b_1,\ldots,b_n$, backchaining computes a new query by
  1. finding terms for all variables in h to make h and A_1 equal and
  2. replacing A_1 in Q with the body literals of R, where all variables are suitably replaced.
- Backchaining motivates the names goal/subgoal:
  - the literals in the query are “goals” that have to be satisfied,
  - backchaining does that by replacing them by new “goals”.
- **Definition 1.12.** The Prolog interpreter keeps backchaining from the top to the bottom of the program until the query
  - succeeds, i.e. contains no more goals, or (answer: **true**)
  - fails, i.e. backchaining becomes impossible. (answer: false)
- **Example 1.13 (Backchaining).** We continue 1.10

```
?- nat(s(s(zero))).
?- nat(s(zero)).
?- nat(zero).
true
```

SOME RIGHTS RESERVED

Querying the Knowledge Base: Failure

- If no instance of a query can be derived from the knowledge base, then the Prolog interpreter reports failure.
- **Example 1.14.** We vary 1.13 using 0 instead of zero.

```
?– nat(s(s(0))).
?– nat(s(0)).
?– nat(0).
FAIL
false
```

SOME RIGHTS RESERVED

Querying the Knowledge base: Answer Substitutions

- **Definition 1.15.** If a query contains variables, then Prolog will return an answer substitution as the result to the query, i.e the values for all the query variables accumulated during repeated backchaining.
- **Example 1.16.** We talk about (Bavarian) cars for a change, and use a query with a variables

```
has_wheels(mybmw,4).
has_motor(mybmw).
car(X):−has_wheels(X,4),has_motor(X).
?− car(Y) % query
?− has_wheels(Y,4),has_motor(Y). % substitution X = Y
?− has_motor(mybmw). % substitution Y = mybmw
Y = mybmw % answer substitution
true
```

SOME RIGHTS RESERVED

PROLOG: Are there Fallible Greeks?

- **Program:**

```
human(leibniz).
human(sokrates).
greek(sokrates).
fallible(X):−human(X).
```

- **Example 1.17 (Query).** ?−fallible(X),greek(X).
- **Answer substitution:** $[\mathrm{sokrates}/X]$

SOME RIGHTS RESERVED

3.2 Programming as Search

SOME RIGHTS RESERVED

3.2.1 Knowledge Bases and Backtracking

SOME RIGHTS RESERVED

Depth-First Search with Backtracking

- So far, all the examples led to direct success or to failure. (simple KB)
- **Definition 2.1 (Prolog Search Procedure).** The Prolog interpreter employs top-down, left-right depth first search, concretely, Prolog search:
  - works on the subgoals in left right order.
  - matches first query with the head literals of the clauses in the program in top-down order.
  - if there are no matches, fail and backtracks to the (chronologically) last backtrack point.
  - otherwise backchain on the first match, keep the other matches in mind for backtracking via backtrack points.

  We say that a goal G matches a head H, iff we can make them equal by replacing variables in H with terms.
- We can force backtracking to compute more answers by typing ;.

SOME RIGHTS RESERVED

Backtracking by Example

- **Example 2.2.** We extend ???:

```
has_wheels(mytricycle,3).
has_wheels(myrollerblade,3).
has_wheels(mybmw,4).
has_motor(mybmw).
car(X):-has_wheels(X,3),has_motor(X). % cars sometimes have three wheels
car(X):-has_wheels(X,4),has_motor(X). % and sometimes four.
?- car(Y).
?- has_wheels(Y,3),has_motor(Y). % backtrack point 1
Y = mytricycle % backtrack point 2
?- has_motor(mytricycle).
FAIL % fails, backtrack to 2
Y = myrollerblade % backtrack point 2
?- has_motor(myrollerblade).
FAIL % fails, backtrack to 1
?- has_wheels(Y,4),has_motor(Y).
Y = mybmw
?- has_motor(mybmw).
Y=mybmw
true
```

SOME RIGHTS RESERVED

3.2.2 Programming Features

SOME RIGHTS RESERVED

Can We Use This For Programming?

- **Question:** What about functions? E.g. the addition function?
- **Question:** We cannot define functions, in Prolog!
- **Idea (back to math):** use a three-place predicate.
- **Example 2.3.** add(X,Y,Z) stands for X+Y=Z
- Now we can directly write the recursive equations $X + 0 = X$ (base case) and $X + s(Y) = s(X + Y)$ into the knowledge base.

```
add(X,zero,X).
add(X,s(Y),s(Z)) :− add(X,Y,Z).
```

- Similarly with multiplication and exponentiation.

```
mult(X,zero,zero).
mult(X,s(Y),Z) :− mult(X,Y,W), add(X,W,Z).

expt(X,zero,s(zero)).
expt(X,s(Y),Z) :− expt(X,Y,W), mult(X,W,Z).
```

More Examples from elementary Arithmetic

- **Example 2.4.** We can also use the add relation for subtraction without changing the implementation. We just use variables in the "input positions" and ground terms in the other two. (possibly very inefficient "generate and test approach")

```
?−add(s(zero),X,s(s(s(zero)))).
X = s(s(zero))
true
```

- **Example 2.5.** Computing the nth Fibonacci number (0, 1, 1, 2, 3, 5, 8, 13,. . . ; add the last two to get the next), using the addition predicate above.

```
fib(zero,zero).
fib(s(zero),s(zero)).
fib(s(s(X)),Y):−fib(s(X),Z),fib(X,W),add(Z,W,Y).
```

- **Example 2.6.** Using Prolog's internal floating-point arithmetic: a goal of the form ?− D **is**e. — where e is a ground arithmetic expression binds D to the result of evaluating e.

```
fib(0,0).
fib(1,1).
fib(X,Y):− D is X − 1, E is X − 2,fib(D,Z),fib(E,W), Y is Z + W.
```

SOME RIGHTS RESERVED

Adding Lists to Prolog

- **Definition 2.7.** In Prolog, lists are represented by list terms of the form
  1. [a,b,c,...] for list literals, and
  2. a first/rest constructor that represents a list with head F and rest list R as [F|R].
- **Observation:** Just as in functional programming, we can define list operations by recursion, only that we program with relations instead of with functions.
- **Example 2.8.** Predicates for member, append and reverse of lists in default Prolog representation.

```
member(X,[X|_]).
member(X,[_|R]):−member(X,R).

append([],L,L).
append([X|R],L,[X|S]):−append(R,L,S).

reverse([],[]).
reverse([X|R],L):−reverse(R,S),append(S,[X],L).
```

SOME RIGHTS RESERVED

Relational Programming Techniques

- **Example 2.9.** Parameters have no unique direction “in” or “out”

```
?- rev(L,[1,2,3]).
?- rev([1,2,3],L1).
?- rev([1|X],[2|Y]).
```

- **Example 2.10.** Symbolic programming by structural induction:

```
rev([],[]).
rev([X|Xs],Ys) :- ...
```

- **Example 2.11.** Generate and test:

```
sort(Xs,Ys) :- perm(Xs,Ys), ordered(Ys).
```

SOME RIGHTS RESERVED

3.2.3 Advanced Relational Programming

Specifying Control in Prolog

- *Remark 2.12.* The running time of the program from 2.11 is not $\mathcal{O}(n\log_2(n))$ which is optimal for sorting algorithms.

  ```
  sort(Xs,Ys) :− perm(Xs,Ys), ordered(Ys).
  ```

- **Idea:** Gain computational efficiency by shaping the search!

SOME RIGHTS RESERVED

Functions and Predicates in Prolog

- *Remark 2.13.* Functions and predicates have radically different roles in Prolog.
  - Functions are used to represent data. (e.g. father(john) or s(s(zero)))
  - Predicates are used for stating properties about and computing with data.
- *Remark 2.14.* In functional programming, functions are used for both. (even more confusing than in Prolog if you think about it)

SOME RIGHTS RESERVED

Functions and Predicates in Prolog

- *Remark 2.17.* Functions and predicates have radically different roles in Prolog.
  - Functions are used to represent data. (e.g. father(john) or s(s(zero)))
  - Predicates are used for stating properties about and computing with data.
- *Remark 2.18.* In functional programming, functions are used for both.
  (even more confusing than in Prolog if you think about it)
- **Example 2.19.** Consider again the reverse predicate for lists below:
  An input datum is e.g. [1,2,3], then the output datum is [3,2,1].

```
reverse([],[]).
reverse([X|R],L):−reverse(R,S),append(S,[X],L).
```

We “define” the computational behavior of the predicate rev, but the list constructors [. . .] are just used to construct lists from arguments.

SOME RIGHTS RESERVED

Functions and Predicates in Prolog

- *Remark 2.21.* Functions and predicates have radically different roles in Prolog.
  - Functions are used to represent data. (e.g. father(john) or s(s(zero)))
  - Predicates are used for stating properties about and computing with data.
- *Remark 2.22.* In functional programming, functions are used for both.
  (even more confusing than in Prolog if you think about it)
- **Example 2.23.** Consider again the reverse predicate for lists below:
  An input datum is e.g. [1,2,3], then the output datum is [3,2,1].

```
reverse([],[]).
reverse([X|R],L):−reverse(R,S),append(S,[X],L).
```

  We “define” the computational behavior of the predicate rev, but the list constructors [. . .] are just used to construct lists from arguments.
- **Example 2.24 (Trees and Leaf Counting).** We represent (unlabelled) trees via the function t from tree lists to trees. For instance, a balanced binary tree of depth 2 is t([t([t([]),t([])]),t([t([]),t([])])]). We count leaves by

```
leafcount(t([]),1).
leafcount(t([V]),W) :− leafcount(V,W).
leafcount(t([X|R]),Y) :− leafcount(X,Z), leafcount(t(R),W), Y is Z + W.
```

SOME RIGHTS RESERVED

For more information on Prolog

RTFM ($\widehat{=}$ “read the fine manuals”)

- **RTFM Resources:** There are also lots of good tutorials on the web,
  - I personally like [**Fisher:pt:on**; **LPN:on**],
  - [**Flach:SL94**] has a very thorough logic-based introduction,
  - consult also the SWI Prolog Manual [**SWIPL-manual:on**],

SOME RIGHTS RESERVED

Chapter 4
Recap of Prerequisites from Math & Theoretical Computer Science

SOME RIGHTS RESERVED

4.1 Recap: Complexity Analysis in AI?

SOME RIGHTS RESERVED

Performance and Scaling

- Suppose we have three algorithms to choose from. (which one to select)
- Systematic analysis reveals performance characteristics.
- **Example 1.1.** For a computational problem of size n we have

	performance		
size	linear	quadratic	exponential
n	$100n\mu s$	$7n^2\mu s$	$2^n\mu s$
1	$100\mu s$	$7\mu s$	$2\mu s$
5	$.5\mathrm{ms}$	$175\mu s$	$32\mu s$
10	$1\mathrm{ms}$	$.7\mathrm{ms}$	$1\mathrm{ms}$
45	$4.5\mathrm{ms}$	$14\mathrm{ms}$	$1.1Y$
100	...	...	...
1 000	...	...	...
10 000	...	...	...
1 000 000	...	...	...

SOME RIGHTS RESERVED

What?! One year?

- $2^{10} = 1\,024$ $(1024\mu\mathrm{s} \simeq 1\mathrm{ms})$
- $2^{45} = 35\,184\,372\,088\,832$ $(3.5{\times}10^{13}\mu\mathrm{s} \simeq 3.5{\times}10^{7}\mathrm{s} \simeq 1.1Y)$
- **Example 1.2.** We denote all times that are longer than the age of the universe with —

	performance		
size	linear	quadratic	exponential
n	$100n\mu\mathrm{s}$	$7n^2\mu\mathrm{s}$	$2^n\mu\mathrm{s}$
1	$100\mu\mathrm{s}$	$7\mu\mathrm{s}$	$2\mu\mathrm{s}$
5	$.5\mathrm{ms}$	$175\mu\mathrm{s}$	$32\mu\mathrm{s}$
10	$1\mathrm{ms}$	$.7\mathrm{ms}$	$1\mathrm{ms}$
45	$4.5\mathrm{ms}$	$14\mathrm{ms}$	$1.1Y$
< 100	$100\mathrm{ms}$	$7\mathrm{s}$	$10^{16}Y$
1 000	$1\mathrm{s}$	$12\mathrm{min}$	—
10 000	$10\mathrm{s}$	$20\mathrm{h}$	—
1 000 000	$1.6\mathrm{min}$	$2.5\mathrm{mon}$	—

SOME RIGHTS RESERVED

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-2.

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-2.
- **Definition 1.6.** We say that an algorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha) := t$.

  Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that α has time complexity in S (written $T(\alpha) \in S$ or colloquially $T(\alpha) = S$), iff $t \in S$. We say α has space complexity in S, iff α uses only memory of size $s(n)$ on inputs of size n and $s \in S$.

SOME RIGHTS RESERVED

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-2.
- **Definition 1.9.** We say that an algorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha) := t$.

  Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that α has time complexity in S (written $T(\alpha) \in S$ or colloquially $T(\alpha) = S$), iff $t \in S$. We say α has space complexity in S, iff α uses only memory of size $s(n)$ on inputs of size n and $s \in S$.
- Time/space complexity depends on size measures. (no canonical one)

SOME RIGHTS RESERVED

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-2.
- **Definition 1.12.** We say that an algorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha) := t$.

  Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that α has time complexity in S (written $T(\alpha) \in S$ or colloquially $T(\alpha) = S$), iff $t \in S$. We say α has space complexity in S, iff α uses only memory of size $s(n)$ on inputs of size n and $s \in S$.
- Time/space complexity depends on size measures. (no canonical one)
- **Definition 1.13.** The following sets are often used for S in $T(\alpha)$:

Landau set	class name	rank	Landau set	class name	rank
$\mathcal{O}(1)$	constant	1	$\mathcal{O}(n^2)$	quadratic	4
$\mathcal{O}(\log_2(n))$	logarithmic	2	$\mathcal{O}(n^k)$	polynomial	5
$\mathcal{O}(n)$	linear	3	$\mathcal{O}(k^n)$	exponential	6

  where $\mathcal{O}(g) = \{f \mid \exists k > 0. f \leq_a k \cdot g\}$ and $f \leq_a g$ (f is asymptotically bounded by g), iff there is an $n_0 \in \mathbb{N}$, such that $f(n) \leq g(n)$ for all $n > n_0$.

SOME RIGHTS RESERVED

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-2.
- **Definition 1.15.** We say that an algorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha) := t$.

  Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that α has time complexity in S (written $T(\alpha) \in S$ or colloquially $T(\alpha) = S$), iff $t \in S$. We say α has space complexity in S, iff α uses only memory of size $s(n)$ on inputs of size n and $s \in S$.
- Time/space complexity depends on size measures. (no canonical one)
- **Definition 1.16.** The following sets are often used for S in $T(\alpha)$:

Landau set	class name	rank	Landau set	class name	rank
$\mathcal{O}(1)$	constant	1	$\mathcal{O}(n^2)$	quadratic	4
$\mathcal{O}(\log_2(n))$	logarithmic	2	$\mathcal{O}(n^k)$	polynomial	5
$\mathcal{O}(n)$	linear	3	$\mathcal{O}(k^n)$	exponential	6

  where $\mathcal{O}(g) = \{f \mid \exists k > 0. f \leq_a k \cdot g\}$ and $f \leq_a g$ (f is asymptotically bounded by g), iff there is an $n_0 \in \mathbb{N}$, such that $f(n) \leq g(n)$ for all $n > n_0$.
- **Lemma 1.17 (Growth Ranking).** *For $k' > 2$ and $k > 1$ we have*

$$\mathcal{O}(1) \subset \mathcal{O}(\log_2(n)) \subset \mathcal{O}(n) \subset \mathcal{O}(n^2) \subset \mathcal{O}(n^{k'}) \subset \mathcal{O}(k^n)$$

SOME RIGHTS RESERVED

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-2.
- **Definition 1.18.** We say that an algorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha) := t$.
  Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that α has time complexity in S (written $T(\alpha) \in S$ or colloquially $T(\alpha) = S$), iff $t \in S$. We say α has space complexity in S, iff α uses only memory of size $s(n)$ on inputs of size n and $s \in S$.
- Time/space complexity depends on size measures. (no canonical one)
- **Definition 1.19.** The following sets are often used for S in $T(\alpha)$:

Landau set	class name	rank	Landau set	class name	rank
$\mathcal{O}(1)$	constant	1	$\mathcal{O}(n^2)$	quadratic	4
$\mathcal{O}(\log_2(n))$	logarithmic	2	$\mathcal{O}(n^k)$	polynomial	5
$\mathcal{O}(n)$	linear	3	$\mathcal{O}(k^n)$	exponential	6

  where $\mathcal{O}(g) = \{f \mid \exists k > 0. f \leq_a k \cdot g\}$ and $f \leq_a g$ (f is asymptotically bounded by g), iff there is an $n_0 \in \mathbb{N}$, such that $f(n) \leq g(n)$ for all $n > n_0$.
- **Lemma 1.20 (Growth Ranking).** *For $k' > 2$ and $k > 1$ we have*

$$\mathcal{O}(1) \subset \mathcal{O}(\log_2(n)) \subset \mathcal{O}(n) \subset \mathcal{O}(n^2) \subset \mathcal{O}(n^{k'}) \subset \mathcal{O}(k^n)$$

- **For AI-2:** I expect that given an algorithm, you can determine its complexity class. (next)

SOME RIGHTS RESERVED

Advantage: Big-Oh Arithmetics

- **Practical Advantage:** Computing with Landau sets is quite simple. (good simplification)
- **Theorem 1.21 (Computing with Landau Sets).**
  1. *If* $\mathcal{O}(c \cdot f) = \mathcal{O}(f)$ *for any constant* $c \in \mathbb{N}$. *(drop constant factors)*
  2. *If* $\mathcal{O}(f) \subseteq \mathcal{O}(g)$, *then* $\mathcal{O}(f + g) = \mathcal{O}(g)$. *(drop low-complexity summands)*
  3. *If* $\mathcal{O}(f \cdot g) = \mathcal{O}(f) \cdot \mathcal{O}(g)$. *(distribute over products)*
- These are not all of "big-Oh calculation rules", but they're enough for most purposes
- **Applications:** Convince yourselves using the result above that
  - $\mathcal{O}(4n^3 + 3n + 7^{1000n}) = \mathcal{O}(2^n)$
  - $\mathcal{O}(n) \subset \mathcal{O}(n \cdot \log_2(n)) \subset \mathcal{O}(n^2)$

Determining the Time/Space Complexity of Algorithms

- **Definition 1.22.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: can be accessed in constant time

Determining the Time/Space Complexity of Algorithms

- **Definition 1.23.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.24.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: need the complexity of the value

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.25.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.26.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: compose the complexities of the function and the argument

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.27.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.28.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: has to compute the value $\rightsquigarrow$ has its complexity

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.29.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v := \varphi$ with $T_\Gamma(\varphi) \in S$, then $T_\Gamma(\alpha) \in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v, S)$.

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.30.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v := \varphi$ with $T_\Gamma(\varphi) \in S$, then $T_\Gamma(\alpha) \in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v, S)$.
  - composition: has the maximal complexity of the components

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.31.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v{:=}\varphi$ with $T_\Gamma(\varphi) \in S$, then $T_\Gamma(\alpha) \in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v,S)$.
  - composition: If α is $\varphi\,;\,\psi$, with $T_\Gamma(\varphi) \in P$ and $T_{\Gamma \cup C_\Gamma(\psi)}(\psi) \in Q$, then $T_\Gamma(\alpha) \in \max\{P, Q\}$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\psi)}(\psi)$.

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.32.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v := \varphi$ with $T_\Gamma(\varphi) \in S$, then $T_\Gamma(\alpha) \in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v,S)$.
  - composition: If α is $\varphi\,;\,\psi$, with $T_\Gamma(\varphi) \in P$ and $T_{\Gamma \cup C_\Gamma(\psi)}(\psi) \in Q$, then $T_\Gamma(\alpha) \in \max\{P, Q\}$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\psi)}(\psi)$.
  - branching: has the maximal complexity of the condition and branches

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.33.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v := \varphi$ with $T_\Gamma(\varphi) \in S$, then $T_\Gamma(\alpha) \in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v, S)$.
  - composition: If α is $\varphi\,;\psi$, with $T_\Gamma(\varphi) \in P$ and $T_{\Gamma \cup C_\Gamma(\psi)}(\psi) \in Q$, then $T_\Gamma(\alpha) \in \max\{P, Q\}$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\psi)}(\psi)$.
  - branching: If α is $\mathbf{if}\gamma\mathbf{then}\varphi\mathbf{else}\psi\mathbf{end}$, with $T_\Gamma(\gamma) \in C$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in P$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in Q$, and then $T_\Gamma(\alpha) \in \max\{C, P, Q\}$ and $C_\Gamma(\alpha) = \Gamma \cup C_\Gamma(\gamma) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\psi)$.

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.34.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v{:=}\varphi$ with $T_\Gamma(\varphi) \in S$, then $T_\Gamma(\alpha) \in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v,S)$.
  - composition: If α is $\varphi\,;\psi$, with $T_\Gamma(\varphi) \in P$ and $T_{\Gamma \cup C_\Gamma(\psi)}(\psi) \in Q$, then $T_\Gamma(\alpha) \in \max\{P, Q\}$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\psi)}(\psi)$.
  - branching: If α is $\mathbf{if}\gamma\mathbf{then}\varphi\mathbf{else}\psi\mathbf{end}$, with $T_\Gamma(\gamma) \in C$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in P$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in Q$, and then $T_\Gamma(\alpha) \in \max\{C, P, Q\}$ and $C_\Gamma(\alpha) = \Gamma \cup C_\Gamma(\gamma) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\psi)$.
  - looping: multiplies complexities

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.35.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v := \varphi$ with $T_\Gamma(\varphi) \in S$, then $T_\Gamma(\alpha) \in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v, S)$.
  - composition: If α is $\varphi\, ;\, \psi$, with $T_\Gamma(\varphi) \in P$ and $T_{\Gamma \cup C_\Gamma(\psi)}(\psi) \in Q$, then $T_\Gamma(\alpha) \in \max\{P, Q\}$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\psi)}(\psi)$.
  - branching: If α is **if**γ**then**φ**else**ψ**end**, with $T_\Gamma(\gamma) \in C$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in P$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in Q$, and then $T_\Gamma(\alpha) \in \max\{C, P, Q\}$ and $C_\Gamma(\alpha) = \Gamma \cup C_\Gamma(\gamma) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\psi)$.
  - looping: If α is **while**γ**do**φ**end**, with $T_\Gamma(\gamma) \in \mathcal{O}(f)$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f(n) \cdot g(n))$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi)$.

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.36.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha)\in\mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha)\in\mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi)\in\mathcal{O}(f)$ and $T_{\Gamma\cup C_\Gamma(\varphi)}(\psi)\in\mathcal{O}(g)$, then $T_\Gamma(\alpha)\in\mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma\cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v{:=}\varphi$ with $T_\Gamma(\varphi)\in S$, then $T_\Gamma(\alpha)\in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v,S)$.
  - composition: If α is $\varphi\,;\psi$, with $T_\Gamma(\varphi)\in P$ and $T_{\Gamma\cup C_\Gamma(\psi)}(\psi)\in Q$, then $T_\Gamma(\alpha)\in\max\,\{P, Q\}$ and $C_\Gamma(\alpha) = C_{\Gamma\cup C_\Gamma(\psi)}(\psi)$.
  - branching: If α is **if**γ**then**φ**else**ψ**end**, with $T_\Gamma(\gamma)\in C$, $T_{\Gamma\cup C_\Gamma(\gamma)}(\varphi)\in P$, $T_{\Gamma\cup C_\Gamma(\gamma)}(\varphi)\in Q$, and then $T_\Gamma(\alpha)\in\max\,\{C, P, Q\}$ and $C_\Gamma(\alpha) = \Gamma \cup C_\Gamma(\gamma) \cup C_{\Gamma\cup C_\Gamma(\gamma)}(\varphi) \cup C_{\Gamma\cup C_\Gamma(\gamma)}(\psi)$.
  - looping: If α is **while**γ**do**φ**end**, with $T_\Gamma(\gamma)\in\mathcal{O}(f)$, $T_{\Gamma\cup C_\Gamma(\gamma)}(\varphi)\in\mathcal{O}(g)$, then $T_\Gamma(\alpha)\in\mathcal{O}(f(n)\cdot g(n))$ and $C_\Gamma(\alpha) = C_{\Gamma\cup C_\Gamma(\gamma)}(\varphi)$.
  - The time complexity $T(\alpha)$ is just $T_\emptyset(\alpha)$, where $\emptyset$ is the empty function.

SOME RIGHTS RESERVED

Determining the Time/Space Complexity of Algorithms

- **Definition 1.37.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: If $\alpha = v$ with $v \in \mathrm{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: If α is $v{:=}\varphi$ with $T_\Gamma(\varphi) \in S$, then $T_\Gamma(\alpha) \in S$ and $C_\Gamma(\alpha) = \Gamma \cup (v, S)$.
  - composition: If α is $\varphi\,;\,\psi$, with $T_\Gamma(\varphi) \in P$ and $T_{\Gamma \cup C_\Gamma(\psi)}(\psi) \in Q$, then $T_\Gamma(\alpha) \in \max\{P, Q\}$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\psi)}(\psi)$.
  - branching: If α is **if**γ**then**φ**else**ψ**end**, with $T_\Gamma(\gamma) \in C$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in P$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in Q$, and then $T_\Gamma(\alpha) \in \max\{C, P, Q\}$ and $C_\Gamma(\alpha) = \Gamma \cup C_\Gamma(\gamma) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\psi)$.
  - looping: If α is **while**γ**do**φ**end**, with $T_\Gamma(\gamma) \in \mathcal{O}(f)$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f(n) \cdot g(n))$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi)$.
  - The time complexity $T(\alpha)$ is just $T_\emptyset(\alpha)$, where $\emptyset$ is the empty function.
- Recursion is much more difficult to analyze ⇝ recurrences and Master's theorem.

SOME RIGHTS RESERVED

Why Complexity Analysis? (General)

- **Example 1.38.** Once upon a time I was trying to invent an efficient algorithm.
  - My first algorithm attempt didn't work, so I had to try harder.

SOME RIGHTS RESERVED

Why Complexity Analysis? (General)

- **Example 1.39.** Once upon a time I was trying to invent an efficient algorithm.
  - My first algorithm attempt didn't work, so I had to try harder.
  - But my 2nd attempt didn't work either, which got me a bit agitated.

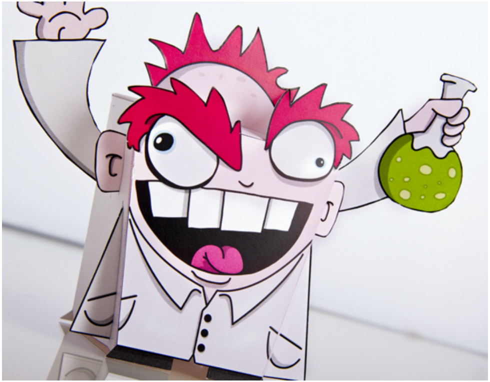
Why Complexity Analysis? (General)

- **Example 1.40.** Once upon a time I was trying to invent an efficient algorithm.
  - My first algorithm attempt didn't work, so I had to try harder.
  - But my 2nd attempt didn't work either, which got me a bit agitated.
  - The 3rd attempt didn't work either. . .

SOME RIGHTS RESERVED

Why Complexity Analysis? (General)

- **Example 1.41.** Once upon a time I was trying to invent an efficient algorithm.
  - My first algorithm attempt didn't work, so I had to try harder.
  - But my 2nd attempt didn't work either, which got me a bit agitated.
  - The 3rd attempt didn't work either...
  - And neither the 4th. But then:



SOME RIGHTS RESERVED

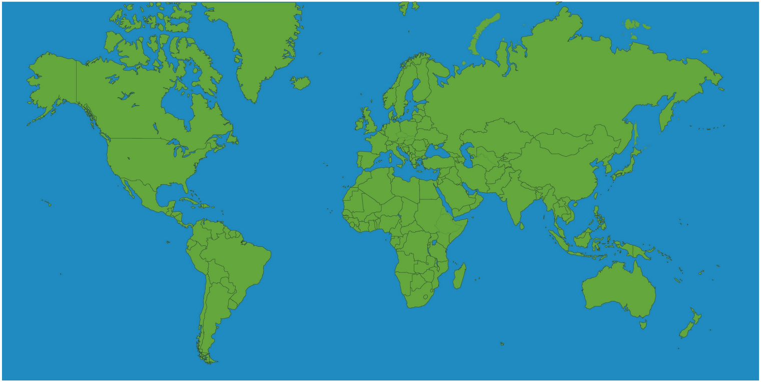
Why Complexity Analysis? (General)

- **Example 1.42.** Once upon a time I was trying to invent an efficient algorithm.
  - My first algorithm attempt didn't work, so I had to try harder.
  - But my 2nd attempt didn't work either, which got me a bit agitated.
  - The 3rd attempt didn't work either. . .
  - And neither the 4th. But then:
  - Ta-da . . . when, for once, I turned around and looked in the other direction– CAN one actually solve this efficiently? – NP-hard was there to rescue me.

SOME RIGHTS RESERVED

Why Complexity Analysis? (General)

- **Example 1.43.** Trying to find a sea route east to India (from Spain) (does not exist)



- **Observation:** Complexity theory saves you from spending lots of time trying to invent algorithms that do not exist.

SOME RIGHTS RESERVED

Reminder (?): NP and PSPACE (details $\rightsquigarrow$ e.g. [**garey:johnson:79**])

- **Turing Machine:** Works on a tape consisting of cells, across which its Read/Write head moves. The machine has internal states. There is a Turing machine program that specifies – given the current cell content and internal state – what the subsequent internal state will be, how what the R/W head does (write a symbol and/or move). Some internal states are accepting.
- Decision problems are in NP if there is a non deterministic Turing machine that halts with an answer after time polynomial in the size of its input. Accepts if *at least one* of the possible runs accepts.
- Decision problems are in NPSPACE, if there is a non deterministic Turing machine that runs in space polynomial in the size of its input.
- **NP vs. PSPACE:** Non-deterministic polynomial space can be simulated in deterministic polynomial space. Thus PSPACE $=$ NPSPACE, and hence (trivially) NP $\subseteq$ PSPACE.
  It is commonly believed that NP$\not\supseteq$PSPACE. (similar to P $\subseteq$ NP)

SOME RIGHTS RESERVED

The Utility of Complexity Knowledge (NP-Hardness)

- **Assume:** In 3 years from now, you have finished your studies and are working in your first industry job. Your boss Mr. X gives you a problem and says "*Solve It!*". By which he means, "*write a program that solves it efficiently*".
- **Question:** Assume further that, after trying in vain for 4 weeks, you got the next meeting with Mr. X. How could knowing about NP-hard problems help?

SOME RIGHTS RESERVED

The Utility of Complexity Knowledge (NP-Hardness)

- **Assume:** In 3 years from now, you have finished your studies and are working in your first industry job. Your boss Mr. X gives you a problem and says "*Solve It!*". By which he means, "*write a program that solves it efficiently*".
- **Question:** Assume further that, after trying in vain for 4 weeks, you got the next meeting with Mr. X. How could knowing about NP-hard problems help?
- **Answer:** It helps you save your skin with (theoretical computer) science!
  - Do you want to say "*Um, sorry, but I couldn't find an efficient solution, please don't fire me*"?
  - Or would you rather say "*Look, I didn't find an efficient solution. But neither could all the Turing-award winners out there put together, because the problem is NP-hard*"?

SOME RIGHTS RESERVED

4.2 Recap: Formal Languages and Grammars

SOME RIGHTS RESERVED

The Mathematics of Strings

- **Definition 2.1.** An alphabet A is a finite set; we call each element $a \in A$ a character, and an n tuple $s \in A^n$ a string (of length n over A).
- **Definition 2.2.** Note that $A^0 = \{\langle\rangle\}$, where $\langle\rangle$ is the (unique) 0-tuple. With the definition above we consider $\langle\rangle$ as the string of length 0 and call it the empty string and denote it with ϵ.
- **Note:** Sets $\neq$ strings, e.g. $\{1,2,3\} = \{3,2,1\}$, but $\langle 1,2,3\rangle \neq \langle 3,2,1\rangle$.
- **Notation:** We will often write a string $\langle c_1, \ldots, c_n\rangle$ as $''\mathtt{c_1 \ldots c_n}''$, for instance $''\mathtt{abc}''$ for $\langle \mathrm{a}, \mathrm{b}, \mathrm{c}\rangle$
- **Example 2.3.** Take $A = \{\mathtt{h}, \mathtt{1}, /\}$ as an alphabet. Each of the members $\mathtt{h}$, $\mathtt{1}$, and $/$ is a character. The vector $\langle /, /, \mathtt{1}, \mathtt{h}, \mathtt{1}\rangle$ is a string of length 5 over A.
- **Definition 2.4 (String Length).** Given a string s we denote its length with $|s|$.
- **Definition 2.5.** The concatenation $\mathrm{conc}(s,t)$ of two strings $s = \langle s_1, ..., s_n\rangle \in A^n$ and $t = \langle t_1, ..., t_m\rangle \in A^m$ is defined as $\langle s_1, ..., s_n, t_1, ..., t_m\rangle \in A^{n+m}$.
  We will often write $\mathrm{conc}(s,t)$ as $s+t$ or simply st
- **Example 2.6.** $\mathrm{conc}(''\mathtt{text}'', ''\mathtt{book}'') = ''\mathtt{text}'' + ''\mathtt{book}'' = ''\mathtt{textbook}''$

SOME RIGHTS RESERVED

Formal Languages

- **Definition 2.7.** Let A be an alphabet, then we define the sets $A^+ := \bigcup_{i \in \mathbb{N}^+} A^i$ of nonempty string and $A^* := A^+ \cup \{\epsilon\}$ of strings.
- **Example 2.8.** If $A = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, then $A^* = \{\epsilon, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{aa}, \mathrm{ab}, \mathrm{ac}, \mathrm{ba}, \ldots, \mathrm{aaa}, \ldots\}$.
- **Definition 2.9.** A set $L \subseteq A^*$ is called a formal language over A.
- **Definition 2.10.** We use $\mathrm{c}^{[n]}$ for the string that consists of the character c repeated n times.
- **Example 2.11.** $\#^{[5]} = \langle \#, \#, \#, \#, \# \rangle$
- **Example 2.12.** The set $M := \{\mathrm{ba}^{[n]} \mid n \in \mathbb{N}\}$ of strings that start with character b followed by an arbitrary numbers of a's is a formal language over $A = \{\mathrm{a}, \mathrm{b}\}$.
- **Definition 2.13.** Let $L_1, L_2, L \subseteq \Sigma^*$ be formal languages over Σ.
  - Intersection and union: $L_1 \cap L_2$, $L_1 \cup L_2$.
  - Language complement L: $\overline{L} := \Sigma^* \backslash L$.
  - The language concatenation of L_1 and L_2: $L_1 L_2 := \{uw \mid u \in L_1, w \in L_2\}$. We often use $L_1 L_2$ instead of $L_1 L_2$.
  - Language power L: $L^0 := \{\epsilon\}$, $L^{n+1} := L L^n$, where $L^n := \{\mathtt{w_1} \ldots \mathtt{w_n} \mid w_i \in L, \text{ for } i = 1 \ldots n\}$, (for $n \in \mathbb{N}$).
  - language Kleene closure L: $L^* := \bigcup_{n \in \mathbb{N}} L^n$ and also $L^+ := \bigcup_{n \in \mathbb{N}^+} L^n$.
  - The reflection of a language L: $L^R := \{w^R \mid w \in L\}$.

SOME RIGHTS RESERVED

Phrase Structure Grammars (Theory)

- **Recap:** A formal language is an arbitrary set of symbol sequences.
- **Problem:** This may be infinite and even undecidable even if A is finite.
- **Idea:** Find a way of representing formal languages with structure finitely.
- **Definition 2.14.** A phrase structure grammar (also called type 0 grammar, unrestricted grammar, or just grammar) is a tuple $\langle N, \Sigma, P, S\rangle$ where
  - N is a finite set of nonterminal symbols,
  - Σ is a finite set of terminal symbols, members of $\Sigma \cup N$ are called symbols.
  - P is a finite set of production rules: pairs $p := h \rightarrow b$ (also written as $h \Rightarrow b$), where $h \in (\Sigma \cup N)^* N (\Sigma \cup N)^*$ and $b \in (\Sigma \cup N)^*$. The string h is called the head of p and b the body.
  - $S \in N$ is a distinguished symbol called the start symbol (also sentence symbol).

  The sets N and Σ are assumed to be disjoint. Any word $w \in \Sigma^*$ is called a terminal word.
- **Intuition:** Production rules map strings with at least one nonterminal to arbitrary other strings.
- **Notation:** If we have n rules $h \rightarrow b_i$ sharing a head, we often write $h \rightarrow b_1 \mid \ldots \mid b_n$ instead.

SOME RIGHTS RESERVED

Phrase Structure Grammars (cont.)

- **Example 2.15.** A simple phrase structure grammar G:

$$\begin{array}{rcl} S & \rightarrow & NP\ Vi \\ NP & \rightarrow & Article\ N \\ Article & \rightarrow & \text{the} \mid \text{a} \mid \text{an} \\ N & \rightarrow & \text{dog} \mid \text{teacher} \mid \ldots \\ Vi & \rightarrow & \text{sleeps} \mid \text{smells} \mid \ldots \end{array}$$

  Here S, is the start symbol, NP, $Article$, N, and Vi are nonterminals.
- **Definition 2.16.** A production rule whose head is a single non-terminal and whose body consists of a single terminal is called lexical or a lexical insertion rule.
  Definition 2.17. The subset of lexical rules of a grammar G is called the lexicon of G and the set of body symbols the vocabulary (or alphabet). The nonterminals in their heads are called lexical categories of G.
- **Definition 2.18.** The non-lexicon production rules are called structural, and the nonterminals in the heads are called phrasal or syntactic categories.

SOME RIGHTS RESERVED

Phrase Structure Grammars (Theory)

- **Idea:** Each symbol sequence in a formal language can be analyzed/generated by the grammar.
- **Definition 2.19.** Given a phrase structure grammar $G := \langle N, \Sigma, P, S\rangle$, we say G derives $t \in (\Sigma \cup N)^*$ from $s \in (\Sigma \cup N)^*$ in one step, iff there is a production rule $p \in P$ with $p = h \rightarrow b$ and there are $u, v \in (\Sigma \cup N)^*$, such that $s = suhv$ and $t = ubv$. We write $s \rightarrow_G^p t$ (or $s \rightarrow_G t$ if p is clear from the context) and use $\rightarrow_G^*$ for the reflexive transitive closure of $\rightarrow_G$. We call $s \rightarrow^*{}_G t$ a G derivation of t from s.
- **Definition 2.20.** Given a phrase structure grammar $G := \langle N, \Sigma, P, S\rangle$, we say that $s \in (N \cup \Sigma)^*$ is a sentential form of G, iff $S \rightarrow^*{}_G s$. A sentential form that does not contain nontermials is called a sentence of G, we also say that G accepts s. We say that G rejects s, iff it is not a sentence of G.
- **Definition 2.21.** The language $\mathrm{L}(G)$ of G is the set of its sentences. We say that $\mathrm{L}(G)$ is generated by G.

  Definition 2.22. We call two grammars equivalent, iff they have the same languages.

  Definition 2.23. A grammar G is said to be universal if $\mathrm{L}(G) = \Sigma^*$.
- **Definition 2.24.** Parsing, syntax analysis, or syntactic analysis is the process of analyzing a string of symbols, either in a formal or a natural language by means of a grammar.

SOME RIGHTS RESERVED

Phrase Structure Grammars (Example)

- **Example 2.25.** In the grammar G from 2.15:

1. *Article* teacher *Vi* is a sentential form,

$$\begin{aligned} S &\rightarrow_G NP\ Vi \\ &\rightarrow_G Article\ N\ Vi \\ &\rightarrow_G Article\ \text{teacher}\ Vi \end{aligned}$$

2. "*The teacher sleeps*" is a sentence.

$$\begin{aligned} S &\rightarrow_G^* Article\ \text{teacher}\ Vi \\ &\rightarrow_G \text{the teacher}\ Vi \\ &\rightarrow_G \text{the teacher sleeps} \end{aligned}$$

$$\begin{aligned} S &\rightarrow NP\ Vi \\ NP &\rightarrow Article\ N \\ Article &\rightarrow \text{the} \mid \text{a} \mid \text{an} \mid \ldots \\ N &\rightarrow \text{dog} \mid \text{teacher} \mid \ldots \\ Vi &\rightarrow \text{sleeps} \mid \text{smells} \mid \ldots \end{aligned}$$

SOME RIGHTS RESERVED

Grammar Types (Chomsky Hierarchy [Chomsky:st57])

- **Observation:** The shape of the grammar determines the “size” of its language.
- **Definition 2.26.** We call a grammar:
  1. context-sensitive (or type 1), if the bodies of production rules have no less symbols than the heads,
  2. context-free (or type 2), if the heads have exactly one symbol,
  3. regular (or type 3), if additionally the bodies are empty or consist of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular (or type 1/type 2/type 3 respectively), iff it is the language of a respective grammar. Context-free grammars are sometimes CFGs and context-free languages CFLs.

SOME RIGHTS RESERVED

Grammar Types (Chomsky Hierarchy [Chomsky:st57])

- **Observation:** The shape of the grammar determines the “size” of its language.
- **Definition 2.30.** We call a grammar:
  1. context-sensitive (or type 1), if the bodies of production rules have no less symbols than the heads,
  2. context-free (or type 2), if the heads have exactly one symbol,
  3. regular (or type 3), if additionally the bodies are empty or consist of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular (or type 1/type 2/type 3 respectively), iff it is the language of a respective grammar. Context-free grammars are sometimes CFGs and context-free languages CFLs.
- **Example 2.31 (Context-sensitive).** The language $\{a^{[n]}b^{[n]}c^{[n]}\}$ is accepted by

$$\begin{array}{rcl} S & \rightarrow & \mathrm{a\ b\ c} \mid A \\ A & \rightarrow & \mathrm{a}\ A\ B\ \mathrm{c} \mid \mathrm{a\ b\ c} \\ \mathrm{c}\ B & \rightarrow & B\ \mathrm{c} \\ \mathrm{b}\ B & \rightarrow & \mathrm{b\ b} \end{array}$$

SOME RIGHTS RESERVED

Grammar Types (Chomsky Hierarchy [**Chomsky:st57**])

- **Observation:** The shape of the grammar determines the “size” of its language.
- **Definition 2.34.** We call a grammar:
  1. context-sensitive (or type 1), if the bodies of production rules have no less symbols than the heads,
  2. context-free (or type 2), if the heads have exactly one symbol,
  3. regular (or type 3), if additionally the bodies are empty or consist of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular (or type 1/type 2/type 3 respectively), iff it is the language of a respective grammar. Context-free grammars are sometimes CFGs and context-free languages CFLs.
- **Example 2.35 (Context-sensitive).** The language $\{a^{[n]}b^{[n]}c^{[n]}\}$
- **Example 2.36 (Context-free).** The language $\{a^{[n]}b^{[n]}\}$ is accepted by $S \rightarrow \mathrm{a}\ S\ \mathrm{b} \mid \epsilon$.

SOME RIGHTS RESERVED

Grammar Types (Chomsky Hierarchy [**Chomsky:st57**])

- **Observation:** The shape of the grammar determines the “size” of its language.
- **Definition 2.38.** We call a grammar:
  1. context-sensitive (or type 1), if the bodies of production rules have no less symbols than the heads,
  2. context-free (or type 2), if the heads have exactly one symbol,
  3. regular (or type 3), if additionally the bodies are empty or consist of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular (or type 1/type 2/type 3 respectively), iff it is the language of a respective grammar. Context-free grammars are sometimes CFGs and context-free languages CFLs.
- **Example 2.39 (Context-sensitive).** The language $\{a^{[n]}b^{[n]}c^{[n]}\}$
- **Example 2.40 (Context-free).** The language $\{a^{[n]}b^{[n]}\}$
- **Example 2.41 (Regular).** The language $\{a^{[n]}\}$ is accepted by $S \rightarrow S$ a

SOME RIGHTS RESERVED

Grammar Types (Chomsky Hierarchy [Chomsky:st57])

- **Observation:** The shape of the grammar determines the "size" of its language.
- **Definition 2.42.** We call a grammar:
  1. context-sensitive (or type 1), if the bodies of production rules have no less symbols than the heads,
  2. context-free (or type 2), if the heads have exactly one symbol,
  3. regular (or type 3), if additionally the bodies are empty or consist of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular (or type 1/type 2/type 3 respectively), iff it is the language of a respective grammar. Context-free grammars are sometimes CFGs and context-free languages CFLs.
- **Example 2.43 (Context-sensitive).** The language $\{a^{[n]}b^{[n]}c^{[n]}\}$
- **Example 2.44 (Context-free).** The language $\{a^{[n]}b^{[n]}\}$
- **Example 2.45 (Regular).** The language $\{a^{[n]}\}$
- **Observation:** Natural languages are probably context-sensitive but parsable in real time! (like languages low in the hierarchy)

SOME RIGHTS RESERVED

Useful Extensions of Phrase Structure Grammars

- **Definition 2.46.** The Bachus Naur form or Backus normal form (BNF) is a metasyntax notation for context-free grammars.
  It extends the body of a production rule by mutiple (admissible) constructors:
  - alternative: $s_1 \mid \ldots \mid s_n$,
  - repetition: s^* (arbitrary many s) and s^+ (at least one s),
  - optional: $[s]$ (zero or one times),
  - grouping: $(s_1 ; \ldots ; s_n)$, useful e.g. for repetition,
  - character sets: $[s{-}t]$ (all characters c with $s \leq c \leq t$ for a given ordering on the characters), and
  - complements: $[^\wedge s_1, \ldots, s_n]$, provided that the base alphabet is finite.
- **Observation:** All of these can be eliminated, .e.g (⇝ many more rules)
  - replace $X \rightarrow Z\ (s^*)\ W$ with the production rules $X \rightarrow Z\ Y\ W$, $Y \rightarrow \epsilon$, and $Y \rightarrow Y\ s$.
  - replace $X \rightarrow Z\ (s^+)\ W$ with the production rules $X \rightarrow Z\ Y\ W$, $Y \rightarrow s$, and $Y \rightarrow Y\ s$.

SOME RIGHTS RESERVED

An Grammar Notation for AI-2

- **Problem:** In grammars, notations for nonterminal symbols should be
  - short and mnemonic (for the use in the body)
  - close to the official name of the syntactic category (for the use in the head)
- In AI-2 we will only use context-free grammars (simpler, but problem still applies)
- **in AI-2:** I will try to give “grammar overviews” that combine those, e.g. the grammar of first-order logic.

variables	X	$\in$	$\mathcal{V}_1$	
function constants	f^k	$\in$	Σ_k^f	
predicate constants	p^k	$\in$	${\Sigma^p}_k$	
terms	t	$::=$	X	variable
		$\mid$	f^0	constant
		$\mid$	$f^k(t_1, \ldots, t_k)$	application
formulae	A	$::=$	$p^k(t_1, \ldots, t_k)$	atomic
		$\mid$	$\neg \mathrm{A}$	negation
		$\mid$	$\mathrm{A}_1 \wedge \mathrm{A}_2$	conjunction
		$\mid$	$\forall X.\mathrm{A}$	quantifier

SOME RIGHTS RESERVED

4.3 Mathematical Language Recap

SOME RIGHTS RESERVED

Mathematical Structures

- **Observation:** Mathematicians often cast classes of complex objects as mathematical structures.
- We have just seen an example of a mathematical structure: (repeated here for convenience)
- **Definition 3.1.** A phrase structure grammar (also called type 0 grammar, unrestricted grammar, or just grammar) is a tuple $\langle N, \Sigma, P, S\rangle$ where
  - N is a finite set of nonterminal symbols,
  - Σ is a finite set of terminal symbols, members of $\Sigma \cup N$ are called symbols.
  - P is a finite set of production rules: pairs $p := h \rightarrow b$ (also written as $h \Rightarrow b$), where $h \in (\Sigma \cup N)^* N (\Sigma \cup N)^*$ and $b \in (\Sigma \cup N)^*$. The string h is called the head of p and b the body.
  - $S \in N$ is a distinguished symbol called the start symbol (also sentence symbol).

  The sets N and Σ are assumed to be disjoint. Any word $w \in \Sigma^*$ is called a terminal word.
- **Intuition:** All grammars share structure: they have four components, which again share struccture, which is further described in the definition above.
- **Observation:** Even though we call production rules "pairs" above, they are also mathematical structures $\langle h, b\rangle$ with a funny notation $h \rightarrow b$.

SOME RIGHTS RESERVED

Mathematical Structures in Programming

- **Observation:** Most programming languages have some way of creating “named structures”. Referencing components is usually done via “dot notation”.

SOME RIGHTS RESERVED

Mathematical Structures in Programming

- **Observation:** Most programming languages have some way of creating "named structures". Referencing components is usually done via "dot notation".
- **Example 3.4 (Structs in `C`).** `C` data structures for representing grammars:

```
struct grule {
  char[][] head;
  char[][] body;
}
struct grammar {
  char[][] nterminals;
  char[][] termininals;
  grule[] grules;
  char[] start;
}
int main() {
  struct grule r1;
  r1.head = "foo";
  r1.body = "bar";
}
```

SOME RIGHTS RESERVED

Mathematical Structures in Programming

- **Observation:** Most programming languages have some way of creating “named structures”. Referencing components is usually done via “dot notation”.
- **Example 3.6 (Structs in `C`).** `C` data structures for representing grammars:

```
struct grule {
  char[][] head;
  char[][] body;
}
struct grammar {
  char[][] nterminals;
  char[][] termininals;
  grule[] grules;
  char[] start;
}
int main() {
  struct grule r1;
  r1.head = "foo";
  r1.body = "bar";
}
```

- **Example 3.7 (Classes in OOP).** Classes in object-oriented programming languages are based on the same ideas as mathematical structures, only that OOP adds powerful inheritance mechanisms.

SOME RIGHTS RESERVED

In AI-2 we use a mixture between Math and Programming Styles

- In AI-2 we use mathematical notation, ...
- **Definition 3.8.** A structure signature combines the components, their "types", and accessor names of a mathematical structure in a tabular overview.
- **Example 3.9.**

$$\text{grammar} = \left\langle \begin{array}{lll} N & \text{Set} & \text{nonterminal symbols}, \\ \Sigma & \text{Set} & \text{terminal symbols}, \\ P & \{h \to b \mid \ldots\} & \text{production rules}, \\ S & N & \text{start symbol} \end{array} \right\rangle$$

$$\text{production rule} \quad h \to b = \left\langle \begin{array}{lll} h & (\Sigma \cup N)^*, N, (\Sigma \cup N)^* & \text{head}, \\ b & (\Sigma \cup N)^* & \text{body} \end{array} \right\rangle$$

Read the first line "N Set nonterminal symbols" in the structure above as "N is in an (unspecified) set and is a nonterminal symbol".
Here – and in the future – we will use Set for the class of sets $\leadsto$ "N is a set".

- I will try to give structure signatures where necessary.

SOME RIGHTS RESERVED

Chapter 5

Rational Agents: a Unifying Framework for Artificial Intelligence

SOME RIGHTS RESERVED

5.1 Introduction: Rationality in Artificial Intelligence

SOME RIGHTS RESERVED

What is AI? Going into Details

- **Recap:** AI studies how we can make the computer do things that humans can still do better at the moment. (humans are proud to be rational)
- **What is AI?:** Four possible answers/facets: Systems that

think like humans	think rationally
act like humans	act rationally

expressed by four different definitions/quotes:

	Humanly	**Rational**
Thinking	"*The exciting new effort to make computers think . . . machines with human-like minds*" **[Haugeland:aitvi85]**	"*The formalization of mental faculties in terms of computational models*" **[ChaMcD:itai85]**
Acting	"*The art of creating machines that perform actions requiring intelligence when performed by people*" **[Kurzweil:taoim90]**	"*The branch of CS concerned with the automation of appropriate behavior in complex situations*" **[LugStu:aisscps93]**

- **Idea:** Rationality is performance-oriented rather than based on imitation.

SOME RIGHTS RESERVED

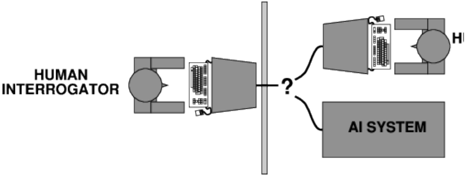
So, what does modern AI do?

- **Acting Humanly:** Turing test, not much pursued outside Loebner prize
  - $\widehat{=}$ building pigeons that can fly so much like real pigeons that they can fool pigeons
  - Not reproducible, not amenable to mathematical analysis
- **Thinking Humanly:** $\rightsquigarrow$ Cognitive Science.
  - How do humans think? How does the (human) brain work?
  - Neural networks are a (extremely simple so far) approximation
- **Thinking Rationally:** Logics, Formalization of knowledge and inference
  - You know the basics, we do some more, fairly widespread in modern AI
- **Acting Rationally:** How to make good action choices?
  - Contains logics (one possible way to make intelligent decisions)
  - We are interested in making good choices in practice (e.g. in AlphaGo)

SOME RIGHTS RESERVED

Acting humanly: The Turing test

- Introduced by Alan Turing (1950) "Computing machinery and intelligence" [**Turing:cmi50**]:
- "Can machines think?" ⟶ "Can machines behave intelligently?"
- **Definition 1.1.** The Turing test is an operational test for intelligent behavior based on an imitation game over teletext (arbitrary topic)



- It was predicted that by 2000, a machine might have a 30% chance of fooling a lay person for 5 minutes.
- **Note:** In [**Turing:cmi50**], Alan Turing
  - anticipated all major arguments against AI in following 50 years and
  - suggested major components of AI: knowledge, reasoning, language understanding, learning
- **Problem:** Turing test is not reproducible, constructive, or amenable to mathematical analysis!

SOME RIGHTS RESERVED

Thinking humanly: Cognitive Science

- **1960s:** "cognitive revolution": information processing psychology replaced prevailing orthodoxy of behaviorism.
- Requires scientific theories of internal activities of the brain
- What level of abstraction? "Knowledge" or "circuits"?
- **How to validate?:** Requires
  1. Predicting and testing behavior of human subjects or (top-down)
  2. Direct identification from neurological data. (bottom-up)
- **Definition 1.2.** Cognitive science is the interdisciplinary, scientific study of the mind and its processes. It examines the nature, the tasks, and the functions of cognition.
- **Definition 1.3.** Cognitive neuroscience studies the biological processes and aspects that underlie cognition, with a specific focus on the neural connections in the brain which are involved in mental processes.
- Both approaches/disciplines are now distinct from AI.
- Both share with AI the following characteristic: *the available theories do not explain (or engender) anything resembling human-level general intelligence*
- Hence, all three fields share one principal direction!

SOME RIGHTS RESERVED

Thinking rationally: Laws of Thought

- Normative (or prescriptive) rather than descriptive
- Aristotle: what are correct arguments/thought processes?
- Several Greek schools developed various forms of logic: *notation* and *rules of derivation* for thoughts; may or may not have proceeded to the idea of mechanization.
- Direct line through mathematics and philosophy to modern AI
- **Problems:**
  1. Not all intelligent behavior is mediated by logical deliberation
  2. What is the purpose of thinking? What thoughts *should* I have out of all the thoughts (logical or otherwise) that I *could* have?

SOME RIGHTS RESERVED

Acting Rationally

- **Idea:** Rational behavior $\widehat{=}$ doing the right thing!
- **Definition 1.4.** Rational behavior consists of always doing what is expected to maximize goal achievement given the available information.
- Rational behavior does not necessarily involve thinking e.g., blinking reflex — but thinking should be in the service of rational action.
- **Aristotle:** *Every art and every inquiry, and similarly every action and pursuit, is thought to aim at some good.* (Nicomachean Ethics)

SOME RIGHTS RESERVED

The Rational Agents

- **Definition 1.5.** A (natural) agent is an entity that perceives and acts.
- **Central Idea:** This course is about designing (artificial) agent that exhibit rational behavior, i.e. for any given class of environments and tasks, we seek the agent (or class of agents) with the best performance.
- **Caveat:** *Computational limitations make perfect rationality unachievable* ⇝ design best program for given machine resources.

SOME RIGHTS RESERVED

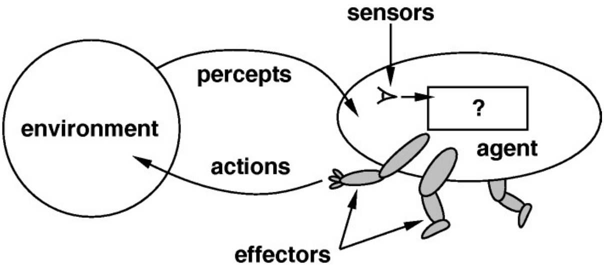
5.2 Agents and Environments as a Framework for AI

SOME RIGHTS RESERVED

Agents and Environments

- **Definition 2.1.** An agent is anything that
  - perceives its environment via sensors (a means of sensing the environment)
  - acts on it with actuators (means of changing the environment).

  Any recognizable, coherent employment of the actuators of an agent is called an action.



- **Example 2.2.** Agents include humans, robots, softbots, thermostats, etc.
- **Remark:** The notion of an agent and its environment is intentionally designed to be inclusive. We will classify and discuss subclasses of both later.

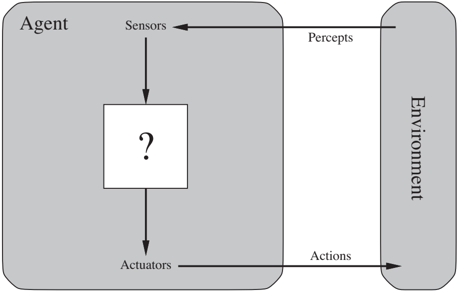
Modeling Agents Mathematically and Computationally

- **Definition 2.3.** A percept is the perceptual input of an agent at a specific time instant.
- **Definition 2.4.** Any recognizable, coherent employment of the actuators of an agent is called an action.
- **Definition 2.5.** An agent $A := \langle \mathcal{P}, \mathcal{A}, f \rangle$ consists of
  1. A set $\mathcal{P}$ of percepts,
  2. a set $\mathcal{A}$ of actions, and
  3. an agent function $f \colon \mathcal{P}^* \to \mathcal{A}$ that maps from percept histories to actions.
- We assume that agents can always perceive their own actions. (but not necessarily their consequences)
- **Problem:** Agent functions can become very big and may be uncomputable. (theoretical tool only)
- **Definition 2.6.** An agent function can be implemented by an agent program that runns on a (physical or hypothetical) agent architecture.

SOME RIGHTS RESERVED

Agent Schema: Visualizing the Internal Agent Structure

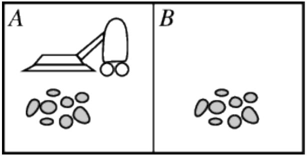
- **Agent Schema:** We will use the following kind of agent schema to visualize the internal structure of an agent:



Different agents differ on the contents of the white box in the center.

SOME RIGHTS RESERVED

Example: Vacuum-Cleaner World and Agent



- percepts: location and contents, e.g., $[A, Dirty]$
- actions: *Left*, *Right*, *Suck*, *NoOp*

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean]$, $[A, Clean]$	*Right*
$[A, Clean]$, $[A, Dirty]$	*Suck*
$[A, Clean]$, $[B, Clean]$	*Left*
$[A, Clean]$, $[B, Dirty]$	*Suck*
$[A, Dirty]$, $[A, Clean]$	*Right*
$[A, Dirty]$, $[A, Dirty]$	*Suck*
$\vdots$	$\vdots$
$[A, Clean]$, $[A, Clean]$, $[A, Clean]$	*Right*
$[A, Clean]$, $[A, Clean]$, $[A, Dirty]$	*Suck*
$\vdots$	$\vdots$

- **Science Question:** What is the *right* agent function?
- **AI Question:** Is there an agent architecture and agent program that implements it.

SOME RIGHTS RESERVED

Table-Driven Agents

- **Idea:** We can just implement the agent function as a lookup table and lookup actions.
- We can directly implement this:

```
function Table−Driven−Agent(percept) returns an action
  persistent table /∗ a table of actions indexed by percept sequences ∗/
  var percepts /∗ a sequence, initially empty ∗/
  append percept to the end of percepts
  action := lookup(percepts, table)
  return action
```

- **Problem:** Why is this not a good idea?
  - The table is much too large: even with n binary percepts whose order of occurrence does not matter, we have 2^n rows in the table.
  - Who is supposed to write this table anyways, even if it “only” has a million entries?

SOME RIGHTS RESERVED

Example: Vacuum-Cleaner Agent Program

- A much better implementation idea is to trigger actions from specific percepts.
- **Example 2.7 (Agent Program).**

```
procedure Reflex−Vacuum−Agent [location,status] returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

- This is the kind of agent programs we will be looking for in AI-2.

SOME RIGHTS RESERVED

5.3 Good Behavior $\rightsquigarrow$ Rationality

Rationality

- **Idea:** Try to design agents that are successful! (aka. "do the right thing")
- **Problem:** What do we mean by "successful", how do we measure "success"?
- **Definition 3.1.** A performance measure is a function that evaluates a sequence of environments.
- **Example 3.2.** A performance measure for a vacuum cleaner could
  - award one point per "square" cleaned up in time T?
  - award one point per clean "square" per time step, minus one per move?
  - penalize for $> k$ dirty squares?
- **Definition 3.3.** An agent is called rational, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date.
- **Critical Observation:** We only need to maximize the expected value, not the actual value of the performance measure!
- **Question:** Why is rationality a good quality to aim for?

SOME RIGHTS RESERVED

Consequences of Rationality: Exploration, Learning, Autonomy

- **Note:** A rational agent need not be perfect:
  - It only needs to maximize expected value (rational $\neq$ omniscient)
    - need not predict e.g. very unlikely but catastrophic events in the future
  - Percepts may not supply all relevant information (rational $\neq$ clairvoyant)
    - if we cannot perceive things we do not need to react to them.
    - but we may need to try to find out about hidden dangers (exploration)
  - Action outcomes may not be as expected (rational $\neq$ successful)
    - but we may need to take action to ensure that they do (more often) (learning)
- **Note:** Rationality may entail exploration, learning, autonomy (depending on the environment / task)
- **Definition 3.4.** An agent is called autonomous, if it does not rely on the prior knowledge about the environment of the designer.
- Autonomy avoids fixed behaviors that can become unsuccessful in a changing environment. (anything else would be irrational)
- The agent may have to learn all relevant traits, invariants, properties of the environment and actions.

SOME RIGHTS RESERVED

PEAS: Describing the Task Environment

- **Observation:** To design a rational agent, we must specify the task environment in terms of performance measure, environment, actuators, and sensors, together called the PEAS components.
- **Example 3.5.** When designing an automated taxi:
  - **Performance measure:** safety, destination, profits, legality, comfort, . . .
  - **Environment:** US streets/freeways, traffic, pedestrians, weather, . . .
  - **Actuators:** steering, accelerator, brake, horn, speaker/display, . . .
  - **Sensors:** video, accelerometers, gauges, engine sensors, keyboard, GPS, . . .
- **Example 3.6 (Internet Shopping Agent).** The task environment:
  - Performance measure: price, quality, appropriateness, efficiency
  - Environment: current and future WWW sites, vendors, shippers
  - Actuators: display to user, follow URL, fill in form
  - Sensors: HTML pages (text, graphics, scripts)

SOME RIGHTS RESERVED

Examples of Agents: PEAS descriptions

Agent Type	Performance measure	Environment	Actuators	Sensors
Chess/Go player	win/loose/draw	game board	moves	board position
Medical diagnosis system	accuracy of diagnosis	patient, staff	display questions, diagnoses	keyboard entry of symptoms
Part-picking robot	percentage of parts in correct bins	conveyor belt with parts, bins	jointed arm and hand	camera, joint angle sensors
Refinery controller	purity, yield, safety	refinery, operators	valves, pumps, heaters, displays	temperature, pressure, chemical sensors
Interactive English tutor	student's score on test	set of students, testing accuracy	display exercises, suggestions, corrections	keyboard entry

Agents

- Which are agents?
  - (A) James Bond.
  - (B) Your dog.
  - (C) Vacuum cleaner.
  - (D) Thermometer.

SOME RIGHTS RESERVED

Agents

- Which are agents?
  - (A) James Bond.
  - (B) Your dog.
  - (C) Vacuum cleaner.
  - (D) Thermometer.
- **Answer:**
  - (A/B) : Definite yes. (James Bond & your dog)
  - (C) : Yes, if it's an autonomous vacuum cleaner. Else, no.
  - (D) : No, because it cannot do anything. (Changing the displayed temperature value could be considered an "action", but that is not the intended usage of the term)

SOME RIGHTS RESERVED

5.4 Classifying Environments

Environment types

- **Observation 4.1.** *Agent design is largely determined by the type of environment it is intended for.*
- **Problem:** There is a vast number of possible kinds of environments in AI.
- **Solution:** Classify along a few "dimensions". (independent characteristics)
- **Definition 4.2.** For an agent *a* we classify the environment *e* of *a* by its type, which is one of the following. We call *e*
  1. fully observable, iff the *a*'s sensors give it access to the complete state of the environment at any point in time, else partially observable.
  2. deterministic, iff the next state of the environment is completely determined by the current state and *a*'s action, else stochastic.
  3. episodic, iff *a*'s experience is divided into atomic episodes, where it perceives and then performs a single action. Crucially, the next episode does not depend on previous ones. Non-episodic environments are called sequential.
  4. dynamic, iff the environment can change without an action performed by *a*, else static. If the environment does not change but *a*'s performance measure does, we call *e* semidynamic.
  5. discrete, iff the sets of *e*'s state and *a*'s actions are countable, else continuous.
  6. single-agent, iff only *a* acts on *e*; else multi-agent(when must we count parts of *e* as agents?)

SOME RIGHTS RESERVED

Environment Types (Examples)

- **Example 4.3.** Some environments classified:

	Solitaire	Backgammon	Internet shopping	Taxi
fully observable	No	Yes	No	No
deterministic	Yes	No	Partly	No
episodic	No	Yes	No	No
static	Yes	Semi	Semi	No
discrete	Yes	Yes	Yes	No
single-agent	Yes	No	Yes (except auctions)	No

SOME RIGHTS RESERVED

Environment Types (Examples)

- **Example 4.6.** Some environments classified:

	Solitaire	Backgammon	Internet shopping	Taxi
fully observable	No	Yes	No	No
deterministic	Yes	No	Partly	No
episodic	No	Yes	No	No
static	Yes	Semi	Semi	No
discrete	Yes	Yes	Yes	No
single-agent	Yes	No	Yes (except auctions)	No

- **Note:** Take the example above with a grain of salt. There are often multiple interpretations that yield different classifications and different agents. (agent designer's choice)
- **Example 4.7.** Seen as a multi-agent game, chess is deterministic, as a single-agent game, it is stochastic.

SOME RIGHTS RESERVED

Environment Types (Examples)

- **Example 4.9.** Some environments classified:

	Solitaire	Backgammon	Internet shopping	Taxi
fully observable	No	Yes	No	No
deterministic	Yes	No	Partly	No
episodic	No	Yes	No	No
static	Yes	Semi	Semi	No
discrete	Yes	Yes	Yes	No
single-agent	Yes	No	Yes (except auctions)	No

- **Note:** Take the example above with a grain of salt. There are often multiple interpretations that yield different classifications and different agents. (agent designer's choice)
- **Example 4.10.** Seen as a multi-agent game, chess is deterministic, as a single-agent game, it is stochastic.
- **Observation 4.11.** *The real world is (of course) a partially observable, stochastic, sequential, dynamic, continuous, and multi-agent environment.(worst case for AI)*
- **Preview:** We will concentrate on the "easy" environment types (fully observable, deterministic, episodic, static, and single-agent) in AI-1 and extend them to "realworld"-compatible ones in AI-2.

SOME RIGHTS RESERVED

5.5 Types of Agents

SOME RIGHTS RESERVED

Agent Types

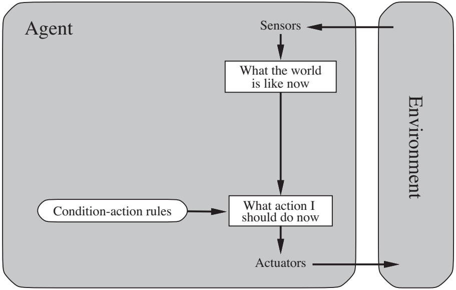
- **Observation:** So far we have described (and analyzed) agents only by their behavior (cf. agent function $f\colon\mathcal{P}^*\to\mathcal{A}$).
- **Problem:** This does not help us to build agents. (the goal of AI)
- To build an agent, we need to fix an agent architecture and come up with an agent program that runs on it.
- **Preview:** Four basic types of agent architectures in order of increasing generality:
  1. reflex agents
  2. model-based agents
  3. goal-based agents
  4. utility-based agents

  All these can be turned into learning agents.

SOME RIGHTS RESERVED

Reflex Agents

- **Definition 5.1.** An agent $\langle\mathcal{P},\mathcal{A},f\rangle$ is called a reflex agent, iff it only takes the last percept into account when choosing an action, i .e. $f(p_1,\ldots,p_k)=f(p_k)$ for all $p_1,\ldots,p_k\in\mathcal{P}$.
- **Agent Schema:**



- **Example 5.2 (Agent Program).**

```
procedure Reflex−Vacuum−Agent [location,status] returns an action
  if status = Dirty then . . .
```

SOME RIGHTS RESERVED

Reflex Agents (continued)

- **General Agent Program:**

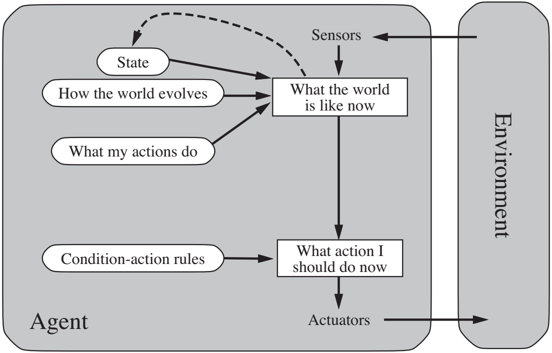
```
function Simple−Reflex−Agent (percept) returns an action
  persistent: rules /∗ a set of condition−action rules∗/

  state := Interpret−Input(percept)
  rule := Rule−Match(state,rules)
  action := Rule−action[rule]
  return action
```

- **Problem:** Reflex agents can only react to the perceived state of the environment, not to changes.
- **Example 5.3.** Automobile tail lights signal braking by brightening. A reflex agent would have to compare subsequent percepts to realize.
- **Problem:** Partially observable environments get reflex agents into trouble.
- **Example 5.4.** Vacuum cleaner robot with defective location sensor ⇝ infinite loops.

SOME RIGHTS RESERVED

Model-based Reflex Agents: Idea

- **Idea:** Keep track of the state of the world we cannot see in an internal model.
- **Agent Schema:**



SOME RIGHTS RESERVED

Model-based Reflex Agents: Definition

- **Definition 5.5.** A model-based agent $\langle\mathcal{P},\mathcal{A},\mathcal{S},\mathcal{T},s_0,S,a\rangle$ is an agent $\langle\mathcal{P},\mathcal{A},f\rangle$ whose actions depend on
  1. a world model: a set $\mathcal{S}$ of possible states, and a start state $s_0\in\mathcal{S}$.
  2. a transition model $\mathcal{T}$, that predicts a new state $\mathcal{T}(s,a)$ from a state s and an action a.
  3. a sensor model S that given a state s and a percept p determine a new state $S(s,p)$.
  4. an action function $a\colon\mathcal{S}\to\mathcal{A}$ that given a state selects the next action.

  If the world model of a model-based agent A is in state s and A has last taken action a, and now perceives p, then A will transition to state $s'=S(p,\mathcal{T}(s,a))$ and take action $a'=a(s')$.

  So, given a sequence $p_1,\ldots,p_n$ of percepts, we recursively define states $s_n=S(\mathcal{T}(s_{n-1},a(s_{n-1})),p_n)$ with $s_1=S(s_0,p_1)$. Then $f(p_1,\ldots,p_n)=a(s_n)$.
- **Note:** As different percept sequences lead to different states, so the agent function $f()\colon\mathcal{P}^*\to\mathcal{A}$ no longer depends only on the last percept.
- **Example 5.6 (Tail Lights Again).** Model-based agents can do the ??? if the states include a concept of tail light brightness.

SOME RIGHTS RESERVED

Model-Based Agents (continued)

- **Observation 5.7.** *The agent program for a model-based agent is of the following form:*

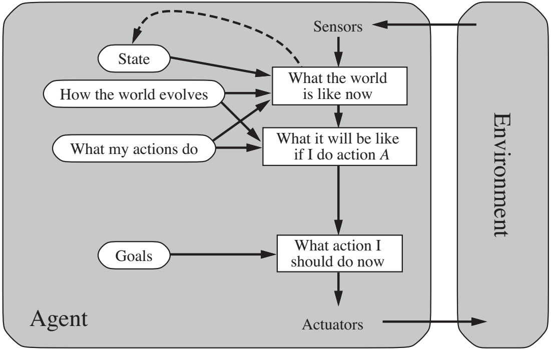
```
function Model−Based−Agent (percept) returns an action
  var state /∗ a description of the current state of the world ∗/
  persistent rules /∗ a set of condition−action rules ∗/
  var action /∗ the most recent action, initially none ∗/

  state := Update−State(state,action,percept)
  rule := Rule−Match(state,rules)
  action := Rule−action(rule)
  return action
```

- **Problem:** Having a world model does not always determine what to do (rationally).
- **Example 5.8.** Coming to an intersection, where the agent has to decide between going left and right.

SOME RIGHTS RESERVED

Goal-based Agents

- **Problem:** A world model does not always determine what to do (rationally).
- **Observation:** Having a goal in mind does! (determines future actions)
- **Agent Schema:**


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Goal-based agents (continued)

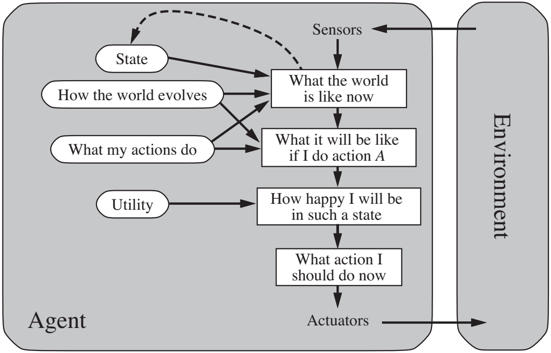
- **Definition 5.9.** A goal-based agent is a model-based agent with transition model T that deliberates actions based on goals and a world model: It employs
  - a set $\mathcal{G}$ of goals and a action function f that given a (new) state s' selects an action a to best reach $\mathcal{G}$.

  The agent function is then $s \mapsto f(T(s), \mathcal{G})$.
- **Observation:** A goal-based agent is more flexible in the knowledge it can utilize.
- **Example 5.10.** A goal-based agent can easily be changed to go to a new destination, a model-based agent's rules make it go to exactly one destination.

SOME RIGHTS RESERVED

Utility-based Agents

- **Definition 5.11.** A utility-based agent uses a world model along with a utility function that models its preferences among the states of that world. It chooses the action that leads to the best expected utility.
- **Agent Schema:**


Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Actuators
Agent
Environment

SOME RIGHTS RESERVED

Utility-based vs. Goal-based Agents

- **Question:** What is the difference between goal-based and utility-based agents?
- **Utility-based Agents are a Generalization:** We can always force goal-directedness by a utility function that only rewards goal states.
- **Goal-based Agents can do less:** A utility function allows rational decisions where mere goals are inadequate:
  - conflicting goals (utility gives tradeoff to make rational decisions)
  - goals obtainable by uncertain actions (utility × likelihood helps)

SOME RIGHTS RESERVED

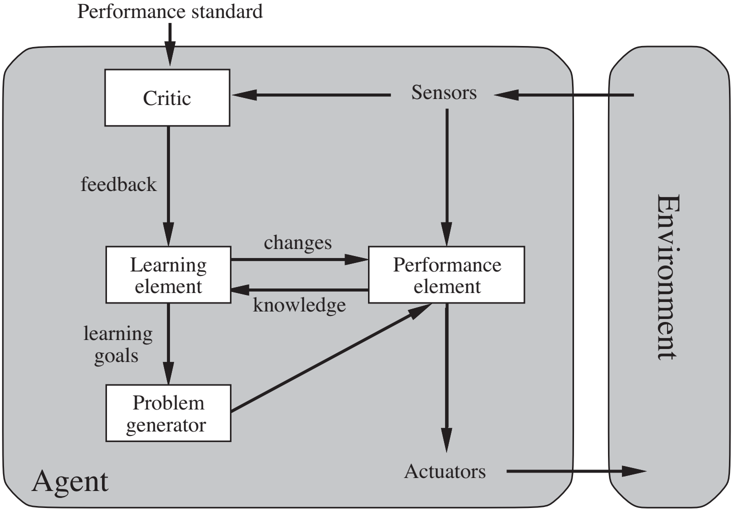
Learning Agents

- **Definition 5.12.** A learning agent is an agent that augments the performance element – which determines actions from percept sequences with
  - a learning element which makes improvements to the agent's components,
  - a critic which gives feedback to the learning element based on an external performance standard,
  - a problem generator which suggests actions that lead to new and informative experiences.
- The performance element is what we took for the whole agent above.

SOME RIGHTS RESERVED

Learning Agents

- **Agent Schema:**



Learning Agents: Example

- **Example 5.13 (Learning Taxi Agent).** It has the components
  - Performance element: the knowledge and procedures for selecting driving actions. (this controls the actual driving)
  - critic: observes the world and informs the learning element (e.g. when passengers complain brutal braking)
  - Learning element modifies the braking rules in the performance element (e.g. earlier, softer)
  - Problem generator might experiment with braking on different road surfaces
- The learning element can make changes to any "knowledge components" of the diagram, e.g. in the
  - model from the percept sequence (how the world evolves)
  - success likelihoods by observing action outcomes (what my actions do)
- **Observation:** here, the passenger complaints serve as part of the "external performance standard" since they correlate to the overall outcome – e.g. in form of tips or blacklists.

SOME RIGHTS RESERVED

Domain-Specific vs. General Agents

-

Domain-Specific Agent	vs.	**General Agent**
 Duell Kasparow gegen Deep Blue (1997): *Demütigende Niederlage*	vs.	
Solver specific to a particular problem ("domain").	vs.	Solver based on *description* in a general problem-description language (e.g., the rules of any board game).
More efficient.	vs.	Much less design/maintenance work.

- What kind of agent are you?

SOME RIGHTS RESERVED

5.6 Representing the Environment in Agents

SOME RIGHTS RESERVED

Representing the Environment in Agents

- We have seen various components of agents that answer questions like
  - "*What is the world like now?*"
  - "*What action should I do now?*"
  - "*What do my actions do?*"
- **Next natural question:** How do these work? (see the rest of the course)

SOME RIGHTS RESERVED

Representing the Environment in Agents

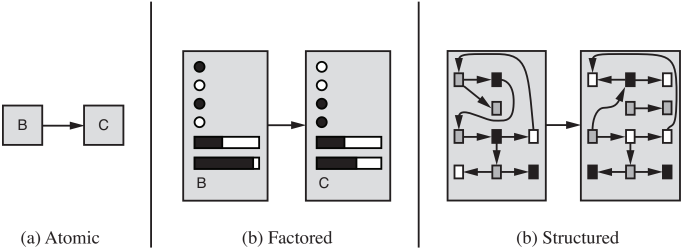
- We have seen various components of agents that answer questions like
  - "*What is the world like now?*"
  - "*What action should I do now?*"
  - "*What do my actions do?*"
- **Next natural question:** How do these work? (see the rest of the course)
- **Important Distinction:** How the agent implements the world model.
- **Definition 6.2.** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attributes and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.

Representing the Environment in Agents

- We have seen various components of agents that answer questions like
  - "*What is the world like now?*"
  - "*What action should I do now?*"
  - "*What do my actions do?*"
- **Next natural question:** How do these work? (see the rest of the course)
- **Important Distinction:** How the agent implements the world model.
- **Definition 6.3.** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attributes and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.
- **Intuition:** From atomic to structured, the representations agent designer more flexibility and the algorithms more components to process.
- **Also** The additional internal structure will make the algorithms more complex.

Atomic/Factored/Structured State Representations

- **Schematically:** We can visualize the three kinds by



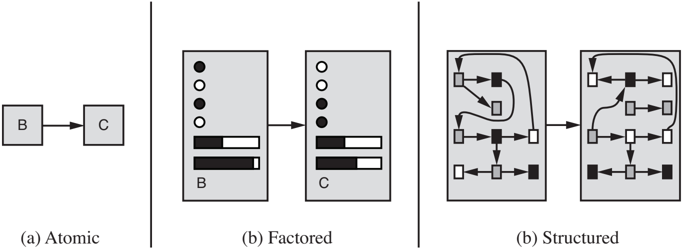
(a) Atomic (b) Factored (b) Structured

- **Example 6.4.** Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities.
  - In an atomic representation the state is represented by the name of a city.

SOME RIGHTS RESERVED

Atomic/Factored/Structured State Representations

- **Schematically:** We can visualize the three kinds by



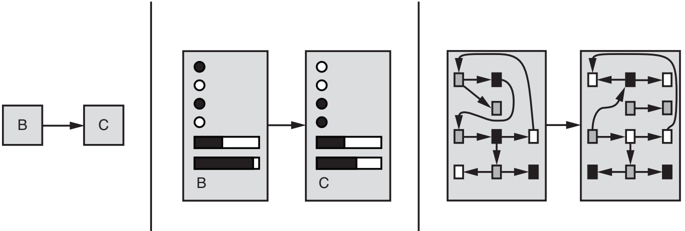
(a) Atomic (b) Factored (b) Structured

- **Example 6.5.** Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities.
  - In an atomic representation the state is represented by the name of a city.
  - In a factored representation we may have attributes "gps-location", "gas",... (allows information sharing between states and uncertainty)
  - But how to represent a situation, where a large truck blocking the road, since it is trying to back into a driveway, but a loose cow is blocking its path. (attribute "TruckAheadBackingIntoDairyFarmDrivewayBlockedByLooseCow" is unlikely)

SOME RIGHTS RESERVED

Atomic/Factored/Structured State Representations

- **Schematically:** We can visualize the three kinds by



(a) Atomic (b) Factored (b) Structured

- **Example 6.6.** Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities.
  - In an atomic representation the state is represented by the name of a city.
  - In a factored representation we may have attributes "gps-location", "gas",... (allows information sharing between states and uncertainty)
  - But how to represent a situation, where a large truck blocking the road, since it is trying to back into a driveway, but a loose cow is blocking its path. (attribute "TruckAheadBackingIntoDairyFarmDrivewayBlockedByLooseCow" is unlikely)
  - In a structured representation, we can have objects for trucks, cows, etc. and their relationships. (at "run-time")

SOME RIGHTS RESERVED

5.7 Rational Agents: Summary

Summary

- Agents interact with environments through actuators and sensors.
- The agent function describes what the agent does in all circumstances.
- The performance measure evaluates the environment sequence.
- A perfectly rational agent maximizes expected performance.
- Agent programs implement (some) agent functions.
- PEAS descriptions define task environments.
- Environments are categorized along several dimensions:
  fully observable? deterministic? episodic? static? discrete? single-agent?
- Several basic agent architectures exist:
  reflex, model-based, goal-based, utility-based

SOME RIGHTS RESERVED

Corollary: We are Agent Designers!

- **State:** We have seen (and will add more details to) different
  - agent architectures,
  - corresponding agent programs and algorithms, and
  - world representation paradigms.
- **Problem:** Which one is the best?

SOME RIGHTS RESERVED

Corollary: We are Agent Designers!

- **State:** We have seen (and will add more details to) different
  - agent architectures,
  - corresponding agent programs and algorithms, and
  - world representation paradigms.
- **Problem:** Which one is the best?
- **Answer:** That really depends on the environment type they have to survive/thrive in! The agent designer – i.e. you – has to choose!
  - The course gives you the necessary competencies.
  - There is often more than one reasonable choice.
  - Often we have to build agents and let them compete to see what really works.

- **Consequence:** The rational agents paradigm used in this course challenges you to become a good agent designer.

SOME RIGHTS RESERVED

Part 2
General Problem Solving

SOME RIGHTS RESERVED

Chapter 6
Problem Solving and Search

6.1 Problem Solving

SOME RIGHTS RESERVED

Problem Solving: Introduction

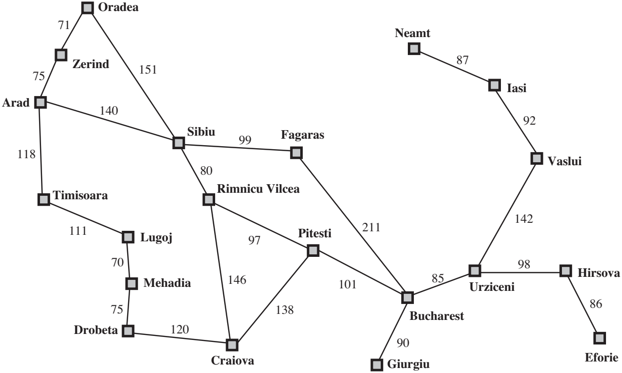
- **Recap:** Agents perceive the environment and compute an action.
- **In other words:** Agents continually solve "the problem of what to do next".
- **AI Goal:** Find algorithms that help solving problems in general.
- **Idea:** If we can describe/represent problems in a standardized way, we may have a chance to find general algorithms.
- **Concretely:** We will use the following two concepts to describe problems
  - States: A set of possible situations in our problem domain ($\widehat{=}$ environments)
  - Actions: that get us from one state to another. ($\widehat{=}$ agents)

  A sequence of actions is a solution, if it brings us from an initial state to a goal state. Problem solving computes solutions from problem formulations.
- **Definition 1.1.** In offline problem solving an agent computing an action sequence based complete knowledge of the environment.
- *Remark 1.2.* Offline problem solving only works in fully observable, deterministic, static, and episodic environments.
- **Definition 1.3.** In online problem solving an agent computes one action at a time based on incoming perceptions.
- **This Semester:** We largely restrict ourselves to offline problem solving.(easier)

SOME RIGHTS RESERVED

Example: Traveling in Romania

- **Scenario:** An agent is on holiday in Romania; currently in Arad; flight home leaves tomorrow from Bucharest; how to get there? We have a map:



- **Formulate the Problem:**
  - States: various cities.
  - Actions: drive between cities.
- **Solution:** Appropriate sequence of cities, e.g.: Arad, Sibiu, Fagaras, Bucharest

SOME RIGHTS RESERVED

Problem Formulation

- **Definition 1.4.** A problem formulation models a situation using states and actions at an appropriate level of abstraction. (do not model things like "put on my left sock", etc.)
    - it describes the initial state (we are in Arad)
    - it also limits the objectives by specifying goal states. (excludes, e.g. to stay another couple of weeks.)

  A solution is a sequence of actions that leads from the initial state to a goal state.

  Problem solving computes solutions from problem formulations.
- Finding the right level of abstraction and the required (not more!) information is often the key to success.

SOME RIGHTS RESERVED

The Math of Problem Formulation: Search Problems

- **Definition 1.5.** A search problem $\Pi := \langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$ consists of a set $\mathcal{S}$ of states, a set $\mathcal{A}$ of actions, and a transition model $\mathcal{T}\colon \mathcal{A}\times\mathcal{S}\to\mathcal{P}(\mathcal{S})$ that assigns to any action $a\in\mathcal{A}$ and state $s\in\mathcal{S}$ a set of successor states.
  Certain states in $\mathcal{S}$ are designated as goal states (also called terminal states) ($\mathcal{G}\subseteq\mathcal{S}$ with $\mathcal{G}\neq\emptyset$) and initial states $\mathcal{I}\subseteq\mathcal{S}$.
- **Definition 1.6.** We say that an action $a\in\mathcal{A}$ is applicable in state $s\in\mathcal{S}$, iff $\mathcal{T}(a,s)\neq\emptyset$ and that any $s'\in\mathcal{T}(a,s)$ is a result of applying action a to state s.
  We call $\mathcal{T}_a\colon \mathcal{S}\to\mathcal{P}(\mathcal{S})$ with $\mathcal{T}_a(s):=\mathcal{T}(a,s)$ the result relation for a and $\mathcal{T}_\mathcal{A}:=\bigcup_{a\in\mathcal{A}}\mathcal{T}_a$ the result relation of Π.
- **Definition 1.7.** The graph $\langle\mathcal{S},\mathcal{T}_\mathcal{A}\rangle$ is called the state space induced by Π.
- **Definition 1.8.** A solution for Π consists of a sequence $a_1,\ldots,a_n$ of actions such that for all $1<i\leq n$
  - a_i is applicable to state s_{i-1}, where $s_0\in\mathcal{I}$ and
  - $s_i\in\mathcal{T}_{a_i}(s_{i-1})$, and $s_n\in\mathcal{G}$.
- **Idea:** A solution bring us from $\mathcal{I}$ to a goal state via applicable actions.
- **Definition 1.9.** Often we add a cost function $c\colon \mathcal{A}\to\mathbb{R}_0^+$ that associates a step cost $c(a)$ to an action $a\in\mathcal{A}$. The cost of a solution is the sum of the step costs of its actions.

SOME RIGHTS RESERVED

Structure Overview: Search Problem

- The structure overview for search problems:

$$\text{search problem} = \left\langle \begin{array}{lll} \mathcal{S} & \text{Set} & \text{states}, \\ \mathcal{A} & \text{Set} & \text{actions}, \\ \mathcal{T} & \mathcal{A}\times\mathcal{S}\to\mathcal{P}(\mathcal{S}) & \text{transition model}, \\ \mathcal{I} & \mathcal{S} & \text{initial state}, \\ \mathcal{G} & \mathcal{P}(\mathcal{S}) & \text{goal states} \end{array} \right\rangle$$

SOME RIGHTS RESERVED

Search Problems in deterministic, fully observable Environments

- This semester, we will restrict ourselves to search problems, where(extend in AI II)
  - $|\mathcal{T}(a,s)| \leq 1$ for the transition models and (⇜ deterministic environment)
  - $\mathcal{I} = \{s_0\}$ (⇜ fully observable environment)

  Definition 1.11. We call a search problem deterministic, iff the underlying transition system is.
-
- **Definition 1.12.** In a search problem $\mathcal{T}_a$ induces partial function $\mathcal{S}_a : \mathcal{S} \rightharpoonup \mathcal{S}$ whose natural domain is the set of states where a is applicable: $\mathcal{S}_a(s) := s'$ if $\mathcal{T}_a = \{s'\}$ and undefined at s otherwise. We call $\mathcal{S}_a$ the successor function for a and $\mathcal{S}_a(s)$ the successor state of s.
- **Definition 1.13.** The predicate that tests for goal states is called a goal test.

SOME RIGHTS RESERVED

6.2 Problem Types

SOME RIGHTS RESERVED

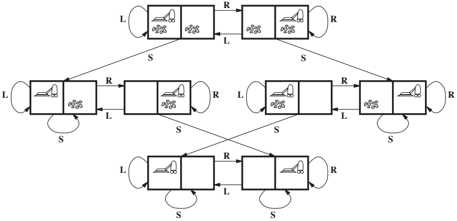
Problem types

- **Definition 2.1.** A search problem is called a single state problem, iff it is
  - fully observable (at least the initial state)
  - deterministic (unique successor states)
  - static (states do not change other than by our own actions)
  - discrete (a countable number of states)
- **Definition 2.2.** A search problem is called a multi state problem
  - states partially observable (e.g. multiple initial states)
  - deterministic, static, discrete
- **Definition 2.3.** A search problem is called a contingency problem, iff
  - the environment is non deterministic (solution can branch, depending on contingencies)
  - the state space is unknown(like a baby, agent has to learn about states and actions)

SOME RIGHTS RESERVED

Example: vacuum-cleaner world

- **Single-state Problem:**
  - Start in 5
  - **Solution**: $[\mathit{right}, \mathit{suck}]$



- **Multiple-state Problem:**
  - Start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
  - **Solution**: $[\mathit{right}, \mathit{suck}, \mathit{left}, \mathit{suck}]$

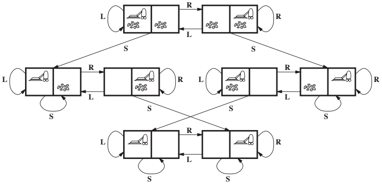
$$\begin{array}{ll} \mathit{right} & \rightarrow \{2, 4, 6, 8\} \\ \mathit{suck} & \rightarrow \{4, 8\} \\ \mathit{left} & \rightarrow \{3, 7\} \\ \mathit{suck} & \rightarrow \{7\} \end{array}$$

SOME RIGHTS RESERVED

Example: Vacuum-Cleaner World (continued)

- **Contingency Problem:**

- Murphy's Law: *suck* can dirty a clean carpet
- Local sensing: *dirty*/*notdirty* at location only
- Start in: $\{1,3\}$
- **Solution**: $[suck, right, suck]$
  $suck \rightarrow \{5,7\}$
  $right \rightarrow \{6,8\}$
  $suck \rightarrow \{6,8\}$



- **better:** $[suck, right,$ if *dirt* then *suck*$]$ (decide whether in 6 or 8 using local sensing)

Single-state problem formulation

- Defined by the following four items
  1. Initial state: (e.g. $Arad$)
  2. Successor function $S_a(s)$: (e.g. $S_{goZer} = \{(Arad, Zerind), (goSib, Sibiu), \ldots\}$)
  3. Goal test: (e.g. $x = Bucharest$ (explicit test))
     $noDirt(x)$ (implicit test)
  4. Path cost (optional): (e.g. sum of distances, number of operators executed, etc.)
- Solution: A sequence of actions leading from the initial state to a goal state.

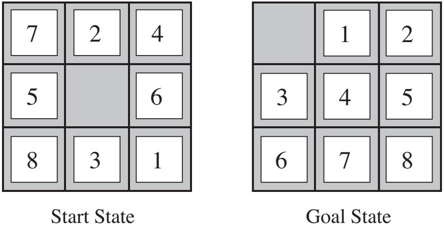
SOME RIGHTS RESERVED

Selecting a state space

- **Abstraction:** Real world is absurdly complex!
  State space must be abstracted for problem solving.
- **(Abstract) state:** Set of real states.
- **(Abstract) operator:** Complex combination of real actions.
- **Example:** *Arad* → *Zerind* represents complex set of possible routes.
- **(Abstract) solution:** Set of real paths that are solutions in the real world.

SOME RIGHTS RESERVED

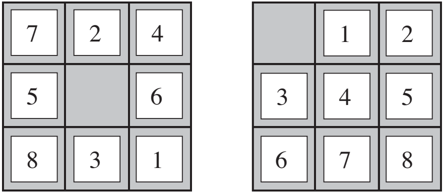
Example: The 8-puzzle



States? Actions?...

SOME RIGHTS RESERVED

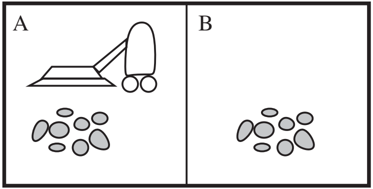
Example: The 8-puzzle



Start State Goal State

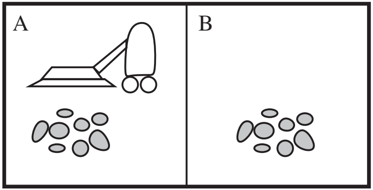
States	integer locations of tiles
Actions	*left*, *right*, *up*, *down*
Goal test	= goal state?
Path cost	1 per move

Example: Vacuum-cleaner



States? Actions?. . .

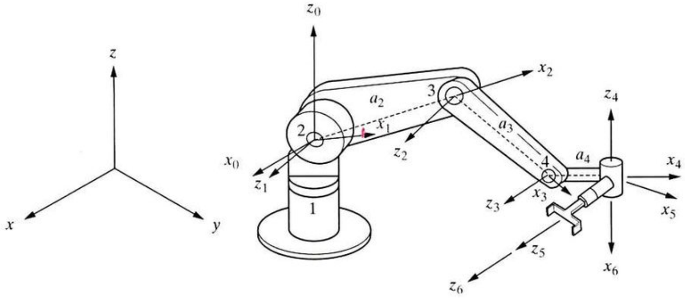
Example: Vacuum-cleaner



States	integer dirt and robot locations
Actions	*left*, *right*, *suck*, *noOp*
Goal test	*notdirty*?
Path cost	1 per operation (0 for *noOp*)

SOME RIGHTS RESERVED

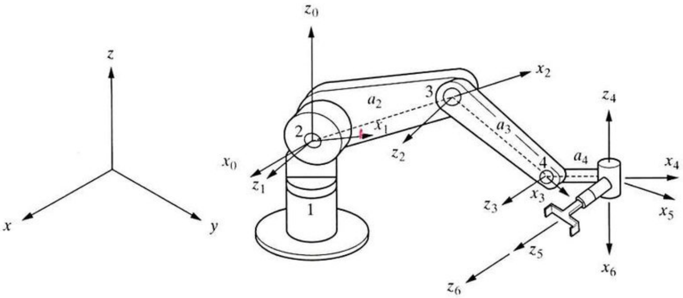
Example: Robotic assembly



States? Actions?...

SOME RIGHTS RESERVED

Example: Robotic assembly



States	real-valued coordinates of robot joint angles and parts of the object to be assembled
Actions	continuous motions of robot joints
Goal test	assembly complete?
Path cost	time to execute

SOME RIGHTS RESERVED

General Problems

- **Question:** Which are “Problems”?
  - (A) You didn’t understand any of the lecture.
  - (B) Your bus today will probably be late.
  - (C) Your vacuum cleaner wants to clean your apartment.
  - (D) You want to win a chess game.

SOME RIGHTS RESERVED

General Problems

- **Question:** Which are “Problems”?
  (A) You didn’t understand any of the lecture.
  (B) Your bus today will probably be late.
  (C) Your vacuum cleaner wants to clean your apartment.
  (D) You want to win a chess game.
- **Answer:**
  (A/B) These are problems in the natural language use of the word, but not “problems” in the sense defined here.

SOME RIGHTS RESERVED

General Problems

- **Question:** Which are “Problems”?
  - (A) You didn’t understand any of the lecture.
  - (B) Your bus today will probably be late.
  - (C) Your vacuum cleaner wants to clean your apartment.
  - (D) You want to win a chess game.
- **Answer:**
  - (A/B) These are problems in the natural language use of the word, but not “problems” in the sense defined here.
  - (C) Yes, presuming that this is a robot, an autonomous vacuum cleaner, and that the robot has perfect knowledge about your apartment (else, it’s not a classical search problem).

SOME RIGHTS RESERVED

General Problems

- **Question:** Which are "Problems"?
  - (A) You didn't understand any of the lecture.
  - (B) Your bus today will probably be late.
  - (C) Your vacuum cleaner wants to clean your apartment.
  - (D) You want to win a chess game.
- **Answer:**
  - (A/B) These are problems in the natural language use of the word, but not "problems" in the sense defined here.
  - (C) Yes, presuming that this is a robot, an autonomous vacuum cleaner, and that the robot has perfect knowledge about your apartment (else, it's not a classical search problem).
  - (D) That's a search problem, but not a classical search problem (because it's multi-agent). We'll tackle this kind of problem in

SOME RIGHTS RESERVED

6.3 Search

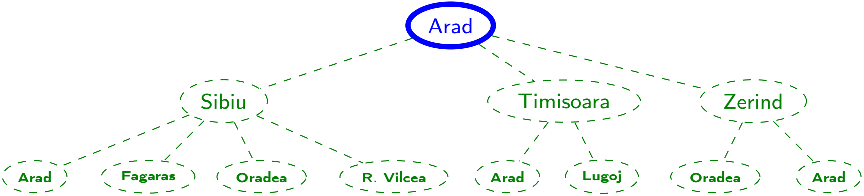
Tree Search Algorithms

- **Note:** The state space of a search problem $\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$ is a graph $\langle\mathcal{S},\mathcal{T}_{\mathcal{A}}\rangle$.
- As graphs are difficult to compute with, we often compute a corresponding tree and work on that. (standard trick in graph algorithms)
- **Definition 3.1.** Given a search problem $\mathcal{P} := \langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, the tree search algorithm consists of the simulated exploration of state space $\langle\mathcal{S},\mathcal{T}_{\mathcal{A}}\rangle$ in a search tree formed by successively expanding already explored states. (offline algorithm)

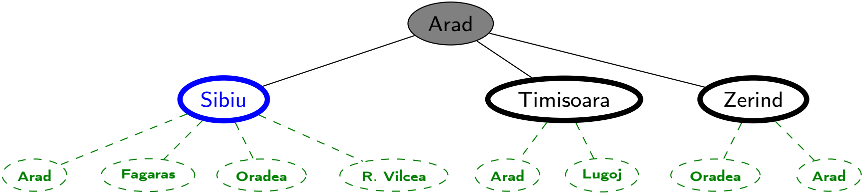
```
procedure Tree−Search (problem, strategy) : <a solution or failure>
  <initialize the search tree using the initial state of problem>
  loop
     if <there are no candidates for expansion> <return failure> end if
    <choose a leaf node for expansion according to strategy>
     if <the node contains a goal state> return <the corresponding solution>
     else <expand the node and add the resulting nodes to the search tree>
     end if
  end loop
end procedure
```

We expand a node n by generating all successors of n and inserting them as children of n in the search tree.

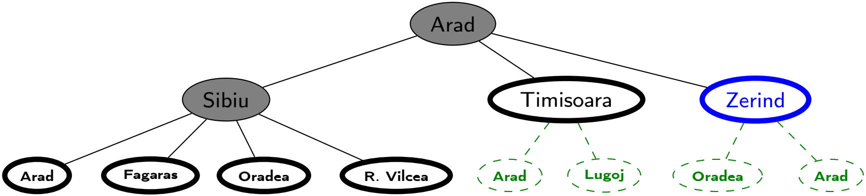
Tree Search: Example



Tree Search: Example

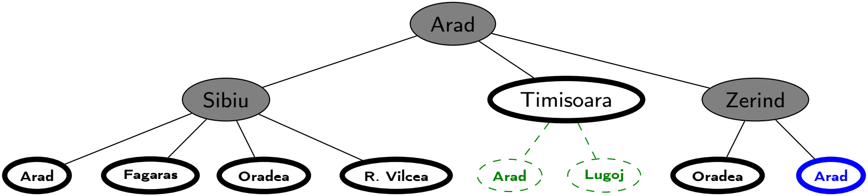


Tree Search: Example



SOME RIGHTS RESERVED

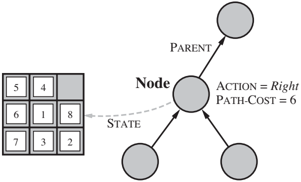
Tree Search: Example



Implementation: States vs. Nodes

- **Recap:** A state is a (representation of) a physical configuration.
- **Definition 3.2 (Implementing a Search Tree).**

A search tree node is a data structure that includes accessors for parent, children, depth, path cost, insertion order, etc.
A goal node (initial node) is a search tree node labeled with a goal state (initial state).



- **Observation:** A set of search tree nodes that can all (recursively) reach a single initial node form a search tree. (they implement it)
- **Observation:** Paths in the search tree correspond to paths in the state space.
- **Definition 3.3.** We define the path cost of a node n in a search tree T to be the sum of the step costs on the path from n to the root of T.
- **Observation:** As a search tree node has access to parents, we can read off the solution from a goal node.

SOME RIGHTS RESERVED

Implementation of Search Algorithms

- **Definition 3.4 (Implemented Tree Search Algorithm).**

```
procedure Tree_Search (problem,strategy)
  fringe := insert(make_node(initial_state(problem)))
    loop
     if empty(fringe) fail end if
      node := first(fringe,strategy)
      if GoalTest(node) return node
      else fringe := insert(expand(node,problem))
      end if
   end loop
end procedure
```

The fringe is the set of search tree nodes not yet expanded in tree search.

- **Idea:** We treat the fringe as an abstract data type with three accessors: the
  - binary function first retrieves an element from the fringe according to a strategy.
  - binary function insert adds a (set of) search tree node into a fringe.
  - unary predicate empty to determine whether a fringe is the empty set.
- The strategy determines the behavior of the fringe (data structure) (see below)

SOME RIGHTS RESERVED

Search strategies

- **Definition 3.5.** A strategy is a function that picks a node from the fringe of a search tree. (equivalently, orders the fringe and picks the first.)
- **Definition 3.6 (Important Properties of Strategies).**

completeness	does it always find a solution if one exists?
time complexity	number of nodes generated/expanded
space complexity	maximum number of nodes in memory
optimality	does it always find a least cost solution?

- **Time and space complexity measured in terms of:**

b	maximum branching factor of the search tree
d	minimal graph depth of a solution in the search tree
m	maximum graph depth of the search tree (may be ∞)

Complexity always means *worst-case* complexity here!

SOME RIGHTS RESERVED

6.4 Uninformed Search Strategies

Uninformed search strategies

- **Definition 4.1.** We speak of an uninformed search algorithm, if it only uses the information available in the problem definition.
- **Next:** Frequently used search algorithms
  - Breadth first search
  - Uniform cost search
  - Depth first search
  - Depth limited search
  - Iterative deepening search

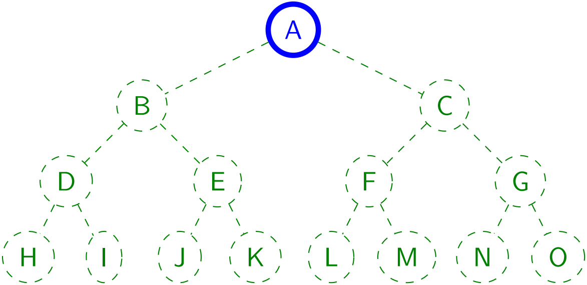
SOME RIGHTS RESERVED

6.4.1 Breadth-First Search Strategies

SOME RIGHTS RESERVED

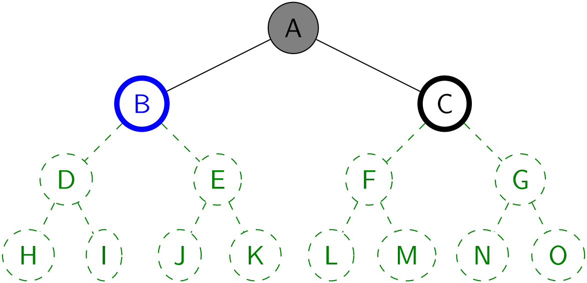
Breadth-First Search

- **Idea:** Expand the shallowest unexpanded node.
- **Definition 4.2.** The breadth first search (BFS) strategy treats the fringe as a FIFO queue, i.e. successors go in at the end of the fringe.
- **Example 4.3 (Synthetic).**



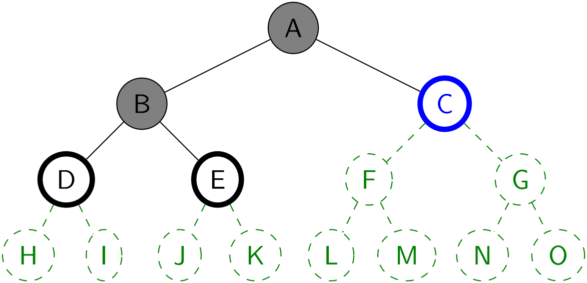
Breadth-First Search

- **Idea:** Expand the shallowest unexpanded node.
- **Definition 4.4.** The breadth first search (BFS) strategy treats the fringe as a FIFO queue, i.e. successors go in at the end of the fringe.
- **Example 4.5 (Synthetic).**



Breadth-First Search

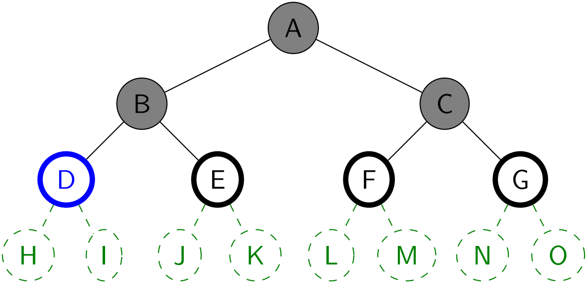
- **Idea:** Expand the shallowest unexpanded node.
- **Definition 4.6.** The breadth first search (BFS) strategy treats the fringe as a FIFO queue, i.e. successors go in at the end of the fringe.
- **Example 4.7 (Synthetic).**



SOME RIGHTS RESERVED

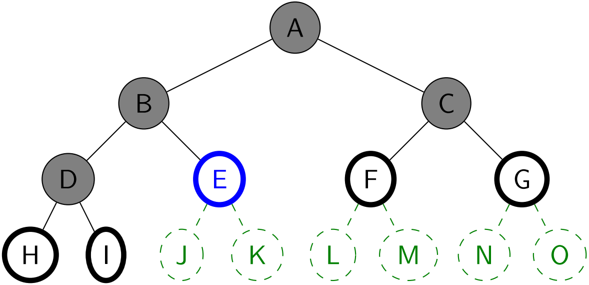
Breadth-First Search

- **Idea:** Expand the shallowest unexpanded node.
- **Definition 4.8.** The breadth first search (BFS) strategy treats the fringe as a FIFO queue, i.e. successors go in at the end of the fringe.
- **Example 4.9 (Synthetic).**



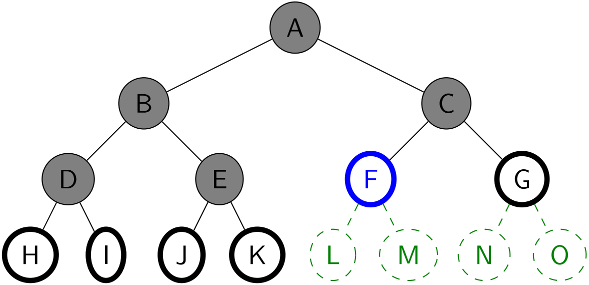
Breadth-First Search

- **Idea:** Expand the shallowest unexpanded node.
- **Definition 4.10.** The breadth first search (BFS) strategy treats the fringe as a FIFO queue, i.e. successors go in at the end of the fringe.
- **Example 4.11 (Synthetic).**



Breadth-First Search

- **Idea:** Expand the shallowest unexpanded node.
- **Definition 4.12.** The breadth first search (BFS) strategy treats the fringe as a FIFO queue, i.e. successors go in at the end of the fringe.
- **Example 4.13 (Synthetic).**



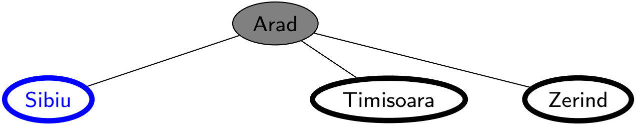
Breadth-First Search: Romania

- **Example 4.14.**

SOME RIGHTS RESERVED

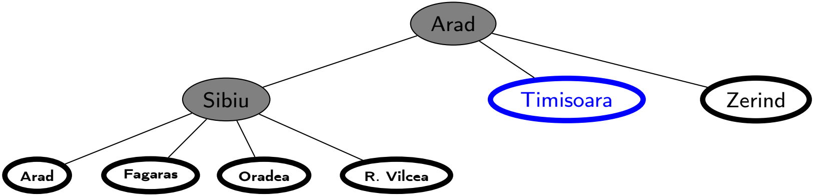
Breadth-First Search: Romania

- **Example 4.15.**



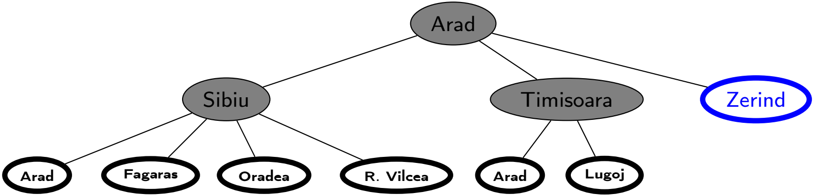
Breadth-First Search: Romania

- **Example 4.16.**



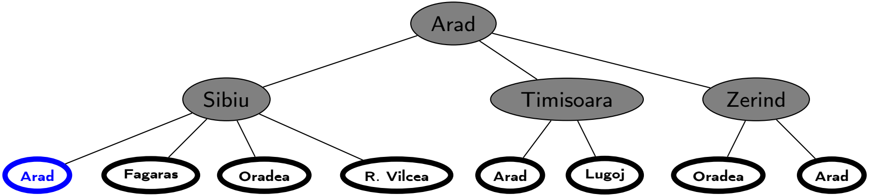
Breadth-First Search: Romania

- **Example 4.17.**



Breadth-First Search: Romania

- **Example 4.18.**

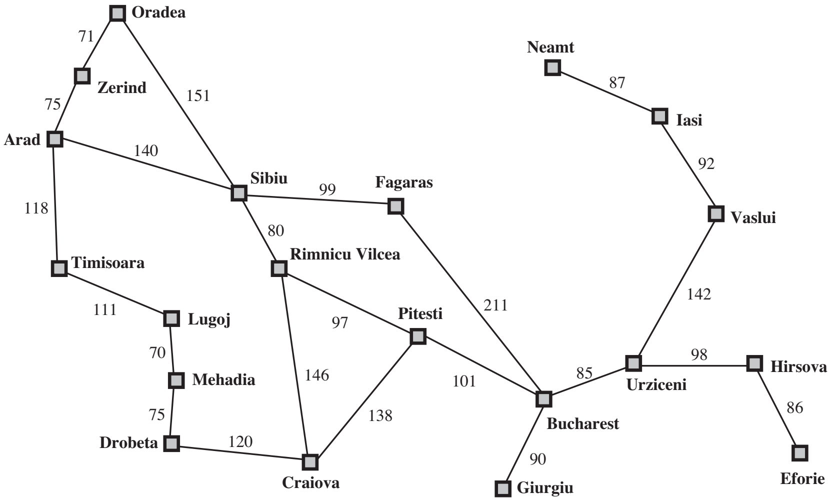


Breadth-first search: Properties

Completeness	Yes (if b is finite)
Time complexity	$1+b+b^2+b^3+\ldots+b^d$, so $\mathcal{O}(b^d)$, i.e. exponential in d
Space complexity	$\mathcal{O}(b^d)$ (fringe may be whole level)
Optimality	Yes (if cost = 1 per step), not optimal in general

- **Disadvantage:** Space is the big problem (can easily generate nodes at 500MB/sec $\widehat{=}$ 1.8TB/h)
- **Optimal?:** No! If cost varies for different steps, there might be better solutions below the level of the first one.
- An alternative is to generate *all* solutions and then pick an optimal one. This works only, if m is finite.

Romania with Step Costs as Distances



SOME RIGHTS RESERVED

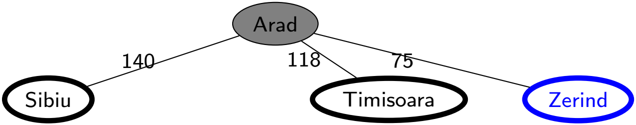
Uniform-cost search

- **Idea:** Expand least cost unexpanded node.
- **Definition 4.19.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- **Note:** Equivalent to breadth first search if all step costs are equal.
- **Synthetic Example:**

SOME RIGHTS RESERVED

Uniform-cost search

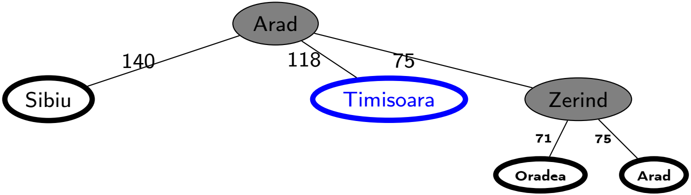
- **Idea:** Expand least cost unexpanded node.
- **Definition 4.20.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- **Note:** Equivalent to breadth first search if all step costs are equal.
- **Synthetic Example:**



SOME RIGHTS RESERVED

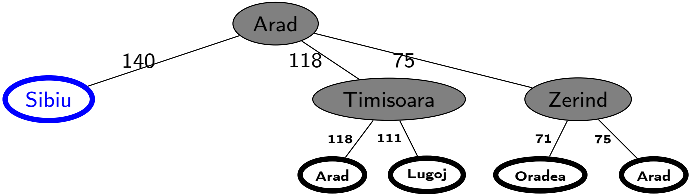
Uniform-cost search

- **Idea:** Expand least cost unexpanded node.
- **Definition 4.21.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- **Note:** Equivalent to breadth first search if all step costs are equal.
- **Synthetic Example:**



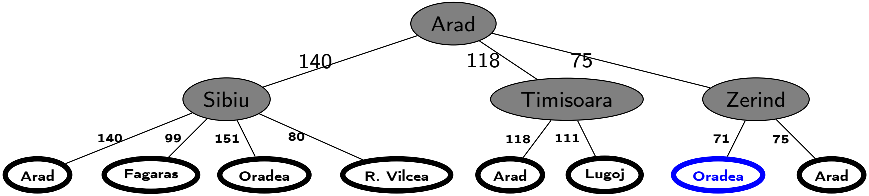
Uniform-cost search

- **Idea:** Expand least cost unexpanded node.
- **Definition 4.22.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- **Note:** Equivalent to breadth first search if all step costs are equal.
- **Synthetic Example:**



Uniform-cost search

- **Idea:** Expand least cost unexpanded node.
- **Definition 4.23.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- **Note:** Equivalent to breadth first search if all step costs are equal.
- **Synthetic Example:**



SOME RIGHTS RESERVED

Uniform-cost search: Properties

Completeness	Yes (if step costs $\geq \epsilon > 0$)
Time complexity	number of nodes with path cost less than that of optimal solution
Space complexity	ditto
Optimality	Yes

SOME RIGHTS RESERVED

6.4.2 Depth-First Search Strategies

SOME RIGHTS RESERVED

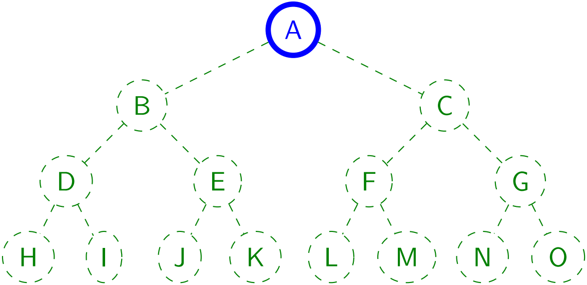
Depth-first Search

- **Idea:** Expand deepest unexpanded node.
- **Definition 4.24.** Depth-first search (DFS) is the strategy where the fringe is organized as a (LIFO) stack i.e. successors go in at front of the fringe.
- **Definition 4.25.** Every node that is pushed to the stack is called a backtrack point. The action of popping a non-goal node from the stack and continuing the search with the new top element of the stack (a backtrack point by construction) is called backtracking, and correspondingly the DFS algorithm backtracking search.
- **Note:** Depth first search can perform infinite cyclic excursions
  Need a finite, non cyclic state space (or repeated state checking)

SOME RIGHTS RESERVED

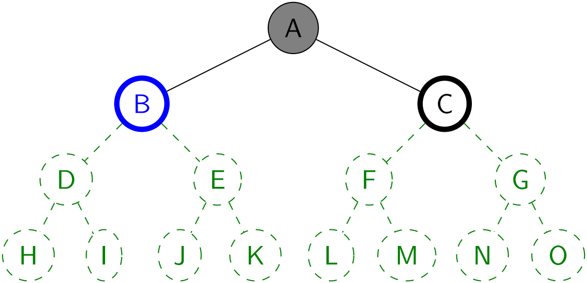
Depth-First Search

- **Example 4.26 (Synthetic).**



Depth-First Search

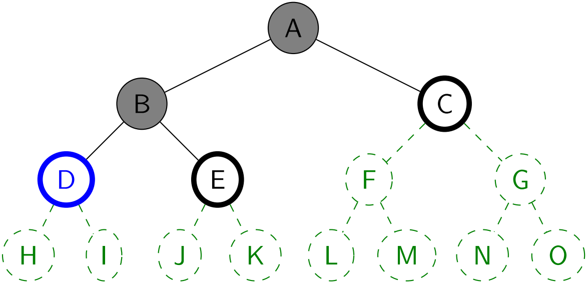
- **Example 4.27 (Synthetic).**



SOME RIGHTS RESERVED

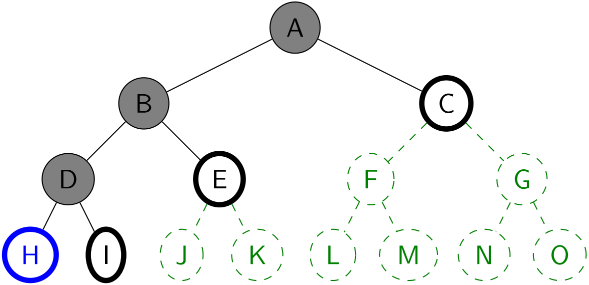
Depth-First Search

- **Example 4.28 (Synthetic).**



Depth-First Search

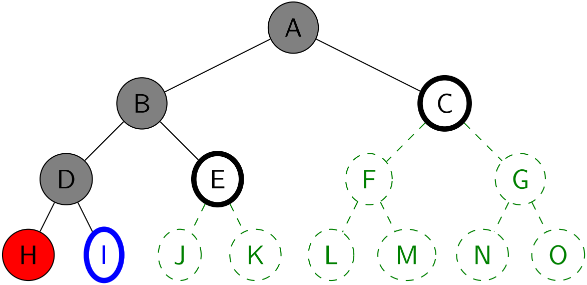
- **Example 4.29 (Synthetic).**



SOME RIGHTS RESERVED

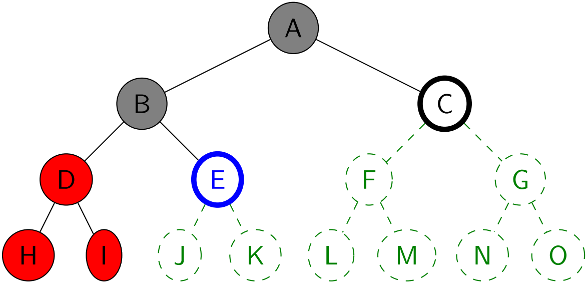
Depth-First Search

- **Example 4.30 (Synthetic).**



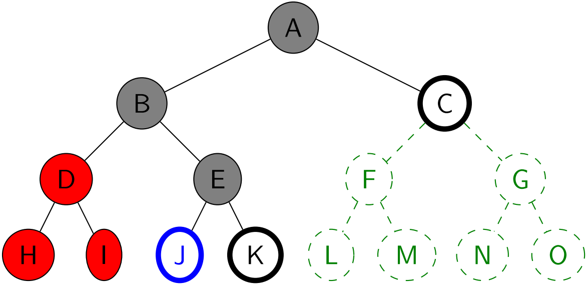
Depth-First Search

- **Example 4.31 (Synthetic).**



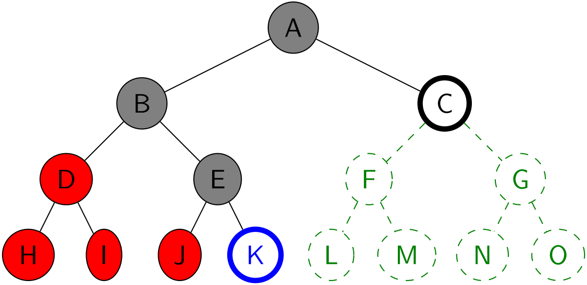
Depth-First Search

- **Example 4.32 (Synthetic).**



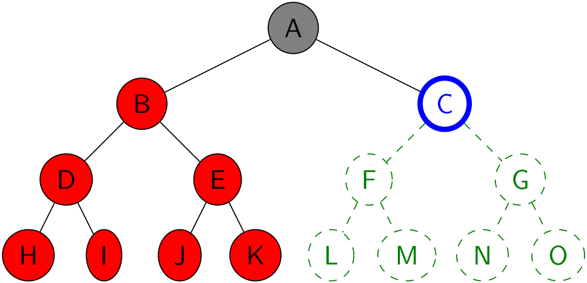
Depth-First Search

- **Example 4.33 (Synthetic).**



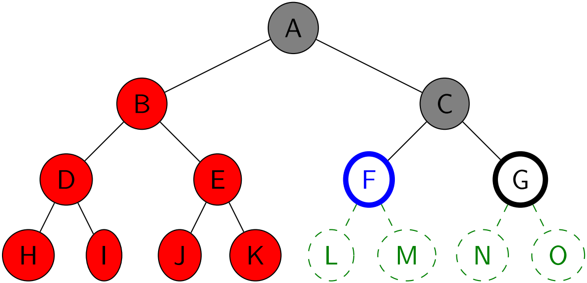
Depth-First Search

- **Example 4.34 (Synthetic).**



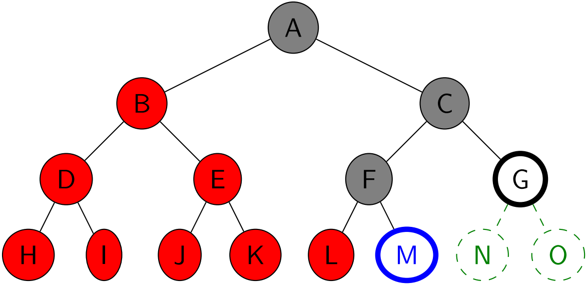
Depth-First Search

- **Example 4.35 (Synthetic).**



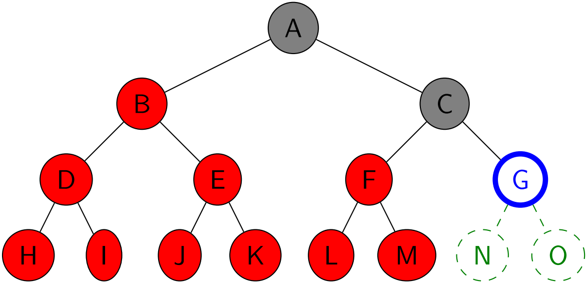
Depth-First Search

- **Example 4.36 (Synthetic).**



Depth-First Search

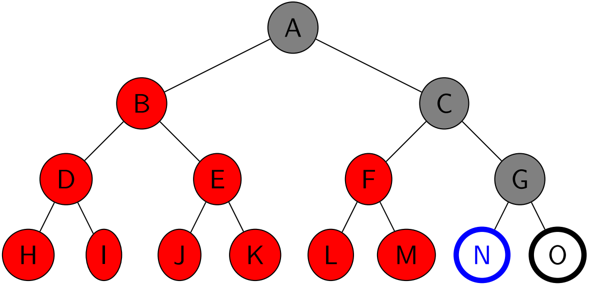
- **Example 4.37 (Synthetic).**



SOME RIGHTS RESERVED

Depth-First Search

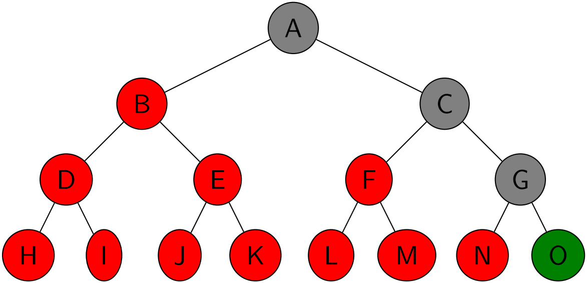
- **Example 4.38 (Synthetic).**



SOME RIGHTS RESERVED

Depth-First Search

- **Example 4.39 (Synthetic).**



SOME RIGHTS RESERVED

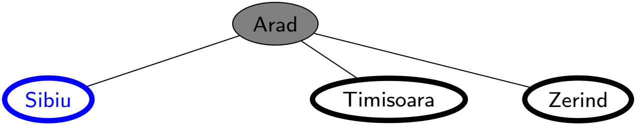
Depth-First Search: Romania

- **Example 4.40 (Romania).**

SOME RIGHTS RESERVED

Depth-First Search: Romania

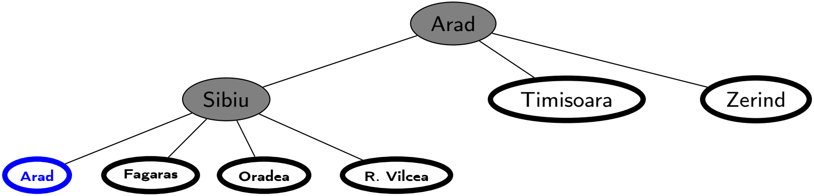
- **Example 4.41 (Romania).**



SOME RIGHTS RESERVED

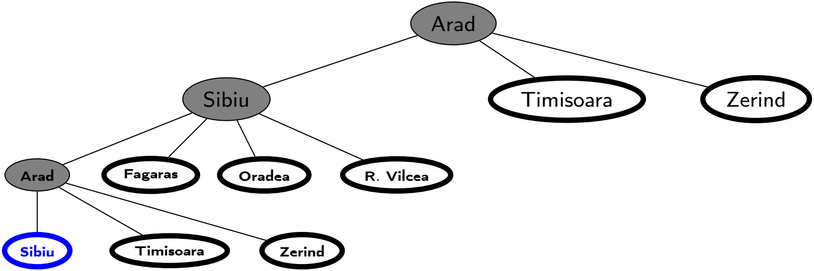
Depth-First Search: Romania

- **Example 4.42 (Romania).**



Depth-First Search: Romania

- **Example 4.43 (Romania).**



Depth-first search: Properties

Completeness	Yes: if search tree finite No: if search tree contains infinite paths or loops
Time complexity	$\mathcal{O}(b^m)$ (we need to explore until max depth m in any case!)
Space complexity	$\mathcal{O}(bm)$ (i.e. linear space) (need at most store m levels and at each level at most b nodes)
Optimality	No (there can be many better solutions in the unexplored part of the search tree)

- **Disadvantage:** Time terrible if m much larger than d.
- **Advantage:** Time may be much less than breadth first search if solutions are dense.

SOME RIGHTS RESERVED

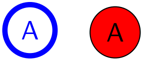
Iterative deepening search

- **Definition 4.44.** Depth limited search is depth first search with a depth limit.
- **Definition 4.45.** Iterative deepening search (IDS) is depth limited search with ever increasing depth limits. We call the difference between successive depth limits the step size.

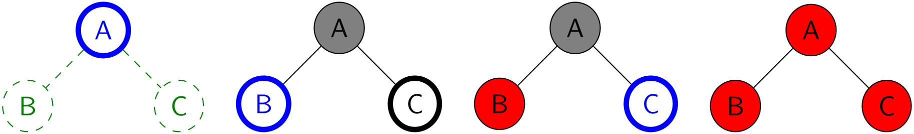
-
  ```
  procedure Tree_Search (problem)
    <initialize the search tree using the initial state of problem>
    for depth = 0 to $\infty$
      result := Depth_Limited_search(problem,depth)
      if depth $\neq$ cutoff return result end if
    end for
  end procedure
  ```

SOME RIGHTS RESERVED

Ilustration: Iterative Deepening Search at various Limit Depths

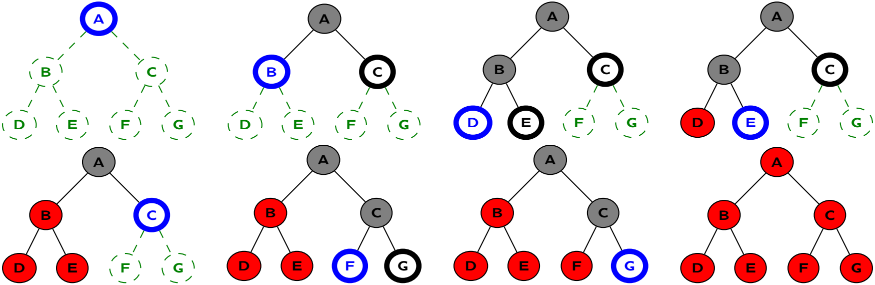


Ilustration: Iterative Deepening Search at various Limit Depths



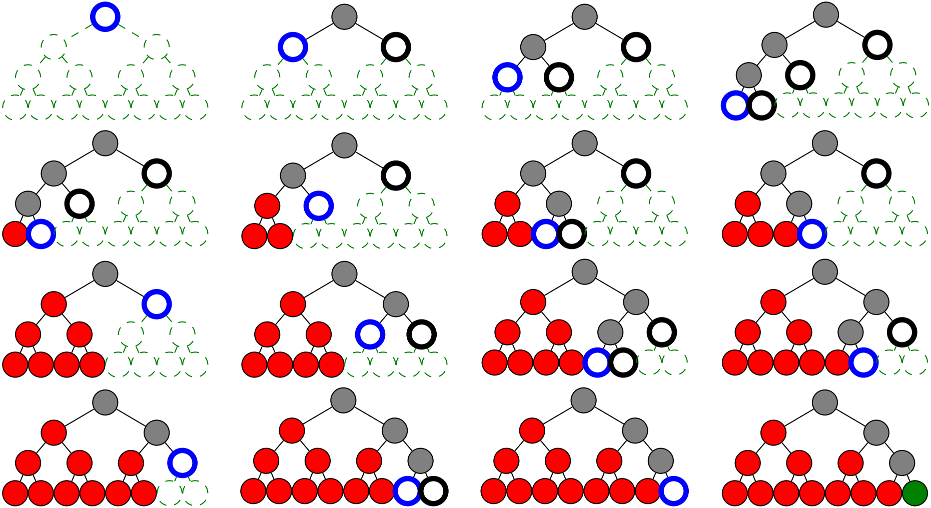
SOME RIGHTS RESERVED

Ilustration: Iterative Deepening Search at various Limit Depths



SOME RIGHTS RESERVED

Ilustration: Iterative Deepening Search at various Limit Depths



SOME RIGHTS RESERVED

Iterative deepening search: Properties

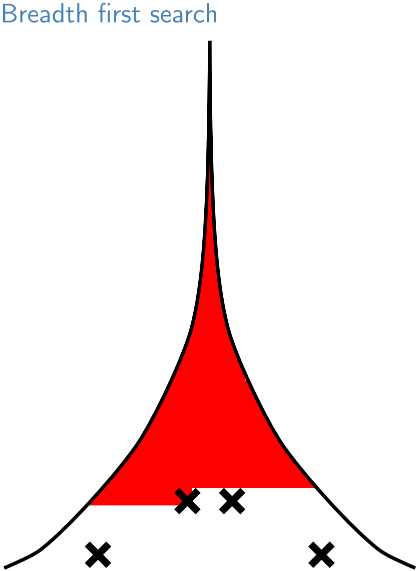
Completeness	Yes
Time complexity	$(d+1)\cdot b^0+d\cdot b^1+(d-1)\cdot b^2+\ldots+b^d\in\mathcal{O}(b^{d+1})$
Space complexity	$\mathcal{O}(b\cdot d)$
Optimality	Yes (if step cost $=1$)

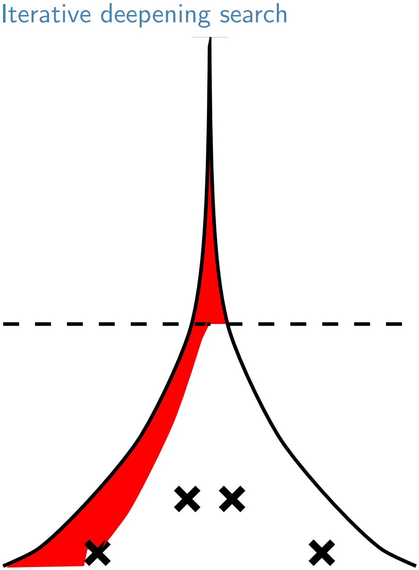
- **Consequence:** IDS used in practice for search spaces of large, infinite, or unknown depth.

SOME RIGHTS RESERVED

Comparison BFS (optimal) and IDS (not)

- **Example 4.46.** IDS may fail to be be optimal at step sizes > 1.





6.4.3 Further Topics

SOME RIGHTS RESERVED

Tree Search vs. Graph Search

- We have only covered tree search algorithms.
- States duplicated in nodes are a huge problem for efficiency.
- **Definition 4.47.** A graph search algorithm is a variant of a tree search algorithm that prunes nodes whose state has already been considered (duplicate pruning), essentially using a DAG data structure.
- **Observation 4.48.** *Tree search is memory intensive it has to store the fringe so keeping a list of "explored states" does not lose much.*
- Graph versions of all the tree search algorithms considered here exist, but are more difficult to understand (and to prove properties about).
- The (time complexity) properties are largely stable under duplicate pruning. (no gain in the worst case)
- **Definition 4.49.** We speak of a search algorithm, when we do not want to distinguish whether it is a tree or graph search algorithm. (difference considered an implementation detail)

SOME RIGHTS RESERVED

Uninformed Search Summary

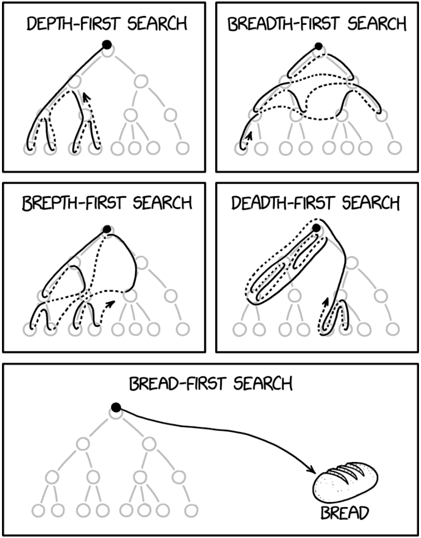
- **Tree/Graph Search Algorithms:** Systematically explore the state tree/graph induced by a search problem in search of a goal state. Search strategies only differ by the treatment of the fringe.
- **Search Strategies and their Properties:** We have discussed

Criterion	Breadth first	Uniform cost	Depth first	Iterative deepening
Completeness	Yes[1]	Yes[2]	No	Yes
Time complexity	b^d	$\approx b^d$	b^m	b^{d+1}
Space complexity	b^d	$\approx b^d$	bm	bd
Optimality	Yes*	Yes	No	Yes*
Conditions	[1] b finite	[2] $0 < \epsilon \leq \mathrm{cost}$		

SOME RIGHTS RESERVED

Search Strategies; the XKCD Take

- **More Search Strategies?:** (from `https://xkcd.com/2407/`)



6.5 Informed Search Strategies

Summary: Uninformed Search/Informed Search

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored.
- Variety of uninformed search strategies.
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms.
- **Next Step:** Introduce additional knowledge about the problem (heuristic search)
  - Best-first-, A^*-strategies (guide the search by heuristics)
  - Iterative improvement algorithms.
- **Definition 5.1.** A search algorithm is called informed, iff it uses some form of external information – that is not part of the search problem – to guide the search.

SOME RIGHTS RESERVED

6.5.1 Greedy Search

Best-first search

- **Idea:** Order the fringe by estimated “desirability” (Expand most desirable unexpanded node)
- **Definition 5.2.** An evaluation function assigns a desirability value to each node of the search tree.
- **Note:** A evaluation function is not part of the search problem, but must be added externally.
- **Definition 5.3.** In best first search, the fringe is a queue sorted in decreasing order of desirability.
- **Special cases:** Greedy search, A^* search

SOME RIGHTS RESERVED

Greedy search

- **Idea:** Expand the node that *appears* to be closest to the goal.
- **Definition 5.4.** A heuristic is an evaluation function h on states that estimates the cost from n to the nearest goal state. We speak of heuristic search if the search algorithm uses a heuristic in some way.
- **Note:** All nodes for the same state must have the same h-value!
- **Definition 5.5.** Given a heuristic h, greedy search is the strategy where the fringe is organized as a queue sorted by increasing h value.
- **Example 5.6.** Straight-line distance from/to Bucharest.
- **Note:** Unlike uniform cost search the node evaluation function has nothing to do with the nodes expanded so far

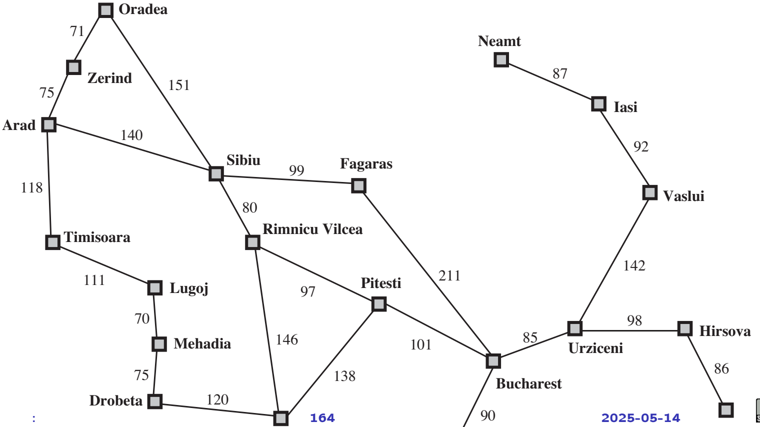
$$\begin{aligned}\text{internal search control} &\leadsto \text{external search control}\\ \text{partial solution cost} &\leadsto \text{goal cost estimation}\end{aligned}$$

SOME RIGHTS RESERVED

Romania with Straight-Line Distances

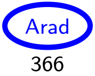
- **Example 5.7 (Informed Travel).**
  $h_{\mathrm{SLD}}(n) = \mathit{straight} - \mathit{line\ distance\ to\ Bucharest}$

Arad	366	Mehadia	241	Bucharest	0	Neamt	234
Craiova	160	Oradea	380	Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193	Fragaras	176	Sibiu	253
Giurgiu	77	Timisoara	329	Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199	Lugoj	244	Zerind	374



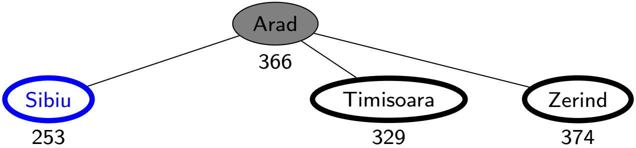
SOME RIGHTS RESERVED

Greedy Search: Romania



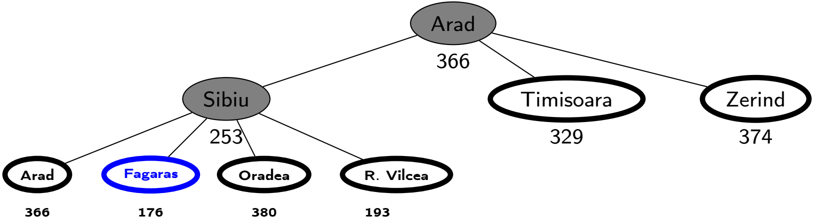
SOME RIGHTS RESERVED

Greedy Search: Romania

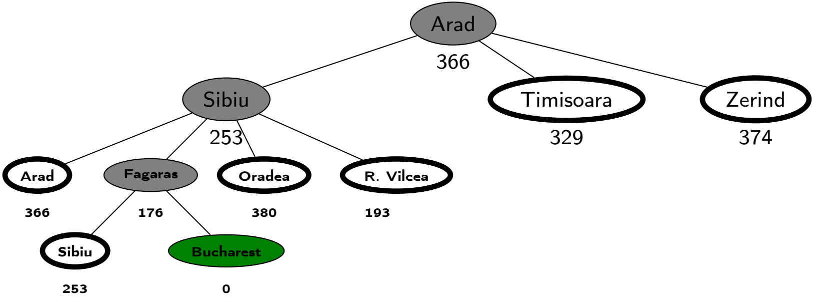


SOME RIGHTS RESERVED

Greedy Search: Romania

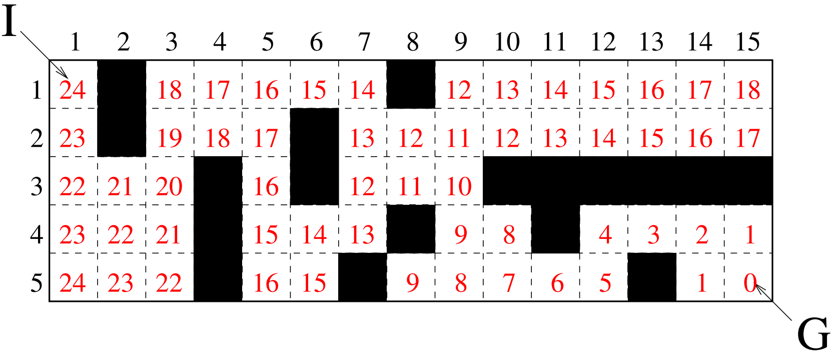


Greedy Search: Romania



Heuristic Functions in Path Planning

- **Example 5.8 (The maze solved).** We indicate h^* by giving the goal distance:



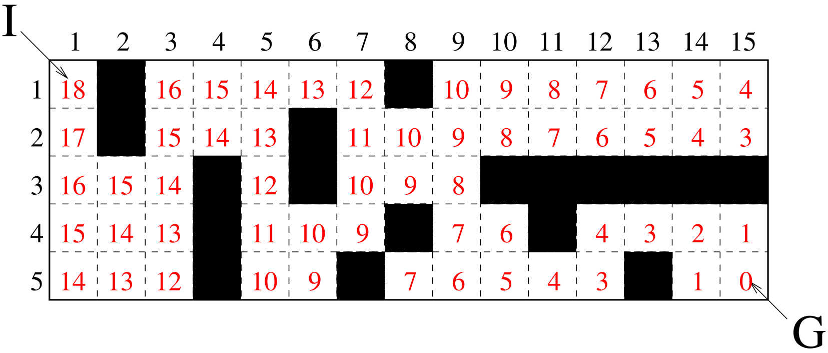
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	24	■	18	17	16	15	14	■	12	13	14	15	16	17	18
2	23	■	19	18	17	■	13	12	11	12	13	14	15	16	17
3	22	21	20	■	16	■	12	11	10	■	■	■	■	■	■
4	23	22	21	■	15	14	13	■	9	8	■	4	3	2	1
5	24	23	22	■	16	15	■	9	8	7	6	5	■	1	0

- **Example 5.9 (Maze Heuristic: The good case).** We use the Manhattan distance to the goal as a heuristic:

SOME RIGHTS RESERVED

Heuristic Functions in Path Planning

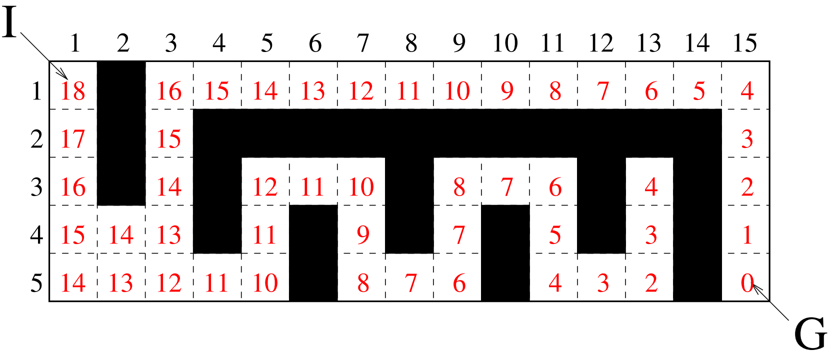
- **Example 5.11 (The maze solved).** We indicate h^* by giving the goal distance:
- **Example 5.12 (Maze Heuristic: The good case).** We use the Manhattan distance to the goal as a heuristic:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	■	16	15	14	13	12	■	10	9	8	7	6	5	4
2	17	■	15	14	13	■	11	10	9	8	7	6	5	4	3
3	16	15	14	■	12	■	10	9	8	■	■	■	■	■	■
4	15	14	13	■	11	10	9	■	7	6	■	4	3	2	1
5	14	13	12	■	10	9	■	7	6	5	4	3	■	1	0

Heuristic Functions in Path Planning

- **Example 5.14 (The maze solved).** We indicate h^* by giving the goal distance:
- **Example 5.15 (Maze Heuristic: The good case).** We use the Manhattan distance to the goal as a heuristic:
- **Example 5.16 (Maze Heuristic: The bad case).** We use the Manhattan distance to the goal as a heuristic again:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	■	16	15	14	13	12	11	10	9	8	7	6	5	4
2	17	■	15	■	■	■	■	■	■	■	■	■	■	■	3
3	16	■	14	■	12	11	10	■	8	7	6	■	4	■	2
4	15	14	13	■	11	■	9	■	7	■	5	■	3	■	1
5	14	13	12	11	10	■	8	7	6	■	4	3	2	■	0

Greedy search: Properties

Completeness	No: Can get stuck in infinite loops. Complete in finite state spaces with repeated state checking
Time complexity	$\mathcal{O}(b^m)$
Space complexity	$\mathcal{O}(b^m)$
Optimality	No

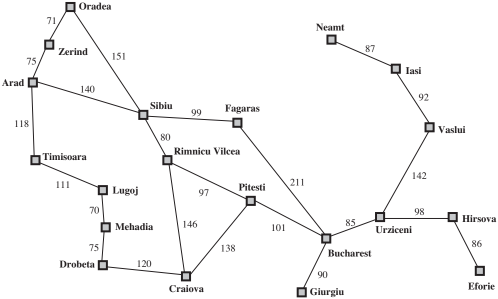
SOME RIGHTS RESERVED

Greedy search: Properties

-

Completeness	No: Can get stuck in infinite loops. Complete in finite state spaces with repeated state checking
Time complexity	$\mathcal{O}(b^m)$
Space complexity	$\mathcal{O}(b^m)$
Optimality	No

- **Example 5.18.** Greedy search can get stuck going from Iasi to Oradea: Iasi → Neamt → Iasi → Neamt → $\cdots$



Greedy search: Properties

Completeness	No: Can get stuck in infinite loops. Complete in finite state spaces with repeated state checking
Time complexity	$\mathcal{O}(b^m)$
Space complexity	$\mathcal{O}(b^m)$
Optimality	No

- **Example 5.19.** Greedy search can get stuck going from Iasi to Oradea:
  Iasi → Neamt → Iasi → Neamt → $\cdots$
- **Worst-case Time:** Same as depth first search.
- **Worst-case Space:** Same as breadth first search. ($\leadsto$ repeated state checking)
- **But:** A good heuristic can give dramatic improvements.

SOME RIGHTS RESERVED

6.5.2 Heuristics and their Properties

SOME RIGHTS RESERVED

Heuristic Functions

- **Definition 5.20.** Let Π be a search problem with states $\mathcal{S}$. A heuristic function (or short heuristic) for Π is a function $h\colon \mathcal{S} \to \mathbb{R}_0^+ \cup \{\infty\}$ so that $h(s) = 0$ whenever s is a goal state.
- $h(s)$ is intended as an estimate the distance between state s and the nearest goal state.
- **Definition 5.21.** Let Π be a search problem with states $\mathcal{S}$, then the function $h^*\colon S \to \mathbb{R}_0^+ \cup \{\infty\}$, where $h^*(s)$ is the cost of a cheapest path from s to a goal state, or ∞ if no such path exists, is called the goal distance function for Π.
- **Notes:**
  - $h(s) = 0$ on goal states: If your estimator returns "I think it's still a long way" on a goal state, then its intelligence is, um . . .
  - Return value ∞: To indicate dead ends, from which the goal state can't be reached anymore.
  - The distance estimate depends only on the state s, not on the node (i.e., the path we took to reach s).

SOME RIGHTS RESERVED

Where does the word "Heuristic" come from?

- Ancient Greek word $\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$ ($\widehat{=}$ "I find") (aka. $\epsilon\upsilon\rho\epsilon\kappa\alpha$!)
- Popularized in modern science by George Polya: "How to solve it" [**Polya:htsi73**]
- Same word often used for "rule of thumb" or "imprecise solution method".

SOME RIGHTS RESERVED

Heuristic Functions: The Eternal Trade-Off

- "Distance Estimate"? (h is an arbitrary function in principle)
  - In practice, we want it to be *accurate* (aka: *informative*), i.e., close to the actual goal distance.
  - We also want it to be fast, i.e., a small overhead for computing h.
  - These two wishes are in contradiction!
- **Example 5.22 (Extreme cases).**
  - $h = 0$: no overhead at all, completely un-informative.
  - $h = h^*$: perfectly accurate, overhead $\widehat{=}$ solving the problem in the first place.
- **Observation 5.23.** *We need to trade off the accuracy of h against the overhead for computing it.*

Properties of Heuristic Functions

- **Definition 5.24.** Let Π be a search problem with states S and actions A. We say that a heuristic h for Π is admissible if $h(s) \leq h^*(s)$ for all $s \in S$.
  We say that h is consistent if $h(s) - h(s') \leq c(a)$ for all $s \in S$, $a \in A$, and $s' \in \mathcal{T}(s, a)$.
- **In other words . . . :**
  - h is admissible if it is a lower bound on goal distance.
  - h is consistent if, when applying an action a, the heuristic value cannot decrease by more than the cost of a.

SOME RIGHTS RESERVED

Properties of Heuristic Functions, ctd.

- Let Π be a search problem, and let h be a heuristic for Π. If h is consistent, then h is admissible.
- *Proof:* we prove $h(s) \leq h^*(s)$ for all $s \in S$ by induction over the length of the cheapest path to a goal node.
  1. base case
     1.1. $h(s) = 0$ by definition of heuristic, so $h(s) \leq h^*(s)$ as desired.
  3. step case
     3.1. We assume that $h(s') \leq h^*(s)$ for all states s' with a cheapest goal node path of length n.
     3.2. Let s be a state whose cheapest goal path has length $n+1$ and the first transition is $o = (s,s')$.
     3.3. By consistency, we have $h(s) - h(s') \leq c(o)$ and thus $h(s) \leq h(s') + c(o)$.
     3.4. By construction, $h^*(s)$ has a cheapest goal path of length n and thus, by induction hypothesis $h(s') \leq h^*(s')$.
     3.5. By construction, $h^*(s) = h^*(s') + c(o)$.
     3.6. Together this gives us $h(s) \leq h^*(s)$ as desired.

☐

- Consistency is a sufficient condition for admissibility (easier to check)

SOME RIGHTS RESERVED

Properties of Heuristic Functions: Examples

- **Example 5.25.** Straight line distance is admissible and consistent by the triangle inequality.
  If you drive 100km, then the straight line distance to Rome can't decrease by more than 100km.
- **Observation:** In practice, admissible heuristics are typically consistent.
- **Example 5.26 (An admissible, but inconsistent heuristic).** When traveling to Rome, let $h(Munich) = 300$ and $h(Innsbruck) = 100$.
- **Inadmissible heuristics** typically arise as approximations of admissible heuristics that are too costly to compute. (see later)

SOME RIGHTS RESERVED

6.5.3 A-Star Search

SOME RIGHTS RESERVED

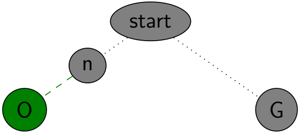
A^* Search: Evaluation Function

- **Idea:** Avoid expanding paths that are already expensive (make use of actual cost)
  The simplest way to combine heuristic and path cost is to simply add them.
- **Definition 5.27.** The evaluation function for A^* search is given by $f(n) = g(n) + h(n)$, where $g(n)$ is the path cost for n and $h(n)$ is the estimated cost to the nearest goal from n.
- Thus $f(n)$ is the estimated total cost of the path through n to a goal.
- **Definition 5.28.** Best first search with evaluation function $g + h$ is called A^* search.

SOME RIGHTS RESERVED

A^* Search: Optimality

- **Theorem 5.29.** *A^* search with admissible heuristic is optimal.*
- *Proof:* We show that suboptimal goal nodes are never expanded by A^*
  1. Suppose a suboptimal goal node G has been generated then we are in the following situation:

  2. Let n be an unexpanded node on a path to an optimality goal node O, then

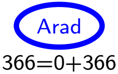
$$\begin{array}{ll}
f(G) = g(G) & \text{since } h(G) = 0 \\
g(G) > g(O) & \text{since } G \text{ suboptimal} \\
g(O) = g(n) + h^*(n) & n \text{ on optimal path} \\
g(n) + h^*(n) \geq g(n) + h(n) & \text{since } h \text{ is admissible} \\
g(n) + h(n) = f(n) &
\end{array}$$

  3. Thus, $f(G) > f(n)$ and A^* never expands G.

□

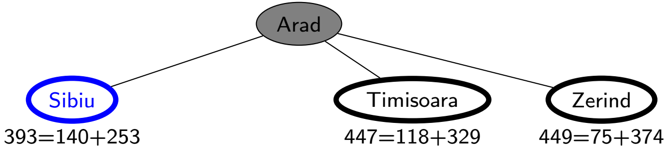
SOME RIGHTS RESERVED

A^* Search Example

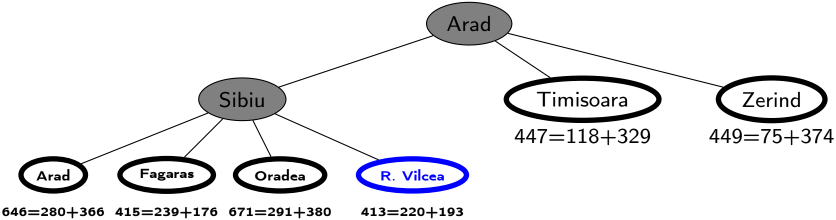


SOME RIGHTS RESERVED

A^* Search Example

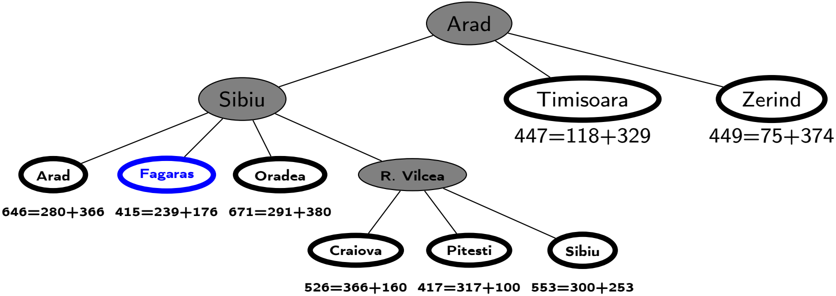


A^* Search Example

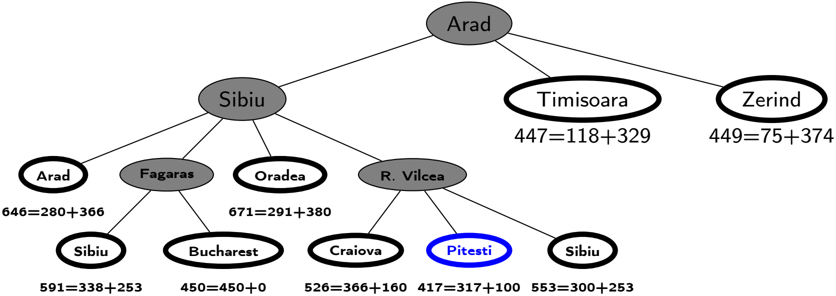


SOME RIGHTS RESERVED

A^* Search Example

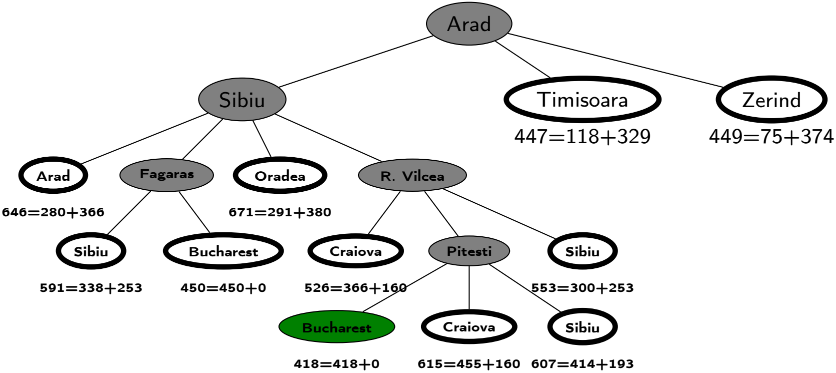


A^* Search Example



SOME RIGHTS RESERVED

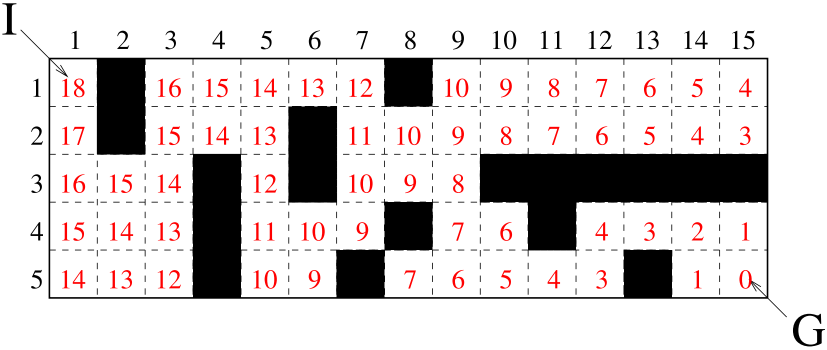
A^* Search Example



SOME RIGHTS RESERVED

Additional Observations (Not Limited to Path Planning)

- **Example 5.30 (Greedy best-first search, “good case”).**

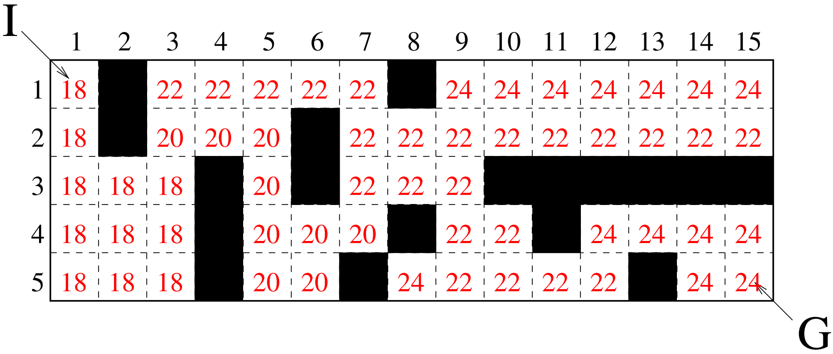


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	■	16	15	14	13	12	■	10	9	8	7	6	5	4
2	17	■	15	14	13	■	11	10	9	8	7	6	5	4	3
3	16	15	14	■	12	■	10	9	8	■	■	■	■	■	■
4	15	14	13	■	11	10	9	■	7	6	■	4	3	2	1
5	14	13	12	■	10	9	■	7	6	5	4	3	■	1	0

We will find a solution with little search.

Additional Observations (Not Limited to Path Planning)

- **Example 5.31 (A^* ($g + h$), “good case”).**



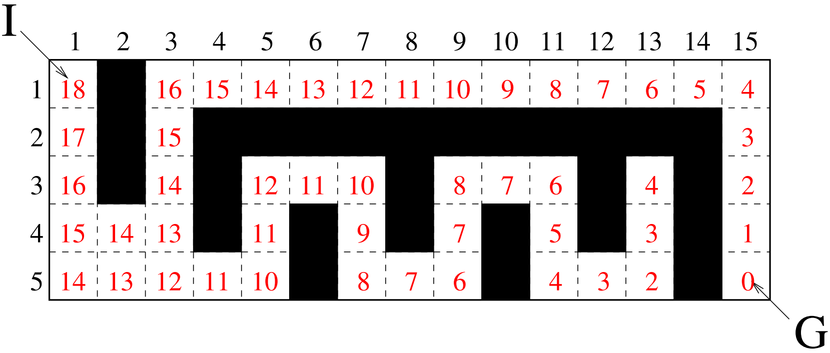
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	■	22	22	22	22	22	■	24	24	24	24	24	24	24
2	18	■	20	20	20	■	22	22	22	22	22	22	22	22	22
3	18	18	18	■	20	■	22	22	22	■	■	■	■	■	■
4	18	18	18	■	20	20	20	■	22	22	■	24	24	24	24
5	18	18	18	■	20	20	■	24	22	22	22	22	■	24	24

  - In A^* with a consistent heuristic, $g + h$ always increases monotonically (*h* cannot decrease more than *g* increases)
  - We need more search, in the “right upper half”. This is typical: Greedy best first search tends to be faster than A^*.

SOME RIGHTS RESERVED

Additional Observations (Not Limited to Path Planning)

- **Example 5.32 (Greedy best-first search, “bad case”).**



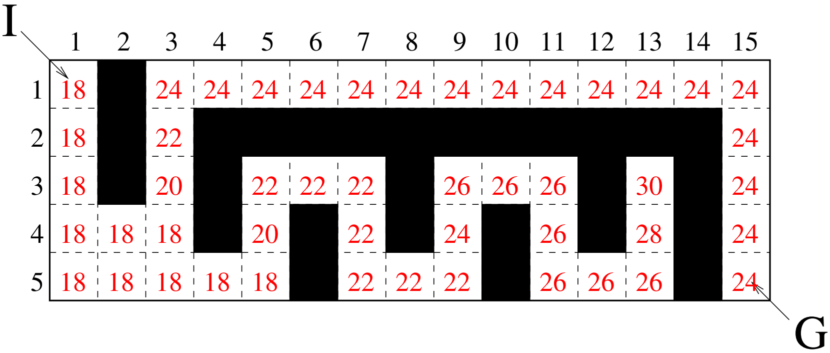
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	■	16	15	14	13	12	11	10	9	8	7	6	5	4
2	17	■	15	■	■	■	■	■	■	■	■	■	■	■	3
3	16	■	14	■	12	11	10	■	8	7	6	■	4	■	2
4	15	14	13	■	11	■	9	■	7	■	5	■	3	■	1
5	14	13	12	11	10	■	8	7	6	■	4	3	2	■	0

Search will be mis-guided into the “dead-end street”.

SOME RIGHTS RESERVED

Additional Observations (Not Limited to Path Planning)

- **Example 5.33 (A^* ($g+h$), "bad case").**

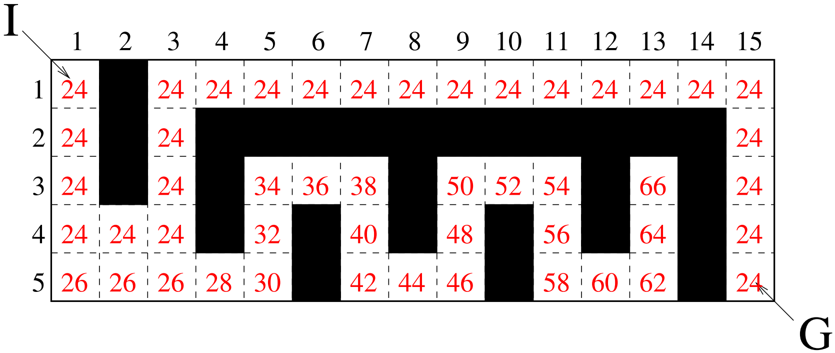


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18		24	24	24	24	24	24	24	24	24	24	24	24	24
2	18		22												24
3	18		20		22	22	22		26	26	26		30		24
4	18	18	18		20		22		24		26		28		24
5	18	18	18	18	18		22	22	22		26	26	26		24

We will search less of the "dead-end street". Sometimes $g+h$ gives better search guidance than h. ($\rightsquigarrow$ A^* is faster there)

Additional Observations (Not Limited to Path Planning)

- **Example 5.34 (A^* $(g+h)$ using h^*).**

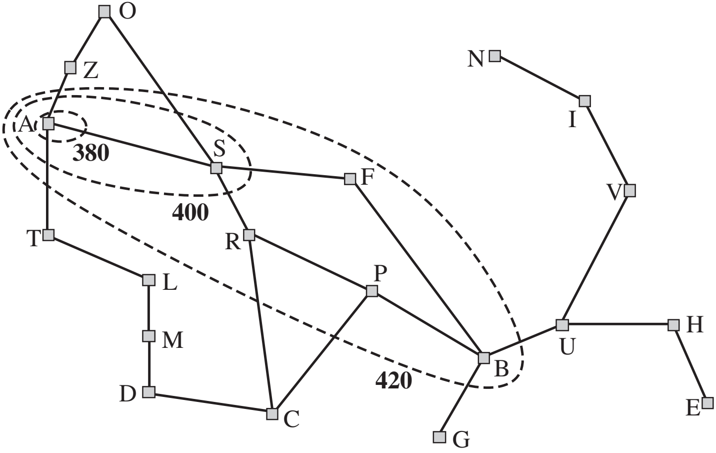


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	24	■	24	24	24	24	24	24	24	24	24	24	24	24	24
2	24	■	24	■	■	■	■	■	■	■	■	■	■	■	24
3	24	■	24	■	34	36	38	■	50	52	54	■	66	■	24
4	24	24	24	■	32	■	40	■	48	■	56	■	64	■	24
5	26	26	26	28	30	■	42	44	46	■	58	60	62	■	24

In A^*, node values always increase monotonically (with any heuristic). If the heuristic is perfect, they remain constant on optimal paths.

A^* search: f-contours

- **Intuition:** A^*-search gradually adds “f-contours” (areas of the same f-value) to the search.



SOME RIGHTS RESERVED

A^* search: Properties

- Properties or A^*-search:

Completeness	Yes (unless there are infinitely many nodes n with $f(n) \leq f(0)$)
Time complexity	Exponential in [relative error in h × length of solution]
Space complexity	Same as time (variant of BFS)
Optimality	Yes

n

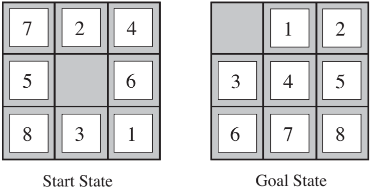
- A^*-search expands all (some/no) nodes with $f(n) < h^*(n)$
- The run-time depends on how well we approximated the real cost h^* with h.

SOME RIGHTS RESERVED

6.5.4 Finding Good Heuristics

SOME RIGHTS RESERVED

Admissible heuristics: Example 8-puzzle



- **Example 5.35.** Let $h_1(n)$ be the number of misplaced tiles in node n. ($h_1(S) = 9$)
- **Example 5.36.** Let $h_2(n)$ be the total Manhattan distance from desired location of each tile. ($h_2(S) = 3+1+2+2+2+3+2+2+3 = 20$)
- **Observation 5.37 (Typical search costs).** (*IDS $\widehat{=}$ iterative deepening search*)

nodes explored	*IDS*	$A^*(h_1)$	$A^*(h_2)$
$d = 14$	*3,473,941*	*539*	*113*
$d = 24$	*too many*	*39,135*	*1,641*

SOME RIGHTS RESERVED

Dominance

- **Definition 5.38.** Let h_1 and h_2 be two admissible heuristics we say that h_2 dominates h_1 if $h_2(n) \geq h_1(n)$ for all n.
- **Theorem 5.39.** *If h_2 dominates h_1, then h_2 is better for search than h_1.*
- *Proof sketch:* If h_2 dominates h_1, then h_2 is “closer to h^*” than h_1, which means better search performance.

SOME RIGHTS RESERVED

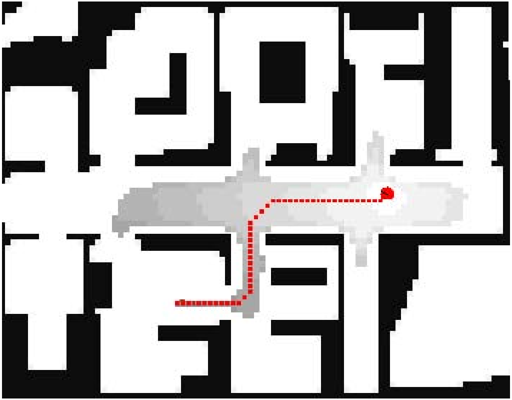
Relaxed problems

- **Observation:** Finding good admissible heuristics is an art!
- **Idea:** Admissible heuristics can be derived from the *exact* solution cost of a relaxed version of the problem.
- **Example 5.40.** If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then we get heuristic h_1.
- **Example 5.41.** If the rules are relaxed so that a tile can move to *any adjacent square*, then we get heuristic h_2. (Manhattan distance)
- **Definition 5.42.** Let $\Pi := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ be a search problem, then we call a search problem $\mathcal{P}^r := \langle \mathcal{S}, \mathcal{A}^r, \mathcal{T}^r, \mathcal{I}^r, \mathcal{G}^r \rangle$ a relaxed problem (wrt. Π; or simply relaxation of Π), iff $\mathcal{A} \subseteq \mathcal{A}^r$, $\mathcal{T} \subseteq \mathcal{T}^r$, $\mathcal{I} \subseteq \mathcal{I}^r$, and $\mathcal{G} \subseteq \mathcal{G}^r$.
- **Lemma 5.43.** *If $\mathcal{P}^r$ relaxes Π, then every solution for Π is one for $\mathcal{P}^r$.*
- **Key point:** The optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem.

SOME RIGHTS RESERVED

Empirical Performance: A^* in Path Planning

- **Example 5.44 (Live Demo vs. Breadth-First Search).**



See `http://qiao.github.io/PathFinding.js/visual/`

- **Difference to Breadth-first Search?:** That would explore all grid cells in a *circle* around the initial state!

SOME RIGHTS RESERVED

6.6 Local Search

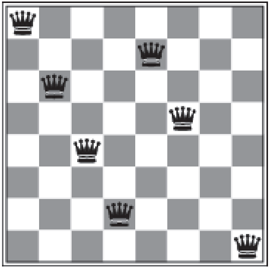
Systematic Search vs. Local Search

- **Definition 6.1.** We call a search algorithm systematic, if it considers all states at some point.
- **Example 6.2.** All tree search algorithms (except pure depth first search) are systematic. (given reasonable assumptions e.g. about costs.)
- **Observation 6.3.** *Systematic search algorithms are complete.*
- **Observation 6.4.** *In systematic search algorithms there is no limit of the number of nodes that are kept in memory at any time.*
- **Alternative:** Keep only one (or a few) nodes at a time
  - $\rightsquigarrow$ no systematic exploration of all options, $\rightsquigarrow$ incomplete.

SOME RIGHTS RESERVED

Local Search Problems

- **Idea:** Sometimes the path to the solution is irrelevant.
- **Example 6.5 (8 Queens Problem).** Place 8 queens on a chess board, so that no two queens threaten each other.
- This problem has various solutions (the one of the right isn't one of them)
- **Definition 6.6.** A local search algorithm is a search algorithm that operates on a single state, the current state (rather than multiple paths). (advantage: constant space)

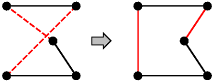


- Typically local search algorithms only move to successor of the current state, and do not retain search paths.
- Applications include: integrated circuit design, factory-floor layout, job-shop scheduling, portfolio management, fleet deployment,. . .

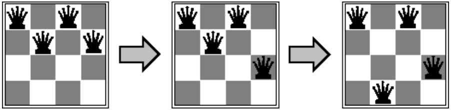
SOME RIGHTS RESERVED

Local Search: Iterative improvement algorithms

- **Definition 6.7.** The traveling salesman problem (TSP is to find shortest trip through set of cities such that each city is visited exactly once.
- **Idea:** Start with any complete tour, perform pairwise exchanges



- **Definition 6.8.** The n-queens problem is to put n queens on $n \times n$ board such that no two queen in the same row, columns, or diagonal.
- **Idea:** Move a queen to reduce number of conflicts



SOME RIGHTS RESERVED

Hill-climbing (gradient ascent/descent)

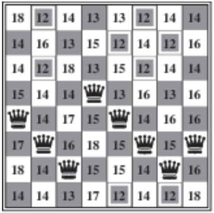
- **Idea:** Start anywhere and go in the direction of the steepest ascent.
- **Definition 6.9.** Hill climbing (also gradient ascent) is a local search algorithm that iteratively selects the best successor:

```
procedure Hill−Climbing (problem) /∗ a state that is a local minimum ∗/
  local current, neighbor /∗ nodes ∗/
  current := Make−Node(Initial−State[problem])
  loop
    neighbor := <a highest−valued successor of current>
    if Value[neighbor] < Value[current] return [current] end if
    current := neighbor
  end loop
end procedure
```

- **Intuition:** Like best first search without memory.
- Works, if solutions are dense and local maxima can be escaped.

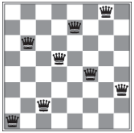
Example Hill Climbing with 8 Queens

- **Idea:** Consider $h \widehat{=}$ number of queens that threaten each other.
- **Example 6.10.** An 8-queens state with heuristic cost estimate $h = 17$ showing h-values for moving a queen within its column:



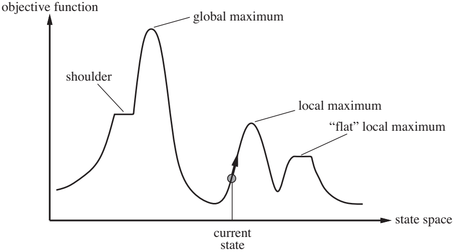
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

- **Problem:** The state space has local minima. e.g. the board on the right has $h = 1$ but every successor has $h > 1$.



Hill-climbing

- **Problem:** Depending on initial state, can get stuck on local maxima/minima and plateaux.
- "Hill-climbing search is like climbing Everest in thick fog with amnesia".

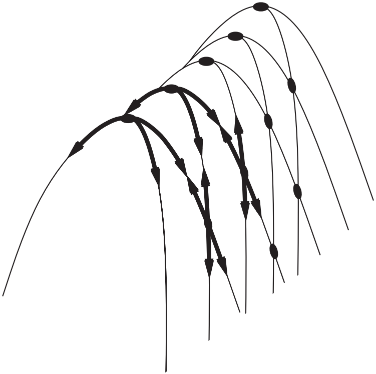


- **Idea:** Escape local maxima by allowing some "bad" or random moves.
- **Example 6.11.** local search, simulated annealing, . . .
- **Properties:** All are incomplete, nonoptimal.
- Sometimes performs well in practice (if (optimal) solutions are dense)

SOME RIGHTS RESERVED

Simulated annealing (Idea)

- **Definition 6.12.** Ridges are ascending successions of local maxima.
- **Problem:** They are extremely difficult to bv navigate for local search algorithms.
- **Idea:** Escape local maxima by allowing some "bad" moves, but gradually decrease their size and frequency.



- Annealing is the process of heating steel and let it cool gradually to give it time to grow an optimal crystal structure.
- Simulated annealing is like shaking a ping pong ball occasionally on a bumpy surface to free it. (so it does not get stuck)
- Devised by Metropolis et al for physical process modelling [**MetRosRos:escfcm53**]
- Widely used in VLSI layout, airline scheduling, etc.

SOME RIGHTS RESERVED

Simulated annealing (Implementation)

- **Definition 6.13.** The following algorithm is called simulated annealing:

```
procedure Simulated−Annealing (problem,schedule) /∗ a solution state ∗/
  local node, next /∗ nodes ∗/
   local T /∗ a ‘‘temperature’’ controlling prob.~of downward steps ∗/
   current := Make−Node(Initial−State[problem])
   for t :=1 to ∞
    T := schedule[t]
      if T = 0 return current end if
      next := <a randomly selected successor of current>
     Δ(E) := Value[next]−Value[current]
      if Δ(E) > 0 current := next
      else
       current := next <only with probability> $e^{\Delta(E)/T}$
     end if
  end for
end procedure
```

A schedule is a mapping from time to “temperature”.

SOME RIGHTS RESERVED

Properties of simulated annealing

- At fixed "temperature" T, state occupation probability reaches Boltzman distribution

  $$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

  T decreased slowly enough $\leadsto$ always reach best state x^* because

  $$\frac{e^{\frac{E(x^*)}{kT}}}{e^{\frac{E(x)}{kT}}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$$

  for small T.
- **Question:** Is this necessarily an interesting guarantee?

SOME RIGHTS RESERVED

Local beam search

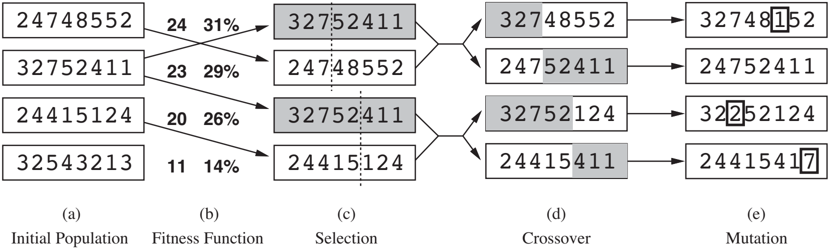
- **Definition 6.14.** Local beam search is a search algorithm that keep k states instead of 1 and chooses the top k of all their successors.
- **Observation:** Local beam search is not the same as k searches run in parallel! (Searches that find good states recruit other searches to join them)
- **Problem:** Quite often, all k searches end up on the same local hill!
- **Idea:** Choose k successors randomly, biased towards good ones. (Observe the close analogy to natural selection!)

SOME RIGHTS RESERVED

Genetic algorithms (very briefly)

- **Definition 6.15.** A genetic algorithm is a variant of local beam search that generates successors by
  - randomly modifying states (mutation)
  - mixing pairs of states (sexual reproduction or crossover)

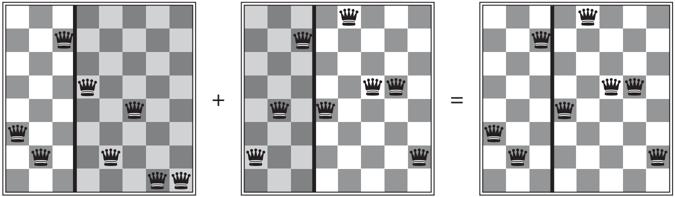
  to optimize a fitness function. (survival of the fittest)
- **Example 6.16.** Generating successors for 8 queens



SOME RIGHTS RESERVED

Genetic algorithms (continued)

- **Problem:** Genetic algorithms require states encoded as strings.
- Crossover only helps iff substrings are meaningful components.
- **Example 6.17 (Evolving 8 Queens).** First crossover



- **Note:** Genetic algorithms $\neq$ evolution: e.g., real genes also encode replication machinery!

Chapter 7
Adversarial Search for Game Playing

SOME RIGHTS RESERVED

7.1 Introduction

SOME RIGHTS RESERVED

The Problem

- **The Problem of Game-Play:** cf.
- **Example 1.1.**

- **Definition 1.2.** Adversarial search $\widehat{=}$ Game playing against an opponent.

SOME RIGHTS RESERVED

Why Game Playing?

- **What do you think?**
  - Playing a game well clearly requires a form of “intelligence”.
  - Games capture a pure form of competition between opponents.
  - Games are abstract and precisely defined, thus very easy to formalize.
- Game playing is one of the oldest sub-areas of AI (ca. 1950).
- The dream of a machine that plays chess is, indeed, *much* older than AI!

“Schachtürke” (1769)

“El Ajedrecista” (1912)

SOME RIGHTS RESERVED

"Game" Playing? *Which* Games?

- . . . sorry, we're not gonna do soccer here.
- **Definition 1.3 (Restrictions).** A game in the sense of AI-1 is one where
  - Game state discrete, number of game state finite.
  - Finite number of possible moves.
  - The game state is fully observable.
  - The outcome of each move is deterministic.
  - Two players: Max and Min.
  - Turn-taking: It's each player's turn alternatingly. Max begins.
  - Terminal game states have a utility u. Max tries to maximize u, Min tries to minimize u.
  - In that sense, the utility for Min is the exact opposite of the utility for Max ("zero sum").
  - There are no infinite runs of the game (no matter what moves are chosen, a terminal state is reached after a finite number of moves).

SOME RIGHTS RESERVED

An Example Game

- Game states: Positions of figures.
- Moves: Given by rules.
- Players: white (Max), black (Min).
- Terminal states: checkmate.
- Utility of terminal states, e.g.:
  - +100 if black is checkmated.
  - 0 if stalemate.
  - −100 if white is checkmated.

SOME RIGHTS RESERVED

“Game” Playing? Which Games *Not*?

- Soccer (sorry guys; not even RoboCup)
- Important types of games that we don’t tackle here:
  - Chance. (E.g., backgammon)
  - More than two players. (E.g., Halma)
  - Hidden information. (E.g., most card games)
  - Simultaneous moves. (E.g., Diplomacy)
  - Not zero-sum, i.e., outcomes may be beneficial (or detrimental) for both players. (cf. Game theory: Auctions, elections, economy, politics, ...)
- Many of these more general game types can be handled by similar/extended algorithms.

SOME RIGHTS RESERVED

(A Brief Note On) Formalization

- **Definition 1.4.** An adversarial search problem is a search problem $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, where
  1. $\mathcal{S} = \mathcal{S}^{\mathrm{Max}} \uplus \mathcal{S}^{\mathrm{Min}} \uplus \mathcal{G}$ and $\mathcal{A} = \mathcal{A}^{\mathrm{Max}} \uplus \mathcal{A}^{\mathrm{Min}}$
  2. For $a \in \mathcal{A}^{\mathrm{Max}}$, if $s \xrightarrow{a} s'$ then $s \in \mathcal{S}^{\mathrm{Max}}$ and $s' \in (\mathcal{S}^{\mathrm{Min}} \cup \mathcal{G})$.
  3. For $a \in \mathcal{A}^{\mathrm{Min}}$, if $s \xrightarrow{a} s'$ then $s \in \mathcal{S}^{\mathrm{Min}}$ and $s' \in (\mathcal{S}^{\mathrm{Max}} \cup \mathcal{G})$.

  together with a game utility function $u \colon \mathcal{G} \to \mathbb{R}$.
- **Definition 1.5 (Commonly used terminology).** position $\widehat{=}$ state, move $\widehat{=}$ action, end state $\widehat{=}$ terminal state $\widehat{=}$ goal state.
- **Remark:** A round of the game – one move Max, one move Min – is often referred to as a "move", and individual actions as "half-moves" (we *don't* in AI-1)

SOME RIGHTS RESERVED

Why Games are Hard to Solve: I

- What is a “solution” here?
- **Definition 1.6.** Let Θ be an adversarial search problem, and let $X \in \{\mathrm{Max}, \mathrm{Min}\}$. A strategy for X is a function $\sigma^X : \mathcal{S}^X \to \mathcal{A}^X$ so that a is applicable to s whenever $\sigma^X(s) = a$.
- We don’t know how the opponent will react, and need to prepare for all possibilities.
- **Definition 1.7.** A strategy is called optimal if it yields the best possible utility for X assuming perfect opponent play (not formalized here).
- **Problem:** In (almost) all games, computing an optimal strategy is infeasible. (state/search tree too huge)
- **Solution:** Compute the next move “on demand”, given the current state instead.

SOME RIGHTS RESERVED

Why Games are hard to solve II

- **Example 1.8.** Number of reachable states in chess: 10^{40}.
- **Example 1.9.** Number of reachable states in go: 10^{100}.
- **It's even worse:** Our algorithms here look at search trees (game trees), no duplicate pruning.
- **Example 1.10.**
  - Chess without duplicate pruning: $35^{100} \simeq 10^{154}$.
  - Go without duplicate pruning: $200^{300} \simeq 10^{690}$.

SOME RIGHTS RESERVED

How To Describe a Game State Space?

- Like for classical search problems, there are three possible ways to describe a game: blackbox/API description, declarative description, explicit game state space.
- **Question:** Which ones do humans use?
  - Explicit $\approx$ Hand over a book with all 10^{40} moves in chess.
  - Blackbox $\approx$ Give possible chess moves on demand but don't say how they are generated.
- **Answer:** Declarative!
  With "game description language" $\widehat{=}$ natural language.

SOME RIGHTS RESERVED

Specialized vs. General Game Playing

- And which game descriptions do computers use?
  - Explicit: Only in illustrations.
  - Blackbox/API: Assumed description in (This Chapter)
    - Method of choice for all those game players out there in the market (Chess computers, video game opponents, you name it).
    - Programs designed for, and specialized to, a particular game.
    - Human knowledge is key: evaluation functions (see later), opening databases (chess!!), end game databases.
  - Declarative: General game playing, active area of research in AI.
    - Generic game description language (GDL), based on logic.
    - Solvers are given only "the rules of the game", no other knowledge/input whatsoever (cf.).
    - Regular academic competitions since 2005.

SOME RIGHTS RESERVED

Our Agenda for This Chapter

- **Minimax Search:** How to compute an optimal strategy?
  - Minimax is the canonical (and easiest to understand) algorithm for *solving* games, i.e., computing an optimal strategy.
- **Evaluation functions:** But what if we don't have the time/memory to solve the entire game?
  - Given limited time, the best we can do is look ahead as far as we can. Evaluation functions tell us how to evaluate the leaf states at the cut off.
- **Alphabeta search:** How to prune unnecessary parts of the tree?
  - Often, we can detect early on that a particular action choice cannot be part of the optimal strategy. We can then stop considering this part of the game tree.
- **State of the art:** What is the state of affairs, for prominent games, of computer game playing vs. human experts?
  - Just FYI (not part of the technical content of this course).

SOME RIGHTS RESERVED

7.2 Minimax Search

SOME RIGHTS RESERVED

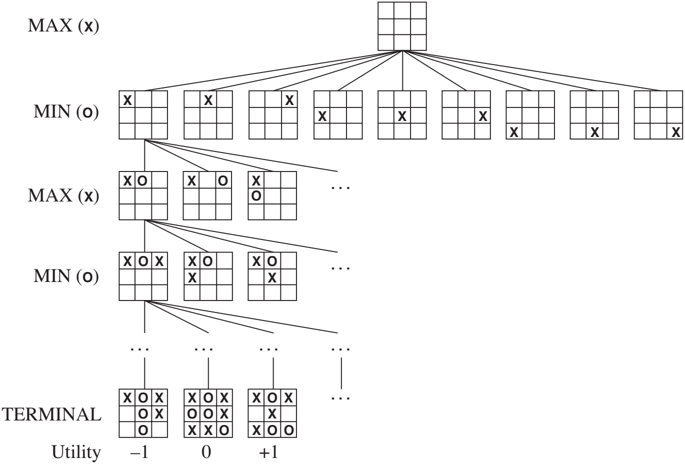
"Minimax"?

- We want to compute an optimal strategy for player "Max".
  - In other words: "*We are Max, and our opponent is Min.*"
- **Recall:** We compute the strategy offline, before the game begins. During the game, whenever it's our turn, we just look up the corresponding action.
- **Idea:** Use tree search using an extension $\hat{u}$ of the utility function u to inner nodes. $\hat{u}$ is computed recursively from u during search:
  - Max attempts to maximize $\hat{u}(s)$ of the terminal states reachable during play.
  - Min attempts to minimize $\hat{u}(s)$.
- The computation alternates between minimization and maximization ⇝ hence "minimax".

SOME RIGHTS RESERVED

Example Tic-Tac-Toe

- **Example 2.1.** A full game tree for tic-tac-toe



- current player and action marked on the left.
- Last row: terminal positions with their utility.

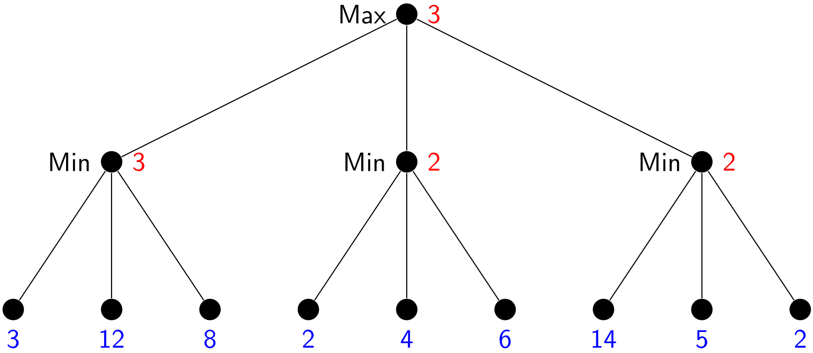
SOME RIGHTS RESERVED

Minimax: Outline

- **We max, we min, we max, we min . . .**
  1. Depth first search in game tree, with Max in the root.
  2. Apply game utility function to terminal positions.
  3. Bottom-up for each inner node n in the search tree, compute the utility $\hat{u}(n)$ of n as follows:
     - If it's Max's turn: Set $\hat{u}(n)$ to the maximum of the utilities of n's successor nodes.
     - If it's Min's turn: Set $\hat{u}(n)$ to the minimum of the utilities of n's successor nodes.
  4. Selecting a move for Max at the root: Choose one move that leads to a successor node with maximal utility.

SOME RIGHTS RESERVED

Minimax: Example



- **Blue numbers:** Utility function u applied to terminal positions.
- **Red numbers:** Utilities of inner nodes, as computed by the minimax algorithm.

The Minimax Algorithm: Pseudo-Code

- **Definition 2.2.** The minimax algorithm (often just called minimax) is given by the following functions whose argument is a state $s \in \mathcal{S}^{\mathrm{Max}}$, in which Max is to move.

```
function Minimax−Decision($s$) returns an action
  $v$ := Max−Value($s$)
  return an action yielding value $v$ in the previous function call

function Max−Value($s$) returns a utility value
  if Terminal−Test($s$) then return $u(s)$
  $v := -\infty$
  for each $a \in$ Actions($s$) do
    $v$ := max($v$,Min−Value(ChildState($s$,$a$)))
  return $v$

function Min−Value($s$) returns a utility value
  if Terminal−Test($s$) then return $u(s)$
  $v := +\infty$
  for each $a \in$ Actions($s$) do
    $v$ := min($v$,Max−Value(ChildState($s$,$a$)))
  return $v$
```

We call nodes, where Max/Min acts Max-nodes/Min-nodes.

SOME RIGHTS RESERVED

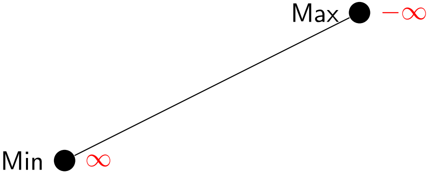
Minimax: Example, Now in Detail

Max ● $-\infty$

- So which action for Max is returned?

SOME RIGHTS RESERVED

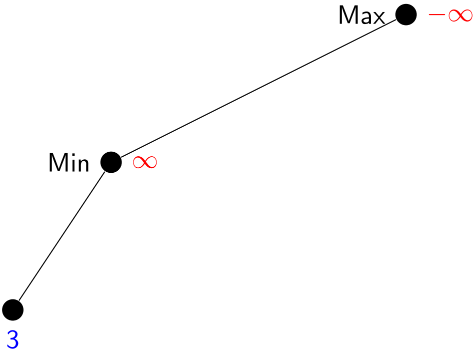
Minimax: Example, Now in Detail



- So which action for Max is returned?

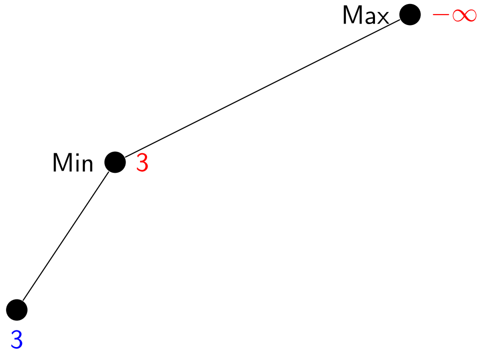
SOME RIGHTS RESERVED

Minimax: Example, Now in Detail



- So which action for Max is returned?

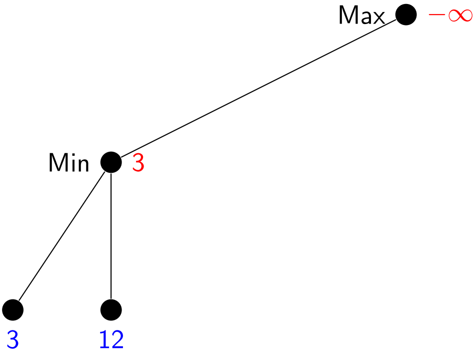
Minimax: Example, Now in Detail



- So which action for Max is returned?

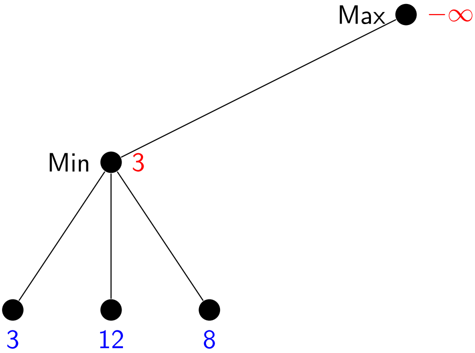
SOME RIGHTS RESERVED

Minimax: Example, Now in Detail



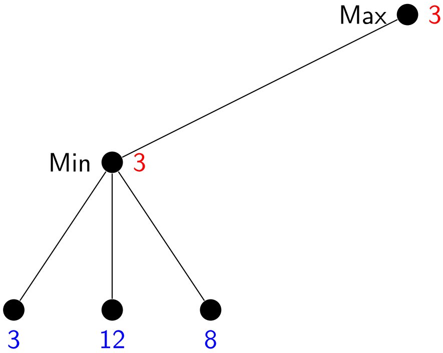
- So which action for Max is returned?

Minimax: Example, Now in Detail



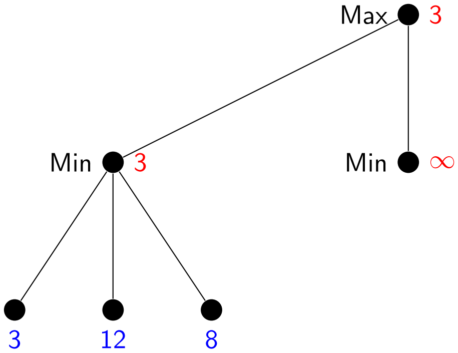
- So which action for Max is returned?

Minimax: Example, Now in Detail



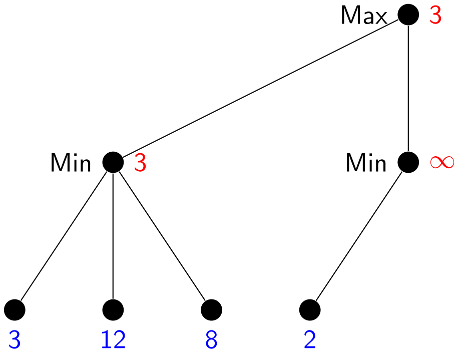
- So which action for Max is returned?

Minimax: Example, Now in Detail



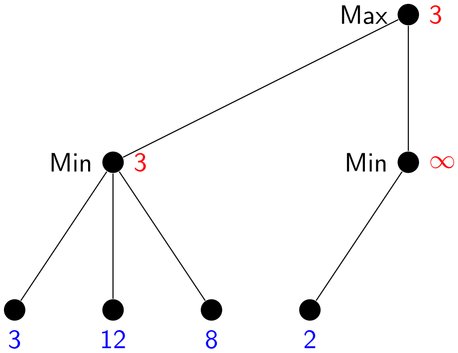
- So which action for Max is returned?

Minimax: Example, Now in Detail



- So which action for Max is returned?

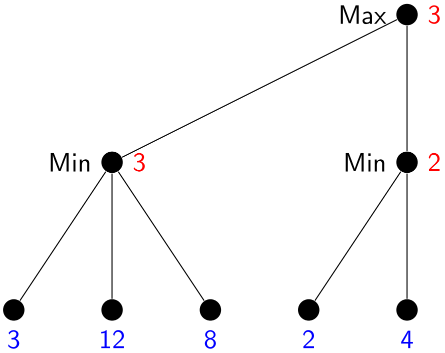
Minimax: Example, Now in Detail



- So which action for Max is returned?

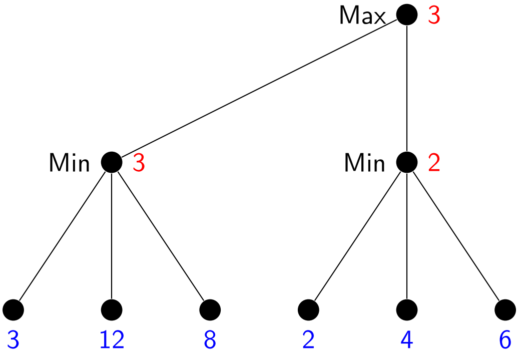
SOME RIGHTS RESERVED

Minimax: Example, Now in Detail



- So which action for Max is returned?

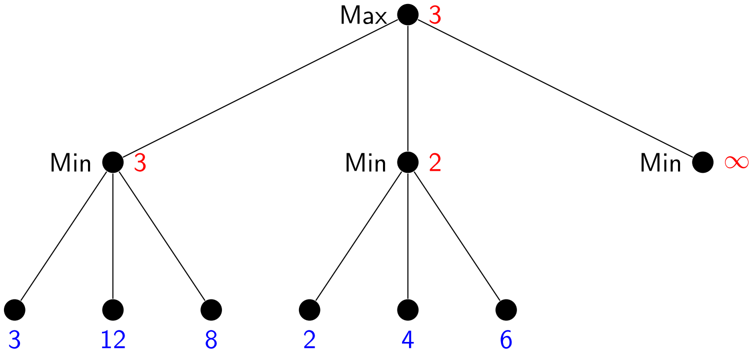
Minimax: Example, Now in Detail



- So which action for Max is returned?

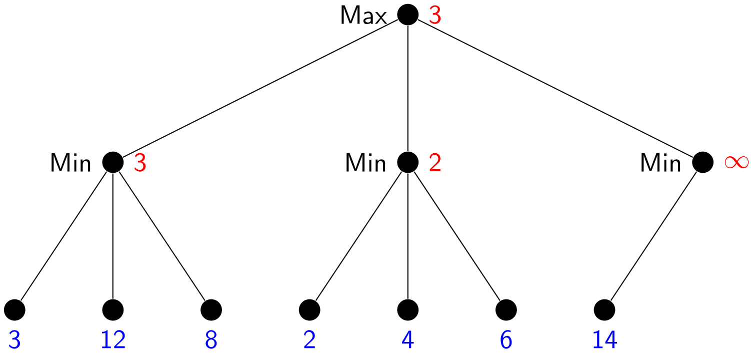
SOME RIGHTS RESERVED

Minimax: Example, Now in Detail



- So which action for Max is returned?

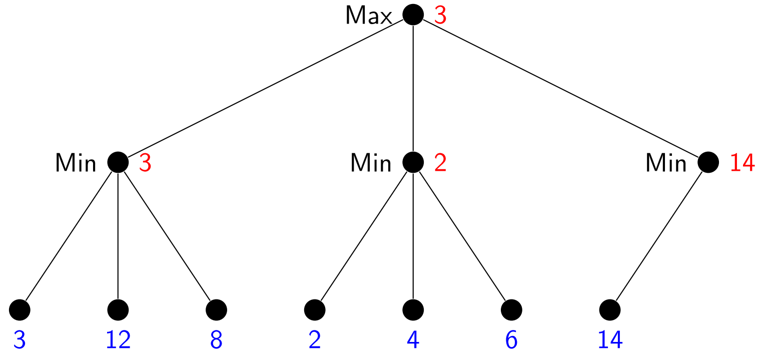
Minimax: Example, Now in Detail



- So which action for Max is returned?

SOME RIGHTS RESERVED

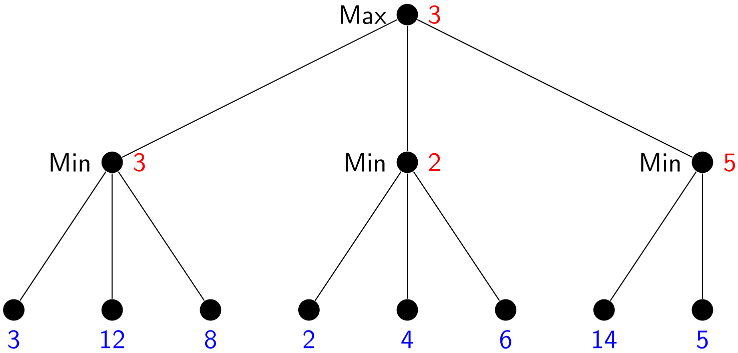
Minimax: Example, Now in Detail



- So which action for Max is returned?

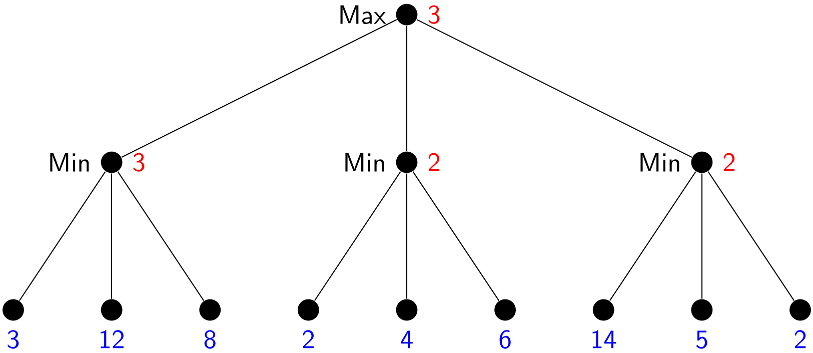
SOME RIGHTS RESERVED

Minimax: Example, Now in Detail



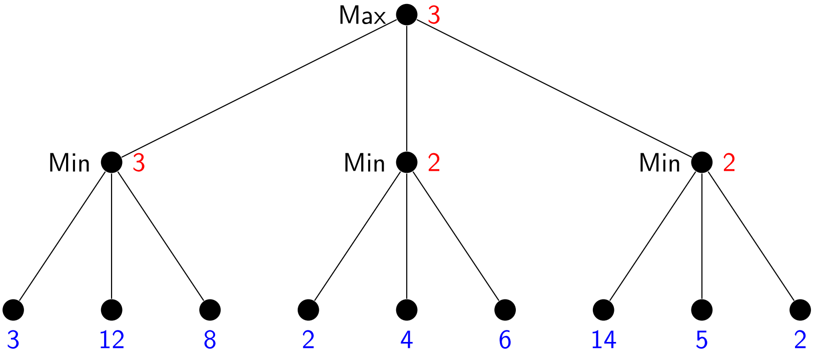
- So which action for Max is returned?

Minimax: Example, Now in Detail



- So which action for Max is returned?

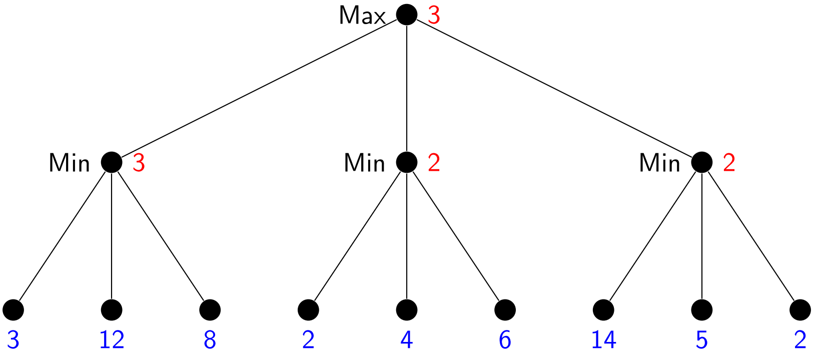
Minimax: Example, Now in Detail



- So which action for Max is returned?

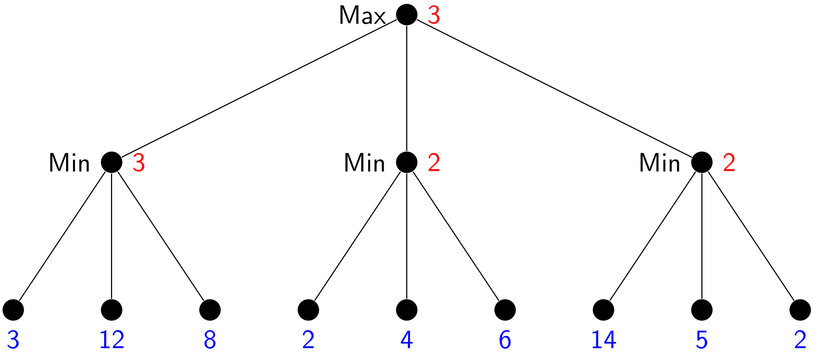
SOME RIGHTS RESERVED

Minimax: Example, Now in Detail



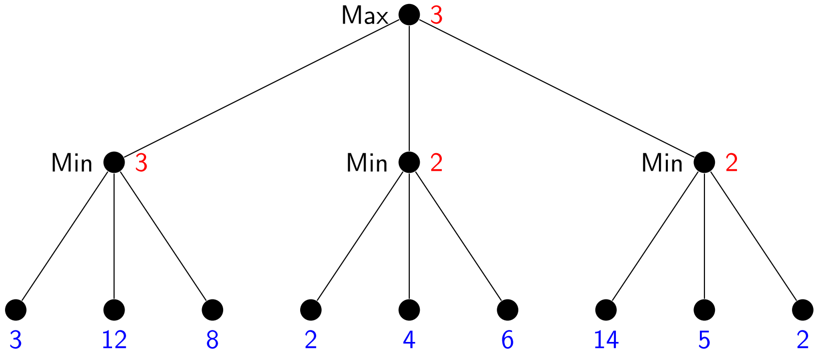
- So which action for Max is returned?

Minimax: Example, Now in Detail



- So which action for Max is returned?
- Leftmost branch.

Minimax: Example, Now in Detail



- So which action for Max is returned?
- Leftmost branch.
- **Note:** The maximal possible pay-off is higher for the rightmost branch, but assuming perfect play of Min, it's better to go left. (Going right would be "relying on your opponent to do something stupid".)

SOME RIGHTS RESERVED

Minimax, Pro and Contra

- **Minimax advantages:**
  - Minimax is the simplest possible (reasonable) search algorithm for games. (If any of you sat down, prior to this lecture, to implement a Tic-Tac-Toe player, chances are you either looked this up on Wikipedia, or invented it in the process.)
  - Returns an optimal action, assuming perfect opponent play.
    - No matter how the opponent plays, the utility of the terminal state reached will be at least the value computed for the root.
    - If the opponent plays perfectly, exactly that value will be reached.
  - There's no need to re-run minimax for every game state: Run it once, offline before the game starts. During the actual game, just follow the branches taken in the tree. Whenever it's your turn, choose an action maximizing the value of the successor states.

SOME RIGHTS RESERVED

Minimax, Pro and Contra

- **Minimax advantages:**
  - Minimax is the simplest possible (reasonable) search algorithm for games. (If any of you sat down, prior to this lecture, to implement a Tic-Tac-Toe player, chances are you either looked this up on Wikipedia, or invented it in the process.)
  - Returns an optimal action, assuming perfect opponent play.
    - No matter how the opponent plays, the utility of the terminal state reached will be at least the value computed for the root.
    - If the opponent plays perfectly, exactly that value will be reached.
  - There's no need to re-run minimax for every game state: Run it once, offline before the game starts. During the actual game, just follow the branches taken in the tree. Whenever it's your turn, choose an action maximizing the value of the successor states.
- **Minimax disadvantages:** It's completely infeasible in practice.
  - When the search tree is too large, we need to limit the search depth and apply an evaluation function to the cut off states.

SOME RIGHTS RESERVED

7.3 Evaluation Functions

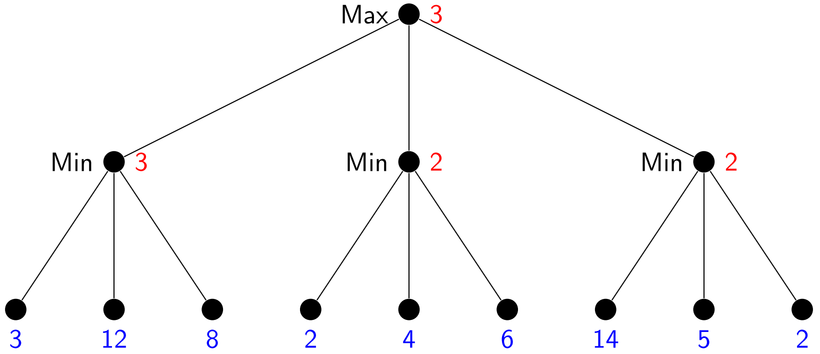
SOME RIGHTS RESERVED

Evaluation Functions for Minimax

- **Problem:** Search tree are too big to search through in minimax.
- **Solution:** We impose a search depth limit (also called horizon) d, and apply an evaluation function to the cut-off states, i.e. states s with $\mathrm{dp}(s) = d$.
- **Definition 3.1.** An evaluation function f maps game states to numbers:
  - $f(s)$ is an estimate of the actual value of s (as would be computed by unlimited-depth minimax for s).
  - If cut-off state is terminal: Just use $\hat{u}$ instead of f.
- Analogy to heuristic functions (cf.): We want f to be both (a) accurate and (b) fast.
- Another analogy: (a) and (b) are in contradiction $\leadsto$ need to trade-off accuracy against overhead.
  - In typical game playing algorithms today, f is inaccurate but very fast. (usually no good methods known for computing accurate f)

SOME RIGHTS RESERVED

Example Revisited: Minimax With Depth Limit $d = 2$



- **Blue numbers:** evaluation function f, applied to the cut-off states at $d = 2$.
- **Red numbers:** utilities of inner node, as computed by minimax using f.

Example Chess

- Evaluation function in chess:
  - **Material**: Pawn 1, Knight 3, Bishop 3, Rook 5, Queen 9.
  - 3 points advantage $\leadsto$ safe win.
  - **Mobility**: How many fields do you control?
  - King safety, Pawn structure, . . .
- Note how simple this is! (probably is not how Kasparov evaluates his positions)

SOME RIGHTS RESERVED

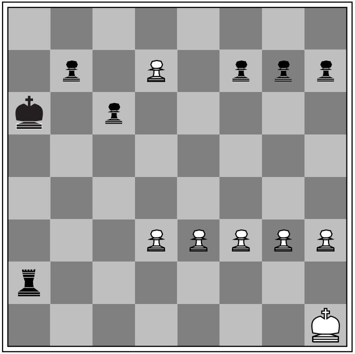
Linear Evaluation Functions

- **Problem:** How to come up with evaluation functions?
- **Definition 3.2.** A common approach is to use a weighted linear function for f, i.e. given a sequence of features $f_i \colon S \to \mathbb{R}$ and a corresponding sequence of weights $w_i \in \mathbb{R}$, f is of the form $f(s) := w_1 \cdot f_1(s) + w_2 \cdot f_2(s) + \cdots + w_n \cdot f_n(s)$
- **Problem:** How to obtain these weighted linear functions?
  - Weights w_i can be learned automatically. (learning agent)
  - The features f_i, however, have to be designed by human experts.
- **Note:** Very fast, very simplistic.
- **Observation:** Can be computed incrementally: In transition $\langle a,s,s' \rangle$, adapt $f(s)$ to $f(s')$ by considering only those features whose values have changed.

SOME RIGHTS RESERVED

The Horizon Problem

- **Problem:** Critical aspects of the game can be cut off by the horizon. We call this the horizon problem.
- **Example 3.3.**



Black to move

- Who's gonna win here?
  - White wins (pawn cannot be prevented from becoming a queen.)
  - Black has a $+4$ advantage in material, so if we cut-off here then our evaluation function will say "100%, black wins".
  - The loss for black is "beyond our horizon" unless we search extremely deeply: black can hold off the end by repeatedly giving check to white's king.

SOME RIGHTS RESERVED

So, How Deeply to Search?

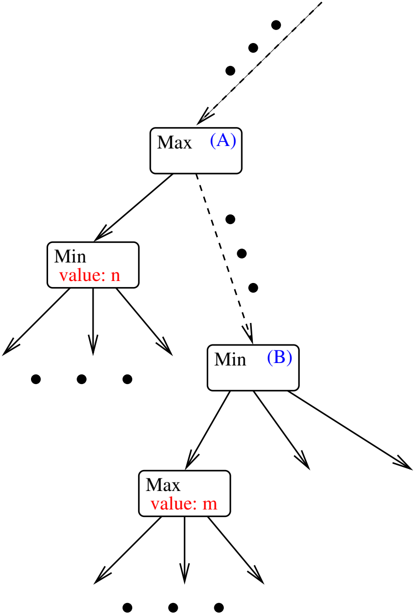
- **Goal:** In given time, search as deeply as possible.
- **Problem:** Very difficult to predict search running time. (need an anytime algorithm)
- **Solution:** Iterative deepening search.
  - Search with depth limit $d = 1, 2, 3, \ldots$
  - When time is up: return result of deepest completed search.
- **Definition 3.4 (Better Solution).** The quiescent search algorithm uses a dynamically adapted search depth *d*: It searches more deeply in unquiet positions, where value of evaluation function changes a lot in neighboring states.
- **Example 3.5.** In quiescent search for chess:
  - piece exchange situations ("you take mine, I take yours") are very unquiet
  - $\rightsquigarrow$ Keep searching until the end of the piece exchange is reached.

SOME RIGHTS RESERVED

7.4 Alpha-Beta Search

SOME RIGHTS RESERVED

When We Already Know We Can Do Better Than This

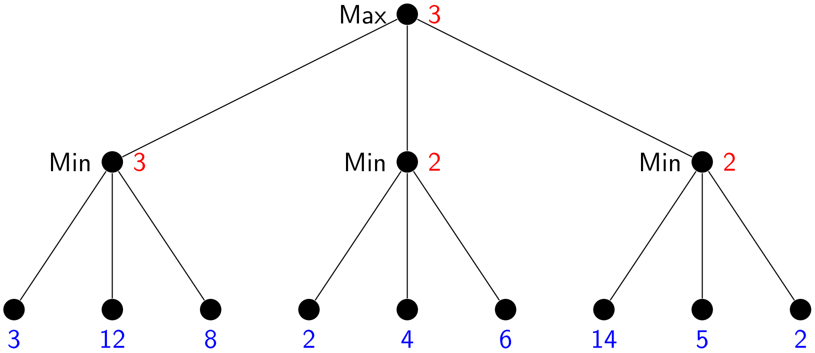


- Say $n > m$.
- By choosing to go to the left in search node (A), Max already can get utility of at least n in this part of the game.
- So, if "later on" (further down in the same subtree), in search node (B) we already know that Min can force Max to get value $m < n$.
- Then Max will play differently in (A) so we will never actually get to (B).

SOME RIGHTS RESERVED

Alpha Pruning: Basic Idea

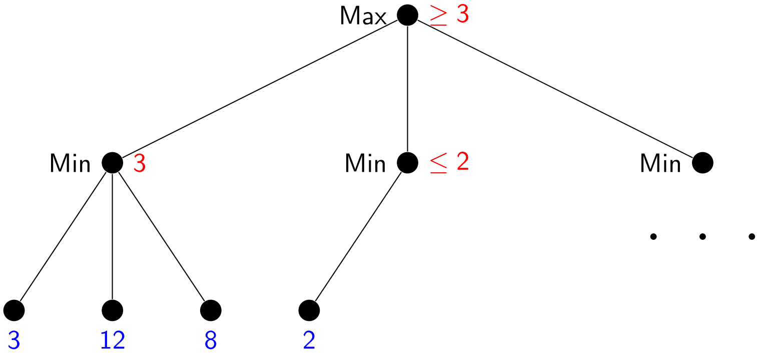
- **Question:** Can we save some work here?



SOME RIGHTS RESERVED

Alpha Pruning: Basic Idea (Continued)

- **Answer:** Yes! We already know at this point that the middle action won't be taken by Max.



- **Idea:** We can use this to prune the search tree ⇝ better algorithm

SOME RIGHTS RESERVED

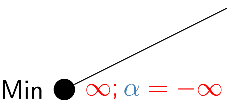
Alpha Pruning

- **Definition 4.1.** For each node n in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to n.

SOME RIGHTS RESERVED

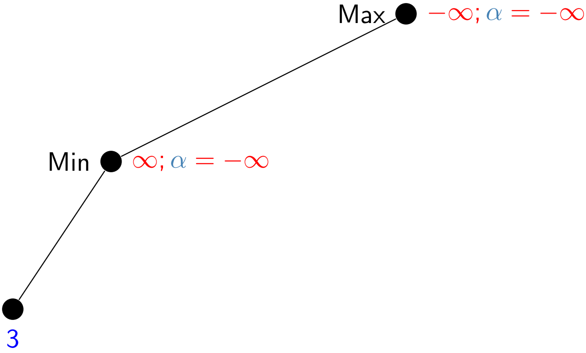
Alpha Pruning

- **Definition 4.3.** For each node n in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to n.
- **Example 4.4 (Computing alpha values).**



Alpha Pruning

- **Definition 4.5.** For each node *n* in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to *n*.
- **Example 4.6 (Computing alpha values).**

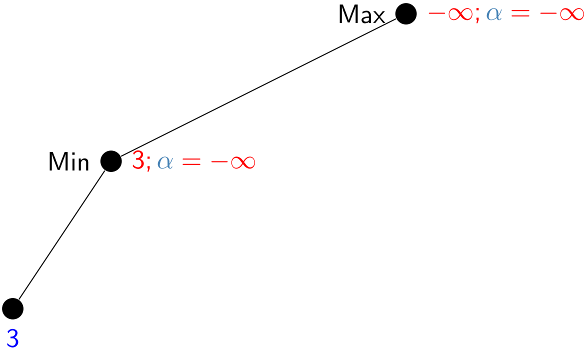


- **How to use α?:** In a Min-node *n*, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering *n*. (pruning out its remaining successors)

SOME RIGHTS RESERVED

Alpha Pruning

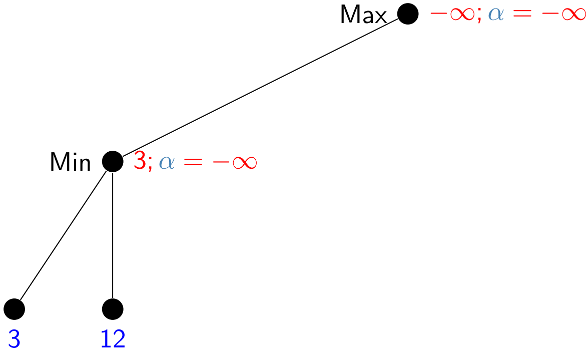
- **Definition 4.7.** For each node *n* in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to *n*.
- **Example 4.8 (Computing alpha values).**



- **How to use α?:** In a Min-node *n*, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering *n*. (pruning out its remaining successors)

Alpha Pruning

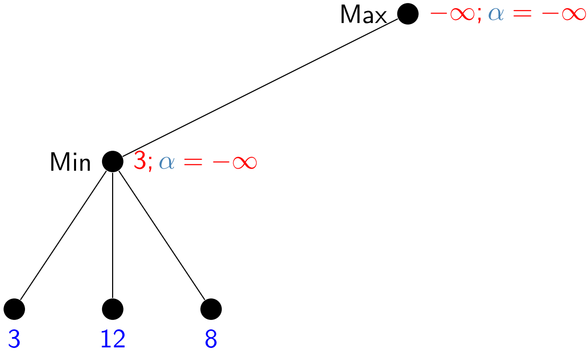
- **Definition 4.9.** For each node *n* in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to *n*.
- **Example 4.10 (Computing alpha values).**



- **How to use α?:** In a Min-node *n*, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering *n*. (pruning out its remaining successors)

Alpha Pruning

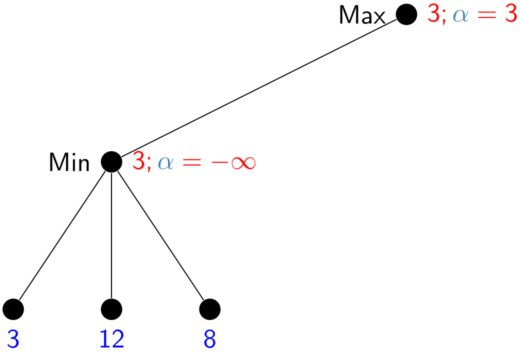
- **Definition 4.11.** For each node n in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to n.
- **Example 4.12 (Computing alpha values).**



- **How to use α?:** In a Min-node n, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering n. (pruning out its remaining successors)

Alpha Pruning

- **Definition 4.13.** For each node n in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to n.
- **Example 4.14 (Computing alpha values).**

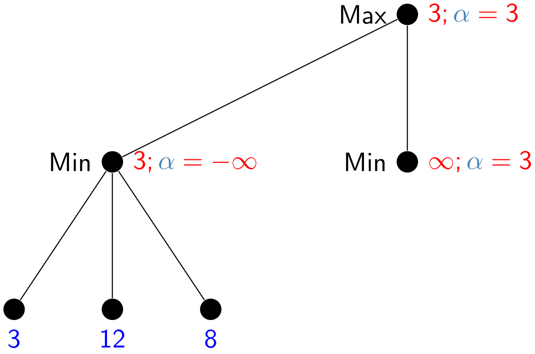


- **How to use α?:** In a Min-node n, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering n. (pruning out its remaining successors)

SOME RIGHTS RESERVED

Alpha Pruning

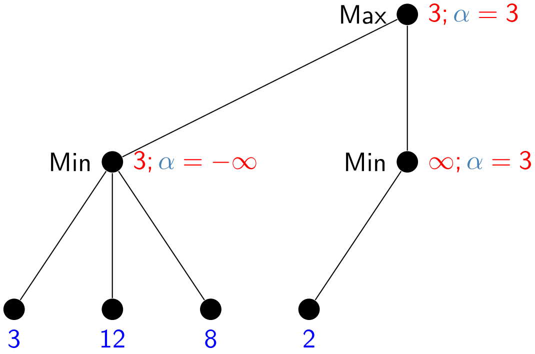
- **Definition 4.15.** For each node n in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to n.
- **Example 4.16 (Computing alpha values).**



- **How to use α?:** In a Min-node n, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering n. (pruning out its remaining successors)

Alpha Pruning

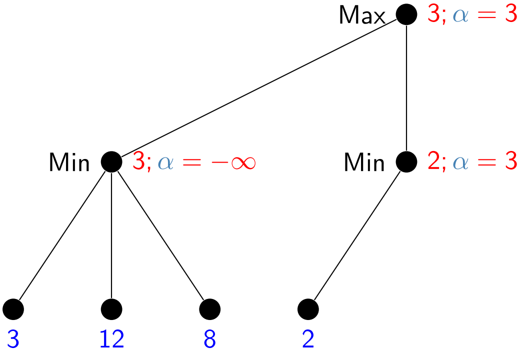
- **Definition 4.17.** For each node n in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to n.
- **Example 4.18 (Computing alpha values).**



- **How to use α?:** In a Min-node n, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering n. (pruning out its remaining successors)

Alpha Pruning

- **Definition 4.19.** For each node n in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to n.
- **Example 4.20 (Computing alpha values).**

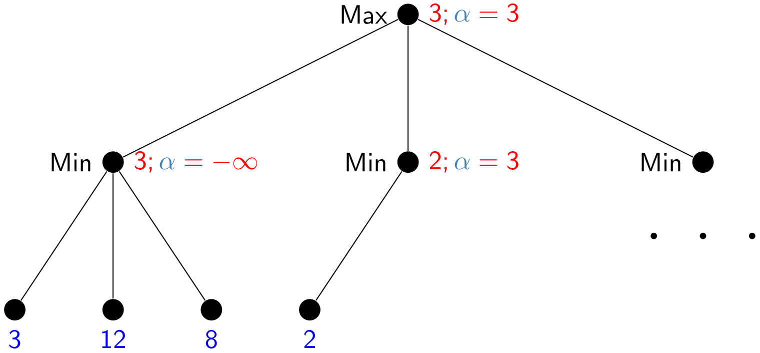


- **How to use α?:** In a Min-node n, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering n. (pruning out its remaining successors)

SOME RIGHTS RESERVED

Alpha Pruning

- **Definition 4.21.** For each node *n* in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to *n*.
- **Example 4.22 (Computing alpha values).**



- **How to use α?:** In a Min-node *n*, if $\hat{u}(n') \leq \alpha(n)$ for one of the successors, then stop considering *n*. (pruning out its remaining successors)

Alpha-Beta Pruning

- **Recall:**
  - **What is α:** For each search node *n*, the highest Max-node utility that search has encountered on its path from the root to *n*.
  - **How to use α:** In a Min-node *n*, if one of the successors already has utility $\leq \alpha(n)$, then stop considering *n*. (Pruning out its remaining successors)
- **Idea:** We can use a dual method for Min!
- **Definition 4.23.** For each node *n* in a minimax search tree, the beta value $\beta(n)$ is the highest Min-node utility that search has encountered on its path from the root to *n*.
- **How to use β:** In a Max-node *n*, if one of the successors already has utility $\geq \beta(n)$, then stop considering *n*. (pruning out its remaining successors)
- ...and of course we can use α and β together! $\rightsquigarrow$ alphabeta-pruning

SOME RIGHTS RESERVED

Alpha-Beta Search: Pseudocode

- **Definition 4.24.** The alphabeta search algorithm is given by the following pseudocode

```
function Alpha−Beta−Search (s) returns an action
    v := Max−Value(s, −∞, +∞)
    return an action yielding value v in the previous function call

function Max−Value(s, α, β) returns a utility value
    if Terminal−Test(s) then return u(s)
    v:= −∞
    for each a ∈ Actions(s) do
        v := max(v,Min−Value(ChildState(s,a), α, β))
        α := max(α, v)
        if v ≥ β then return v /∗ Here: v ≥ β ⇔ α ≥ β ∗/
    return v

function Min−Value(s, α, β) returns a utility value
    if Terminal−Test(s) then return u(s)
    v := +∞
    for each a ∈ Actions(s) do
        v := min(v,Max−Value(ChildState(s,a), α, β))
        β := min(β, v)
        if v ≤ α then return v /∗ Here: v ≤ α ⇔ α ≥ β ∗/
    return v
```

$\widehat{=}$ Minimax (slide 215) + α/β book-keeping and pruning.

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

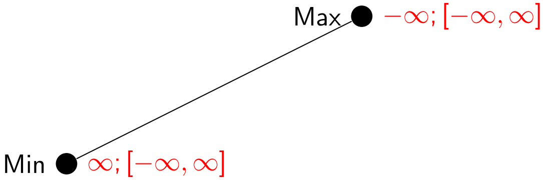
- **Notation:** $v; [\alpha, \beta]$

Max ● $-\infty; [-\infty, \infty]$

- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

Alpha-Beta Search: Example

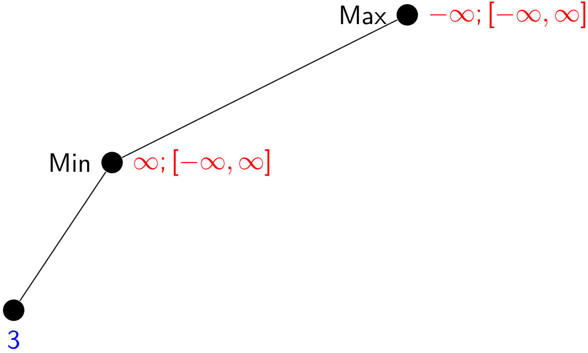
- **Notation:** $v; [\alpha, \beta]$



- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

Alpha-Beta Search: Example

- **Notation:** $v; [\alpha, \beta]$

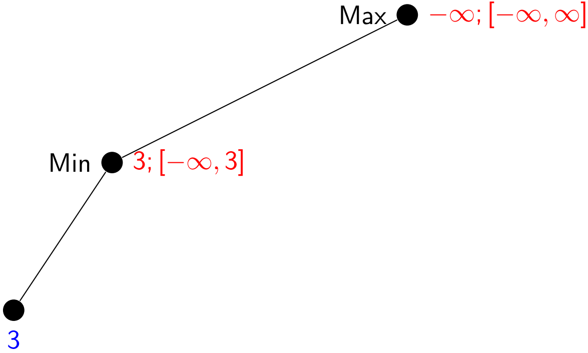


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

- **Notation:** $v; [\alpha, \beta]$

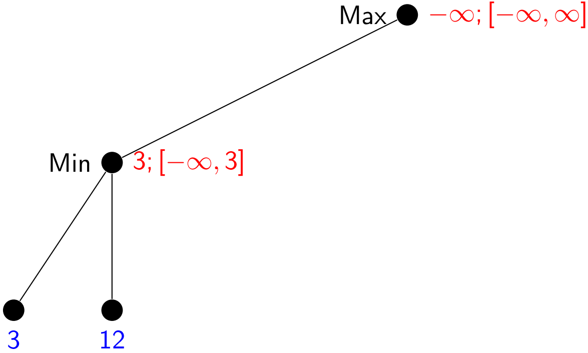


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

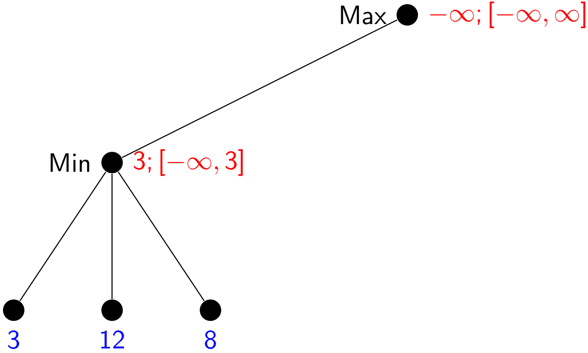
- **Notation:** $v; [\alpha, \beta]$



- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

Alpha-Beta Search: Example

- **Notation:** $v; [\alpha, \beta]$

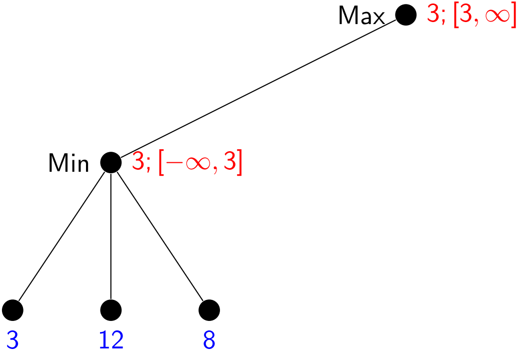


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

- **Notation:** $v; [\alpha, \beta]$

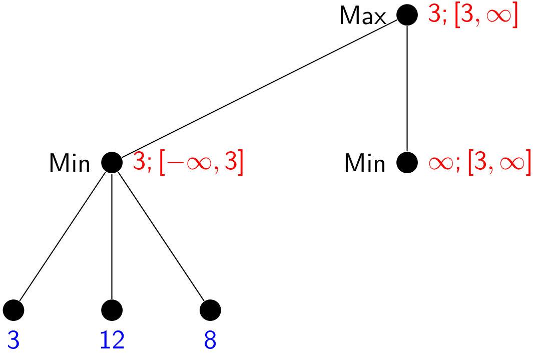


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

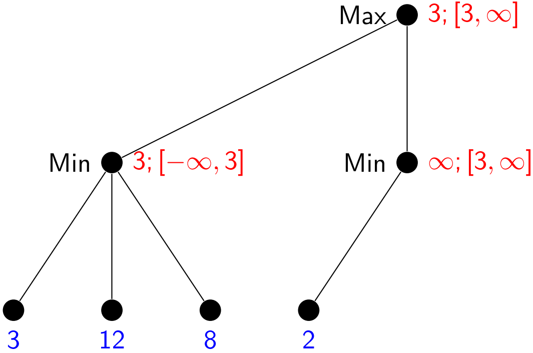
- **Notation:** $v; [\alpha, \beta]$



- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

Alpha-Beta Search: Example

- **Notation:** $v; [\alpha, \beta]$

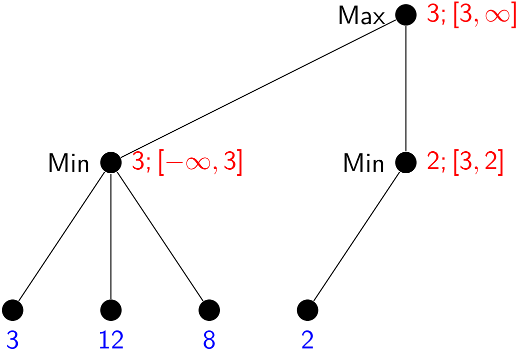


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

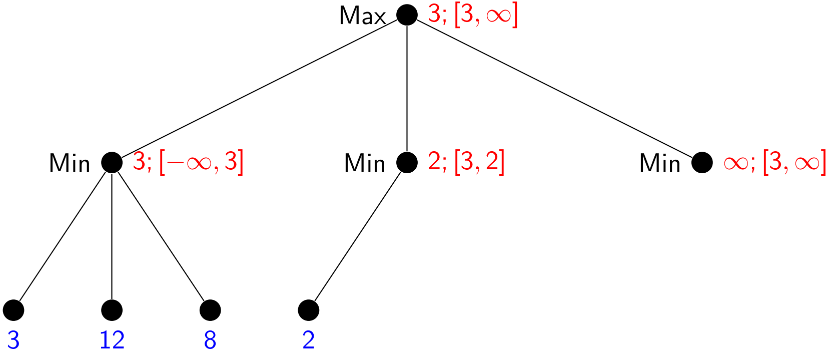
- **Notation:** $v; [\alpha, \beta]$



- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

Alpha-Beta Search: Example

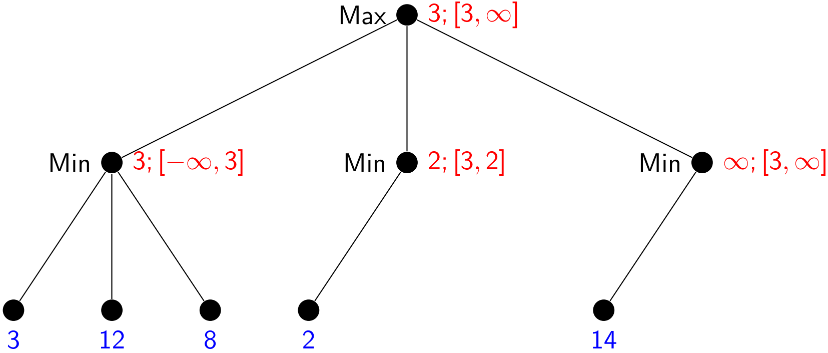
- **Notation:** $v; [\alpha, \beta]$



- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

Alpha-Beta Search: Example

- **Notation:** $v; [\alpha, \beta]$

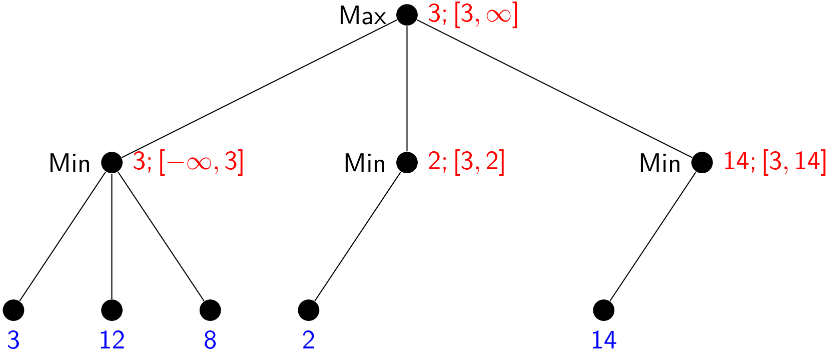


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

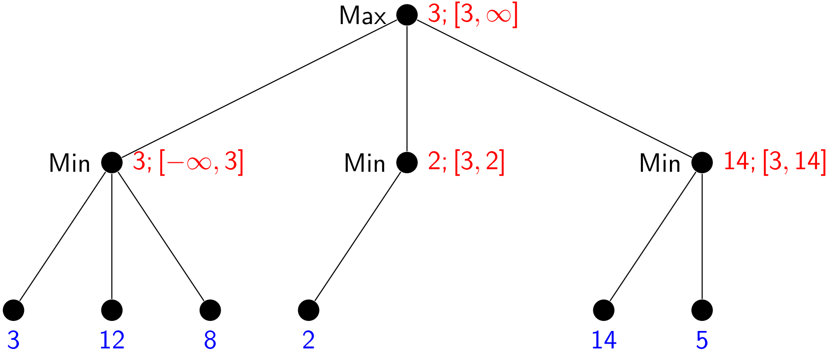
- **Notation:** $v; [\alpha, \beta]$



- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

Alpha-Beta Search: Example

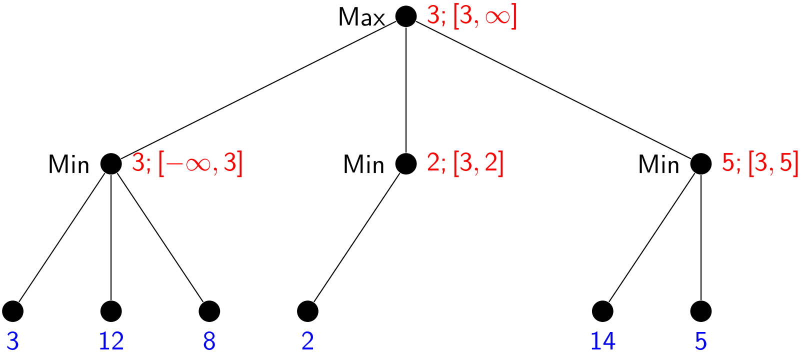
- **Notation:** $v; [\alpha, \beta]$



- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

Alpha-Beta Search: Example

- **Notation:** v; $[\alpha, \beta]$

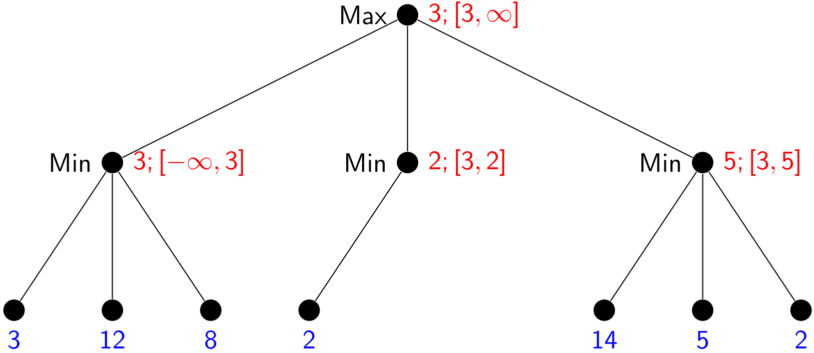


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

- **Notation:** v; $[\alpha, \beta]$

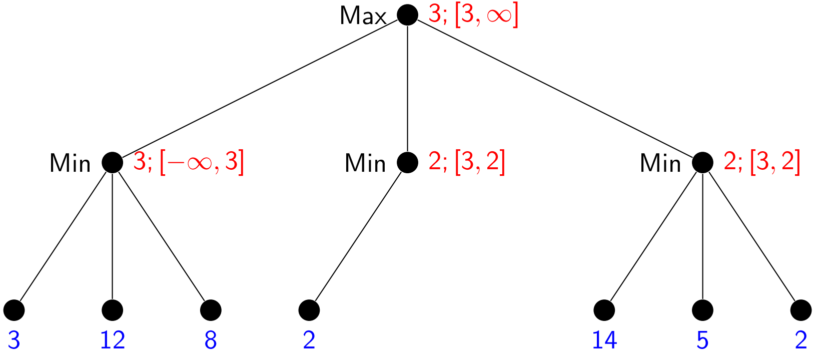


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

Alpha-Beta Search: Example

- **Notation:** $v; [\alpha, \beta]$

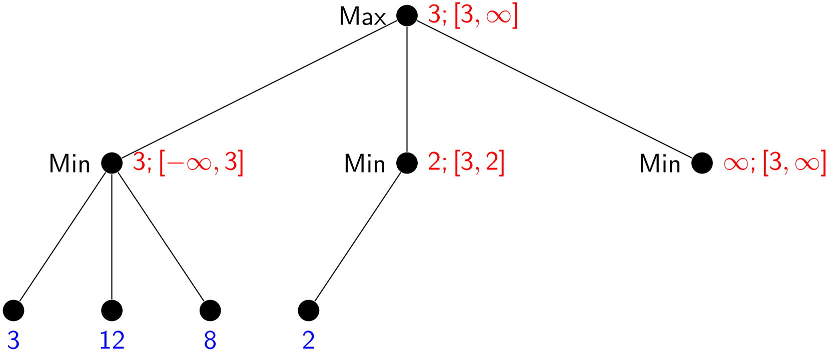


- **Note:** We could have saved work by choosing the opposite order for the successors of the rightmost Min-node.
  Choosing the best moves (for each of Max and Min) first yields more pruning!

SOME RIGHTS RESERVED

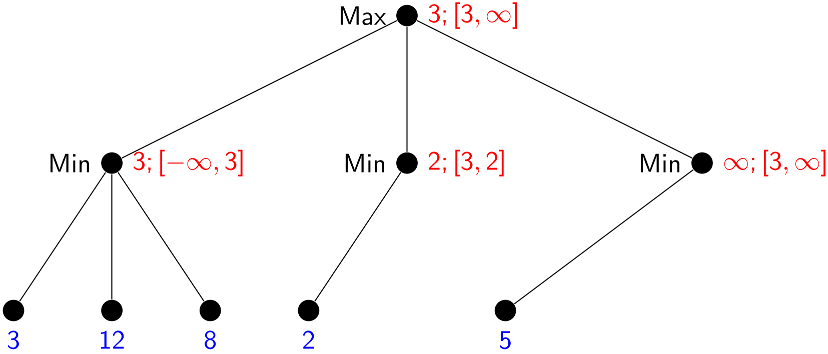
Alpha-Beta Search: Modified Example

- **Showing off some actual** β **pruning:**



Alpha-Beta Search: Modified Example

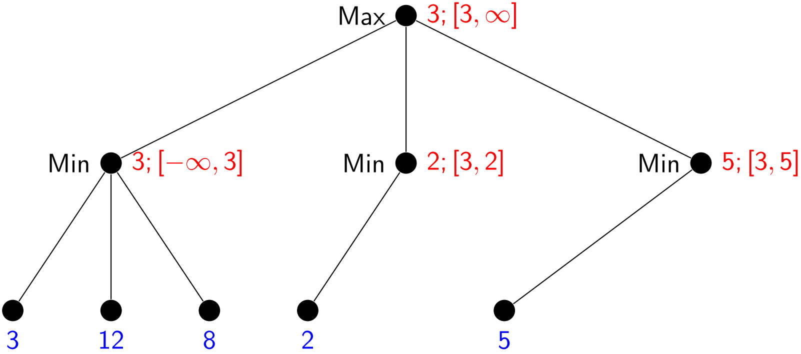
- **Showing off some actual β pruning:**



SOME RIGHTS RESERVED

Alpha-Beta Search: Modified Example

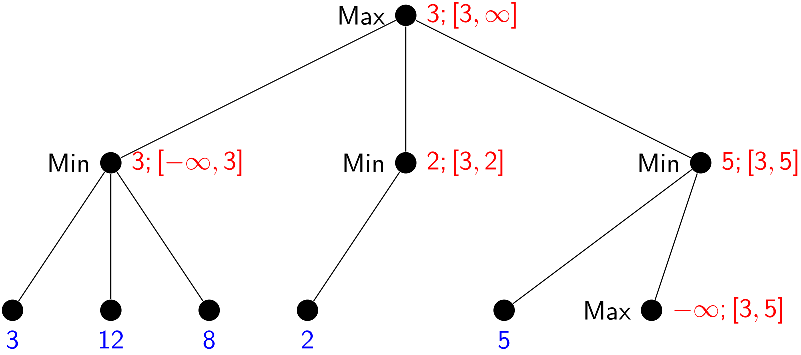
- **Showing off some actual β pruning:**



SOME RIGHTS RESERVED

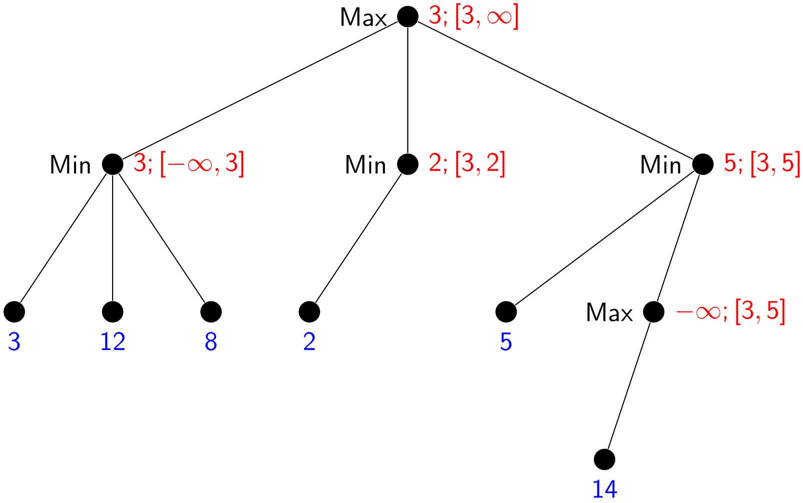
Alpha-Beta Search: Modified Example

- **Showing off some actual β pruning:**



Alpha-Beta Search: Modified Example

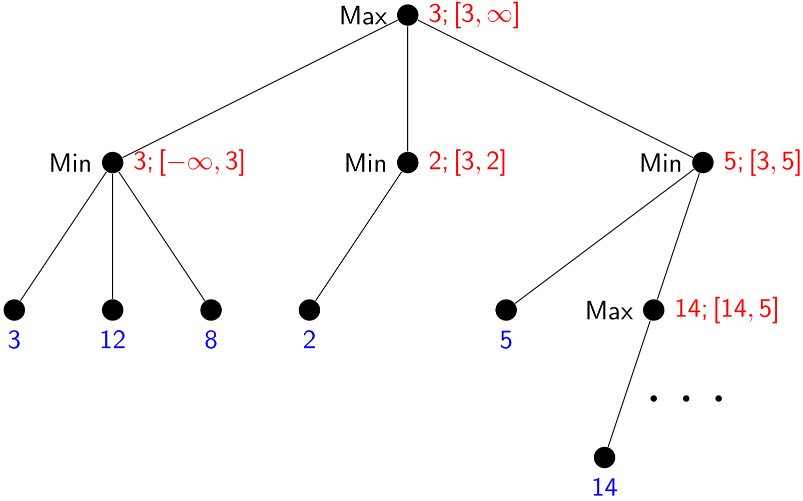
- **Showing off some actual β pruning:**



SOME RIGHTS RESERVED

Alpha-Beta Search: Modified Example

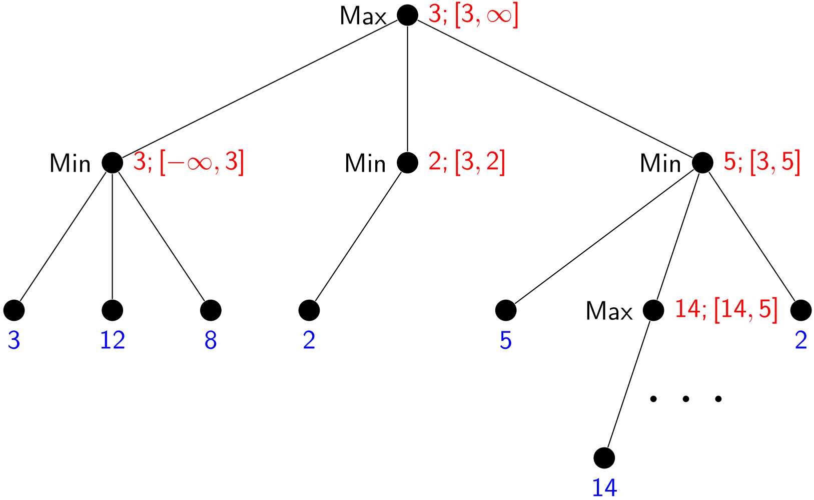
- **Showing off some actual β pruning:**



SOME RIGHTS RESERVED

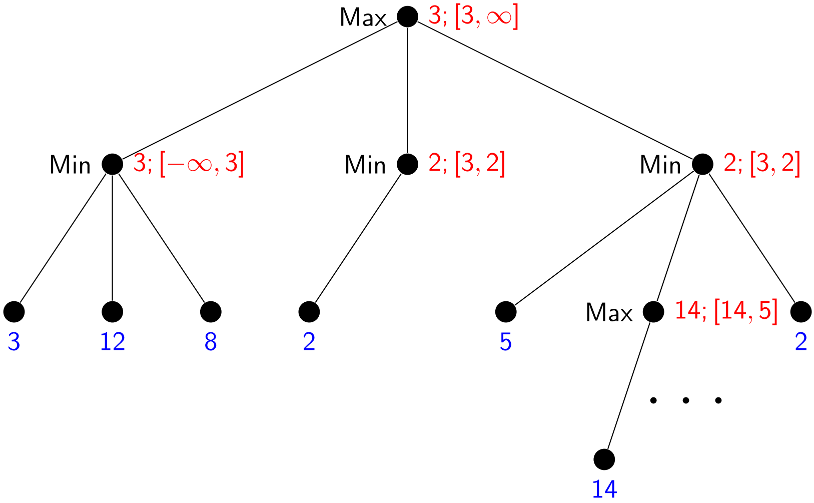
Alpha-Beta Search: Modified Example

- **Showing off some actual β pruning:**



Alpha-Beta Search: Modified Example

- **Showing off some actual β pruning:**



How Much Pruning Do We Get?

- Choosing the best moves first yields most pruning in alphabeta search.
  - The maximizing moves for Max, the minimizing moves for Min.
- **Observation:** Assuming game tree with branching factor b and depth limit d:
  - Minimax would have to search b^d nodes.
  - **Best case:** If we always choose the best moves first, then the search tree is reduced to $b^{\frac{d}{2}}$ nodes!
  - Practice: It is often possible to get very close to the best case by simple move-ordering methods.
- **Example 4.25 (Chess).**
  - Move ordering: Try captures first, then threats, then forward moves, then backward moves.
  - From 35^d to $35^{\frac{d}{2}}$. E.g., if we have the time to search a billion (10^9) nodes, then minimax looks ahead $d = 6$ moves, i.e., 3 rounds (white-black) of the game. Alpha-beta search looks ahead 6 rounds.

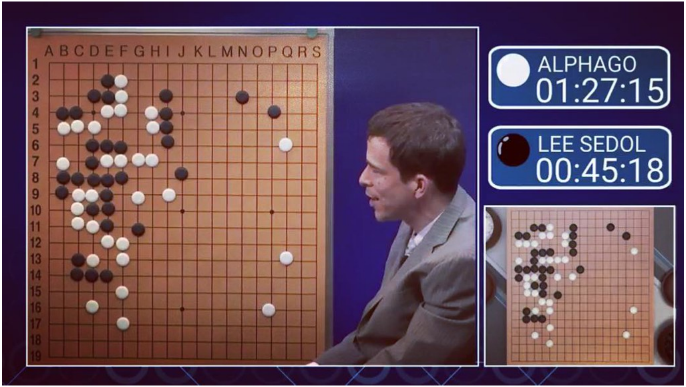
SOME RIGHTS RESERVED

7.5 Monte-Carlo Tree Search (MCTS)

SOME RIGHTS RESERVED

And now . . .

- **AlphaGo = Monte Carlo tree search (AI-1) + neural networks (AI-2)**



CC-BY-SA: Buster Benson@ `https://www.flickr.com/photos/erikbenson/25717574115`

SOME RIGHTS RESERVED

Monte-Carlo Tree Search: Basic Ideas

- **Observation:** We do not always have good evaluation functions.
- **Definition 5.1.** For Monte Carlo sampling we evaluate actions through sampling.
  - When deciding which action to take on game state s:

```
while time not up do
  select action $a$ applicable to $s$
  run a random sample from $a$ until terminal state $t$
return an $a$ for $s$ with maximal average $u(t)$
```

- **Definition 5.2.** For the Monte Carlo tree search algorithm (MCTS) we maintain a search tree T, the MCTS tree.

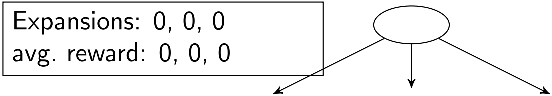
```
while time not up do
  apply actions within $T$ to select a leaf state $s'$
  select action $a'$ applicable to $s'$, run random sample from $a'$
  add $s'$ to $T$, update averages etc.
return an $a$ for $s$ with maximal average $u(t)$
When executing $a$, keep the part of $T$ below $a$.
```

- Compared to alphabeta search: no exhaustive enumeration.
  - **Pro**: running time & memory.
  - **Contra**: need good guidance how to select and sample.

SOME RIGHTS RESERVED

Monte-Carlo Sampling: Illustration of Sampling

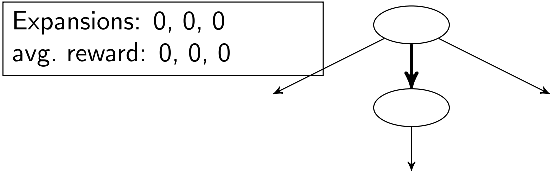
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.3 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)


Expansions: 0, 0, 0
avg. reward: 0, 0, 0

SOME RIGHTS RESERVED

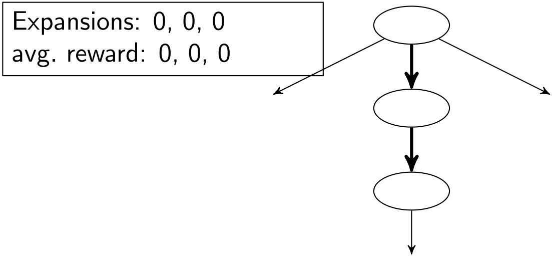
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.4 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)


Expansions: 0, 0, 0
avg. reward: 0, 0, 0

Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.5 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



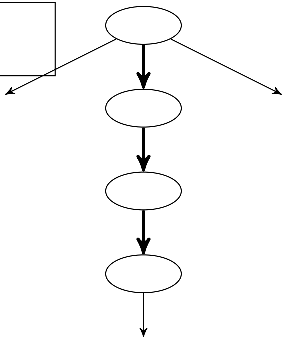
Expansions: 0, 0, 0
avg. reward: 0, 0, 0

SOME RIGHTS RESERVED

Monte-Carlo Sampling: Illustration of Sampling

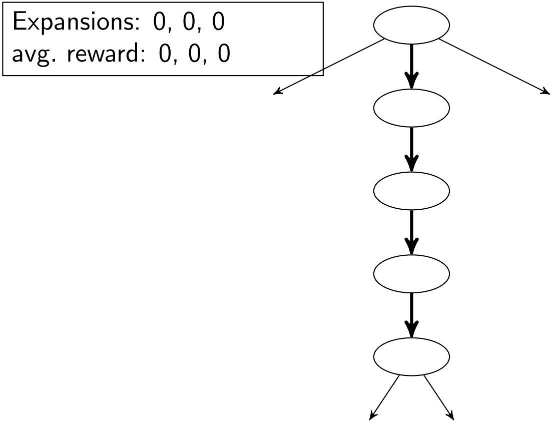
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.6 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

Expansions: 0, 0, 0
avg. reward: 0, 0, 0



Monte-Carlo Sampling: Illustration of Sampling

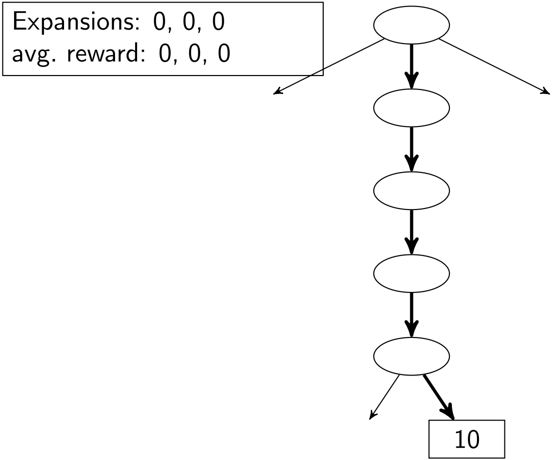
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.7 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 0, 0, 0
avg. reward: 0, 0, 0

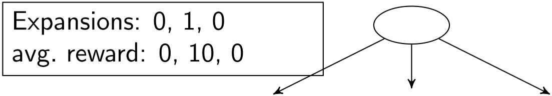
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.8 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



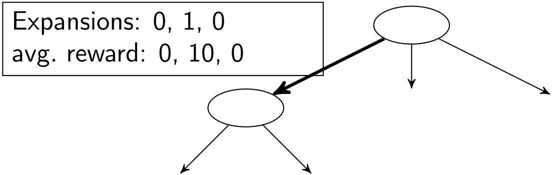
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.9 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.10 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

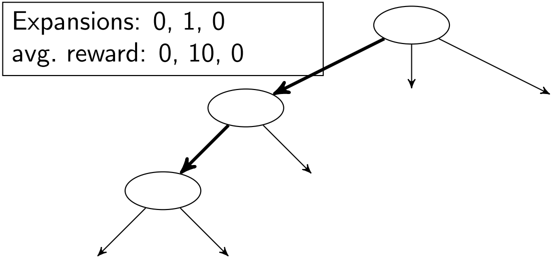


Expansions: 0, 1, 0

avg. reward: 0, 10, 0

Monte-Carlo Sampling: Illustration of Sampling

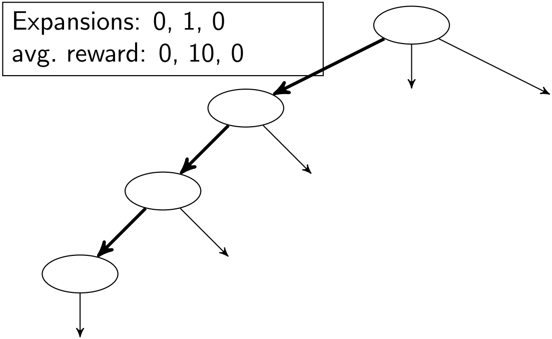
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.11 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 0, 1, 0
avg. reward: 0, 10, 0

Monte-Carlo Sampling: Illustration of Sampling

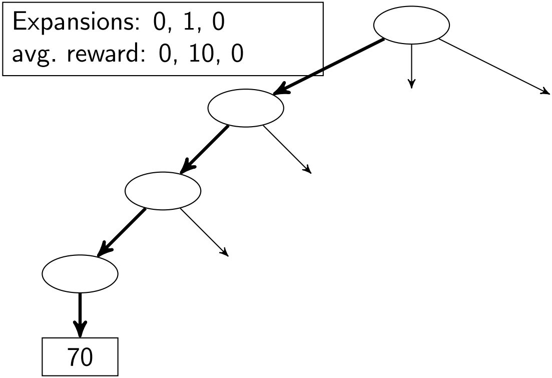
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.12 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 0, 1, 0
avg. reward: 0, 10, 0

Monte-Carlo Sampling: Illustration of Sampling

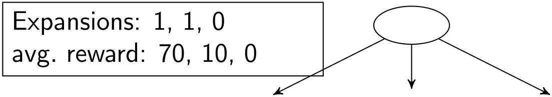
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.13 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 0, 1, 0
avg. reward: 0, 10, 0

Monte-Carlo Sampling: Illustration of Sampling

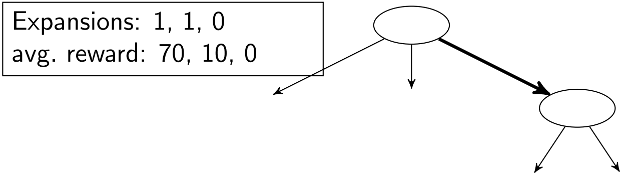
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.14 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 1, 1, 0
avg. reward: 70, 10, 0

Monte-Carlo Sampling: Illustration of Sampling

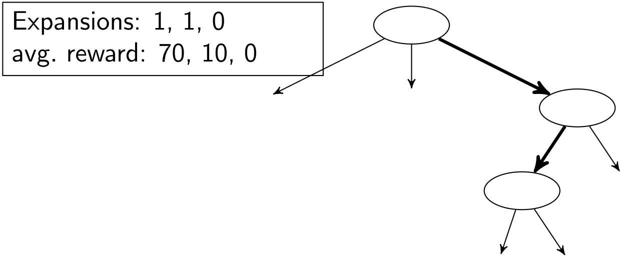
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.15 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 1, 1, 0
avg. reward: 70, 10, 0

Monte-Carlo Sampling: Illustration of Sampling

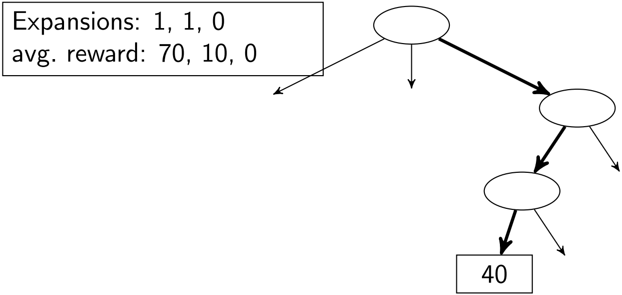
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.16 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 1, 1, 0
avg. reward: 70, 10, 0

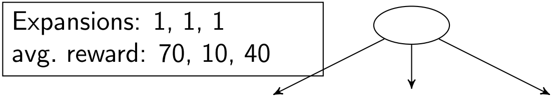
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.17 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.18 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

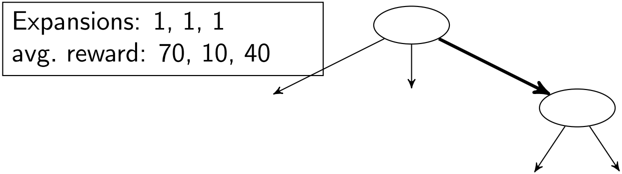


Expansions: 1, 1, 1
avg. reward: 70, 10, 40

SOME RIGHTS RESERVED

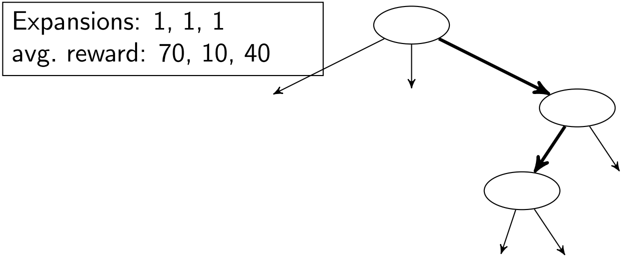
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.19 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)


Expansions: 1, 1, 1
avg. reward: 70, 10, 40

Monte-Carlo Sampling: Illustration of Sampling

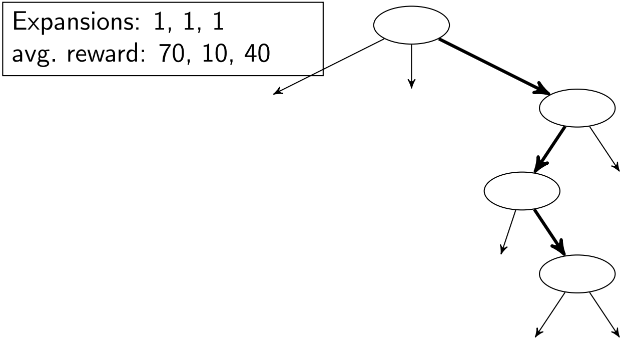
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.20 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 1, 1, 1
avg. reward: 70, 10, 40

Monte-Carlo Sampling: Illustration of Sampling

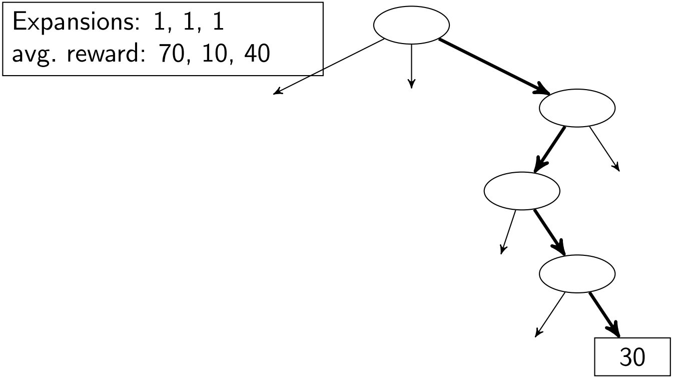
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.21 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 1, 1, 1
avg. reward: 70, 10, 40

Monte-Carlo Sampling: Illustration of Sampling

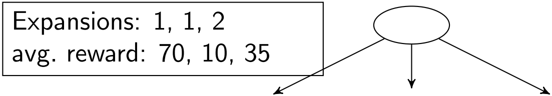
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.22 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 1, 1, 1
avg. reward: 70, 10, 40

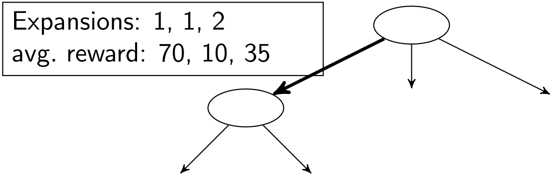
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.23 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



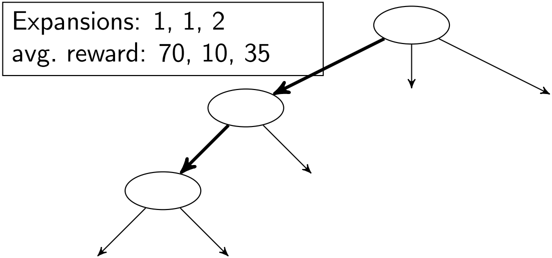
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.24 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



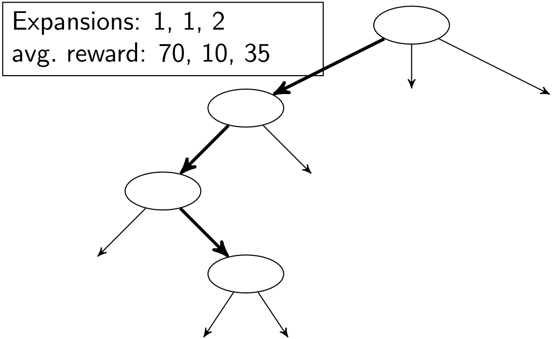
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.25 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.26 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



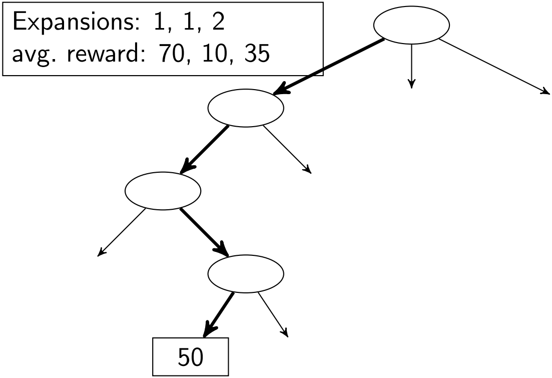
Expansions: 1, 1, 2

avg. reward: 70, 10, 35

SOME RIGHTS RESERVED

Monte-Carlo Sampling: Illustration of Sampling

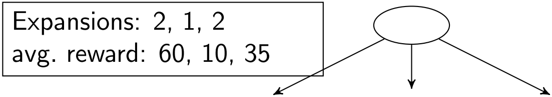
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.27 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)


Expansions: 1, 1, 2
avg. reward: 70, 10, 35

SOME RIGHTS RESERVED

Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.28 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

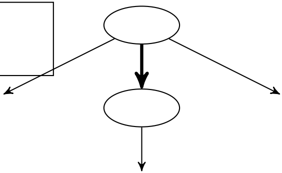


Expansions: 2, 1, 2
avg. reward: 60, 10, 35

Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.29 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

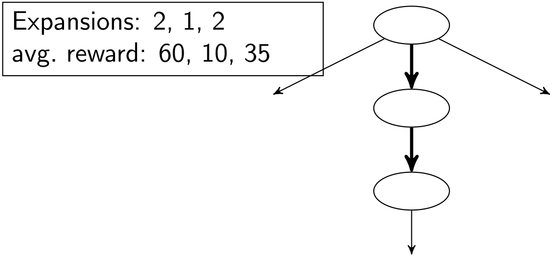
Expansions: 2, 1, 2
avg. reward: 60, 10, 35



SOME RIGHTS RESERVED

Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.30 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

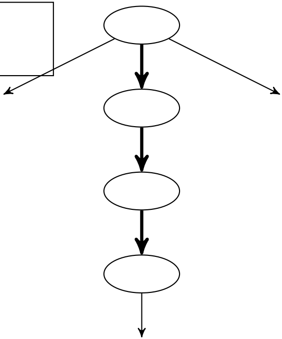


Expansions: 2, 1, 2
avg. reward: 60, 10, 35

Monte-Carlo Sampling: Illustration of Sampling

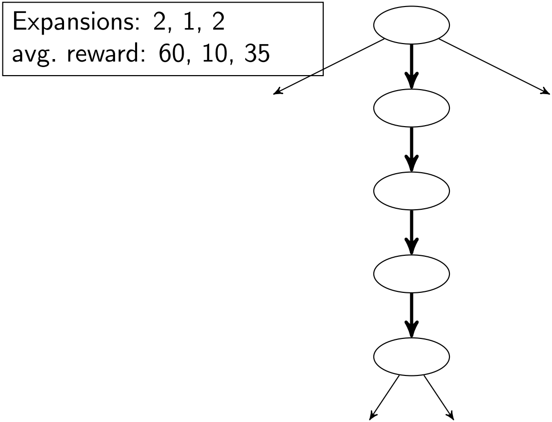
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.31 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

Expansions: 2, 1, 2
avg. reward: 60, 10, 35



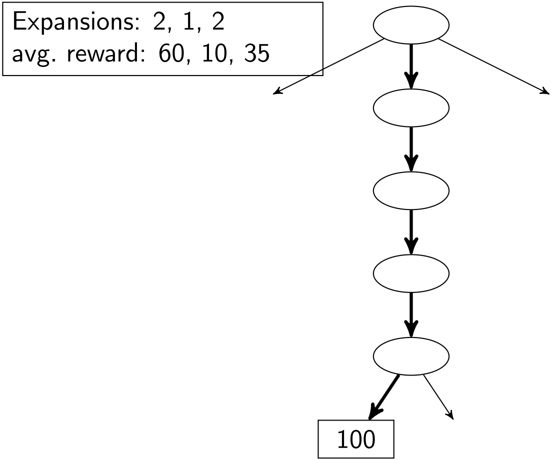
Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.32 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)


Expansions: 2, 1, 2
avg. reward: 60, 10, 35

Monte-Carlo Sampling: Illustration of Sampling

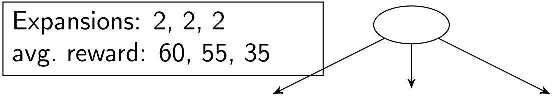
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.33 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 2, 1, 2
avg. reward: 60, 10, 35

Monte-Carlo Sampling: Illustration of Sampling

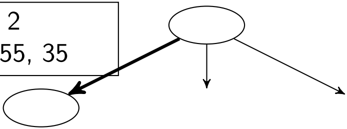
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.34 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 2, 2, 2
avg. reward: 60, 55, 35

Monte-Carlo Sampling: Illustration of Sampling

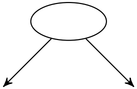
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.35 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 2, 2, 2
avg. reward: **60**, 55, 35

Monte-Carlo Sampling: Illustration of Sampling

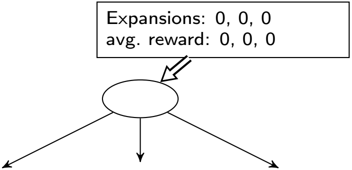
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.36 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Expansions: 0, 0
avg. reward: 0, 0

Monte-Carlo Tree Search: Building the Tree

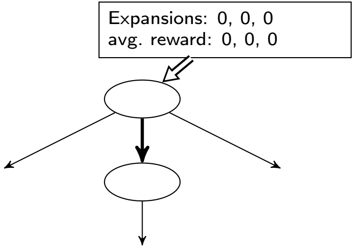
- **Idea:** We can save work by building the tree as we go along.
- **Example 5.37 (Redoing the previous example).**



Expansions: 0, 0, 0
avg. reward: 0, 0, 0

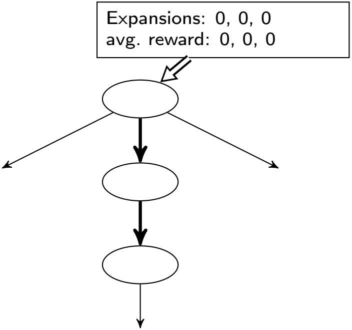
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.38 (Redoing the previous example).**



Monte-Carlo Tree Search: Building the Tree

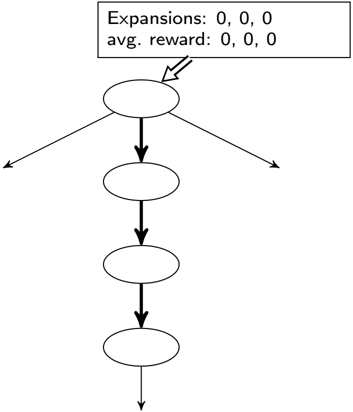
- **Idea:** We can save work by building the tree as we go along.
- **Example 5.39 (Redoing the previous example).**



Expansions: 0, 0, 0
avg. reward: 0, 0, 0

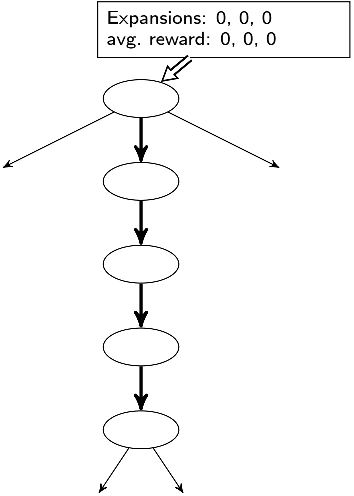
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.40 (Redoing the previous example).**



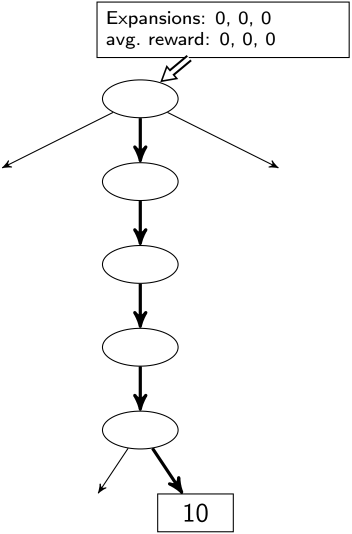
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.41 (Redoing the previous example).**


Expansions: 0, 0, 0
avg. reward: 0, 0, 0

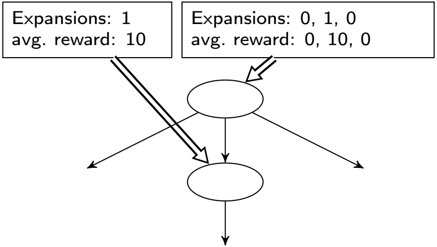
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.42 (Redoing the previous example).**



Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.43 (Redoing the previous example).**

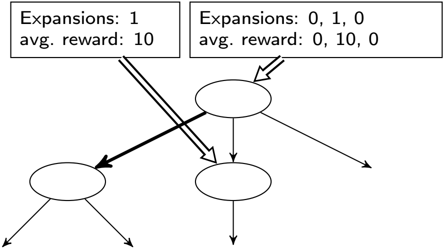


Expansions: 1
avg. reward: 10

Expansions: 0, 1, 0
avg. reward: 0, 10, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.44 (Redoing the previous example).**

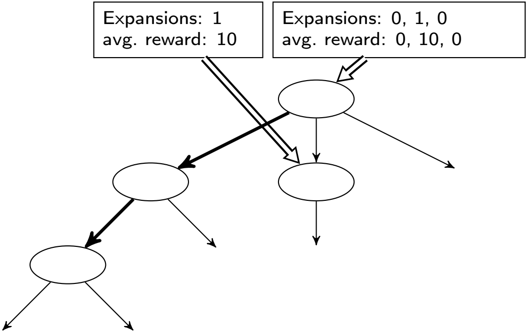


Expansions: 1
avg. reward: 10

Expansions: 0, 1, 0
avg. reward: 0, 10, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.45 (Redoing the previous example).**

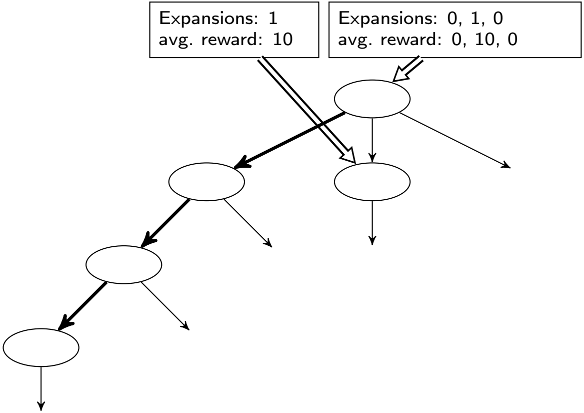


Expansions: 1
avg. reward: 10

Expansions: 0, 1, 0
avg. reward: 0, 10, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.46 (Redoing the previous example).**

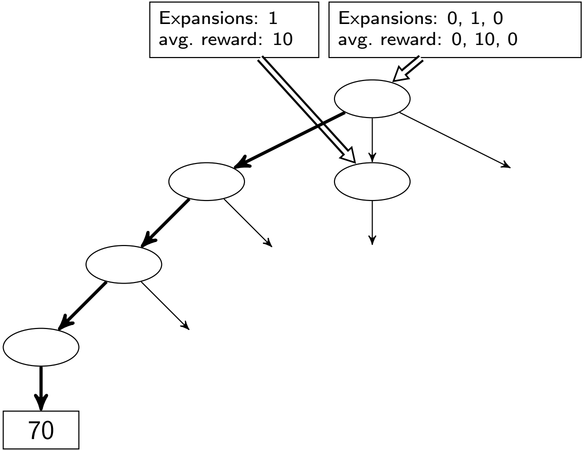


Expansions: 1
avg. reward: 10

Expansions: 0, 1, 0
avg. reward: 0, 10, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.47 (Redoing the previous example).**



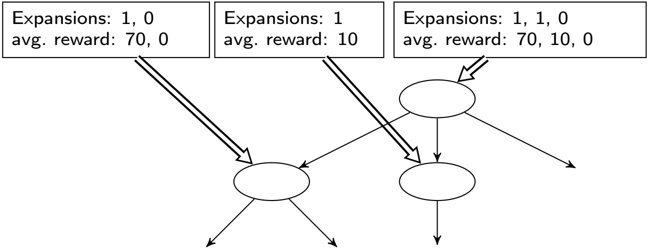
Expansions: 1
avg. reward: 10

Expansions: 0, 1, 0
avg. reward: 0, 10, 0

SOME RIGHTS RESERVED

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.48 (Redoing the previous example).**



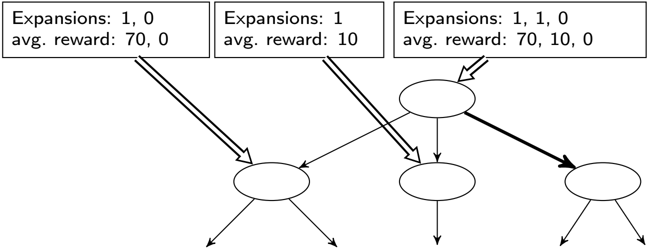
Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

Expansions: 1, 1, 0
avg. reward: 70, 10, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.49 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

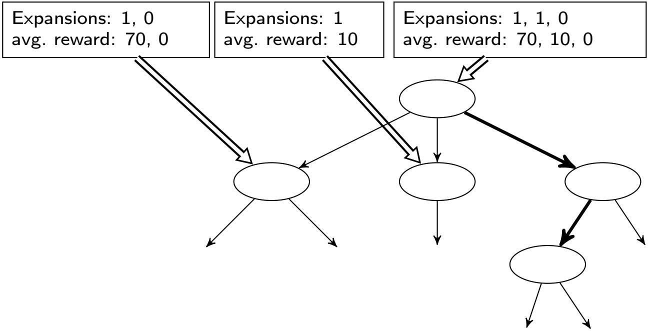
Expansions: 1
avg. reward: 10

Expansions: 1, 1, 0
avg. reward: 70, 10, 0

SOME RIGHTS RESERVED

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.50 (Redoing the previous example).**



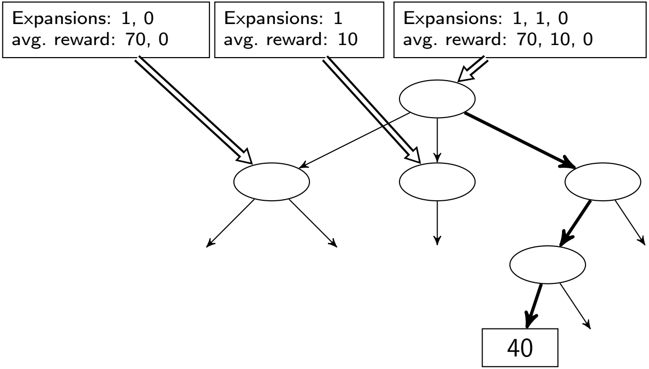
Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

Expansions: 1, 1, 0
avg. reward: 70, 10, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.51 (Redoing the previous example).**



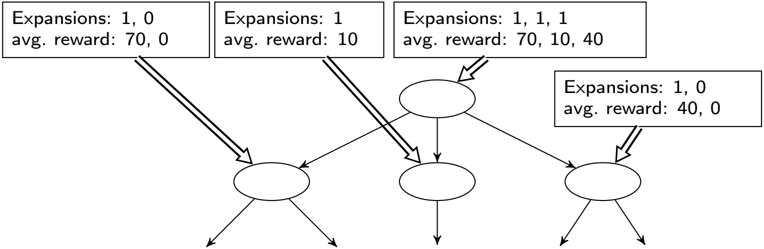
Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

Expansions: 1, 1, 0
avg. reward: 70, 10, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.52 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

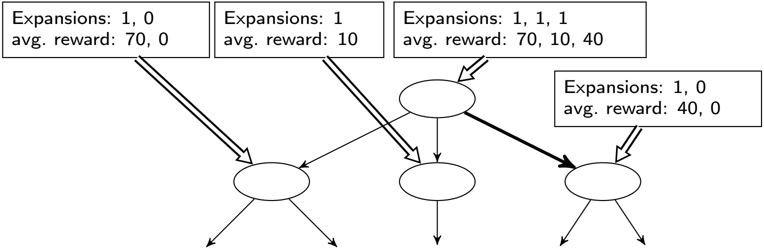
Expansions: 1, 1, 1
avg. reward: 70, 10, 40

Expansions: 1, 0
avg. reward: 40, 0

SOME RIGHTS RESERVED

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.53 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

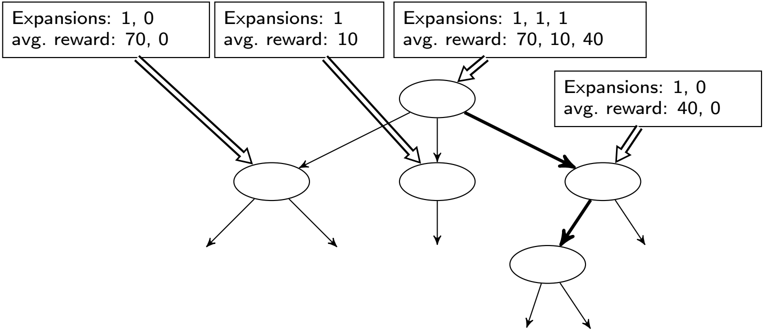
Expansions: 1
avg. reward: 10

Expansions: 1, 1, 1
avg. reward: 70, 10, 40

Expansions: 1, 0
avg. reward: 40, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.54 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

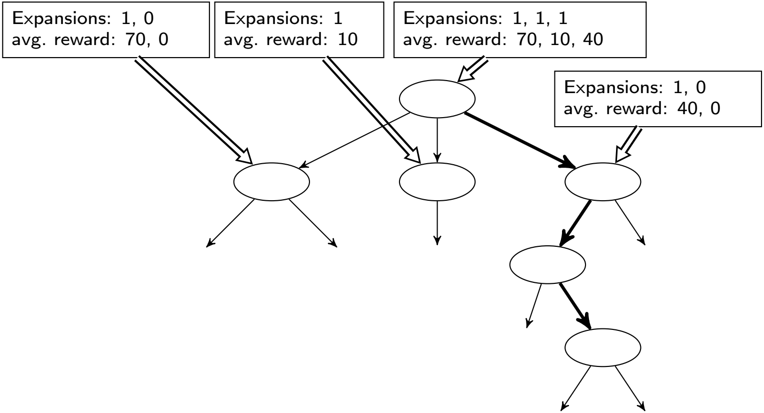
Expansions: 1
avg. reward: 10

Expansions: 1, 1, 1
avg. reward: 70, 10, 40

Expansions: 1, 0
avg. reward: 40, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.55 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

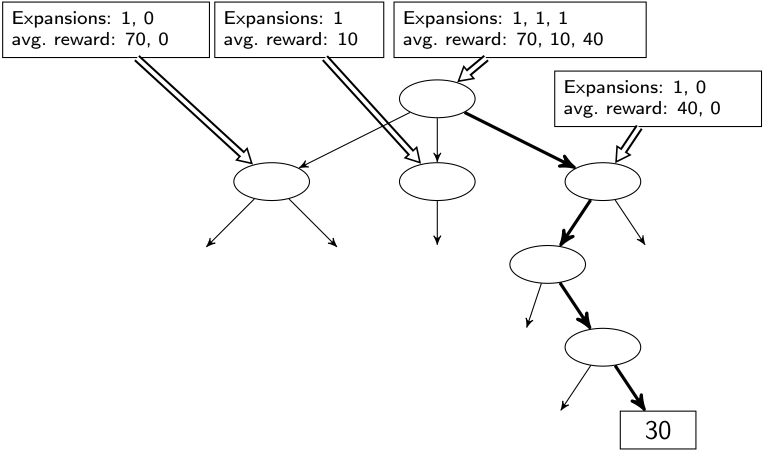
Expansions: 1
avg. reward: 10

Expansions: 1, 1, 1
avg. reward: 70, 10, 40

Expansions: 1, 0
avg. reward: 40, 0

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.56 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

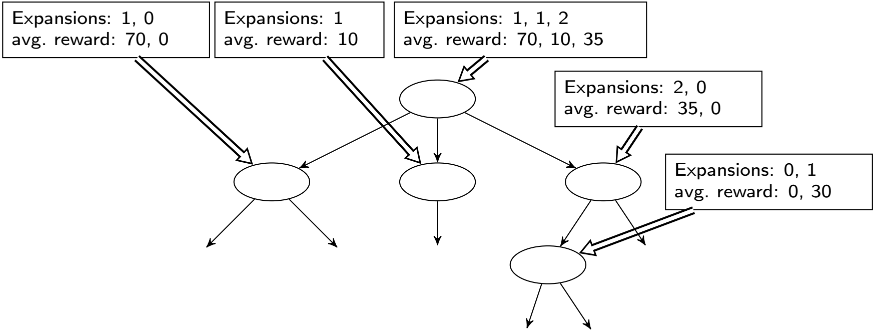
Expansions: 1, 1, 1
avg. reward: 70, 10, 40

Expansions: 1, 0
avg. reward: 40, 0

SOME RIGHTS RESERVED

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.57 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

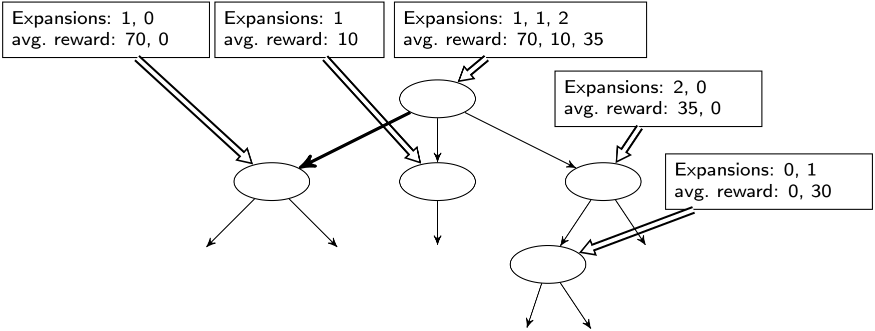
Expansions: 1, 1, 2
avg. reward: 70, 10, 35

Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.58 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

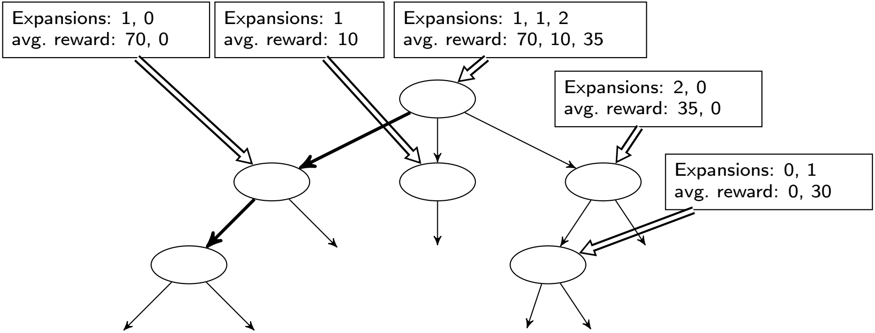
Expansions: 1, 1, 2
avg. reward: 70, 10, 35

Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.59 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

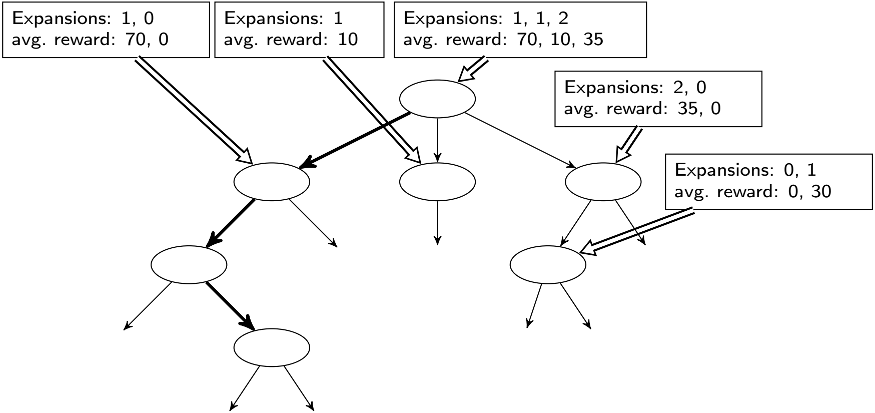
Expansions: 1, 1, 2
avg. reward: 70, 10, 35

Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.60 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

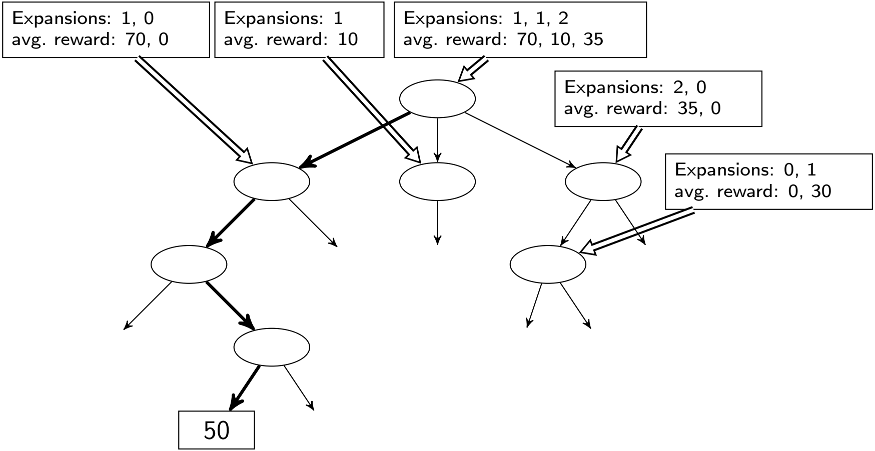
Expansions: 1, 1, 2
avg. reward: 70, 10, 35

Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.61 (Redoing the previous example).**



Expansions: 1, 0
avg. reward: 70, 0

Expansions: 1
avg. reward: 10

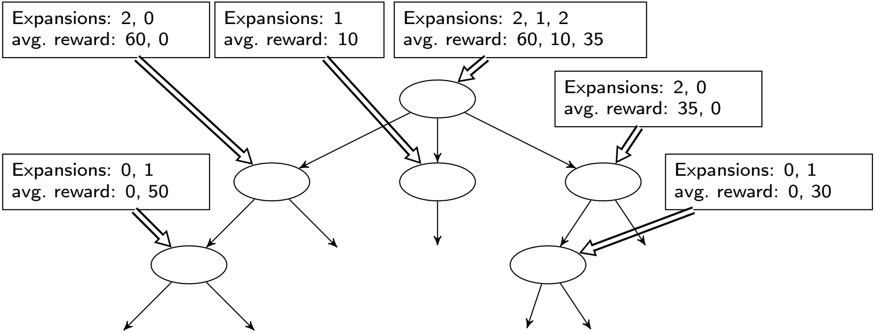
Expansions: 1, 1, 2
avg. reward: 70, 10, 35

Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.62 (Redoing the previous example).**



Expansions: 2, 0
avg. reward: 60, 0

Expansions: 1
avg. reward: 10

Expansions: 2, 1, 2
avg. reward: 60, 10, 35

Expansions: 2, 0
avg. reward: 35, 0

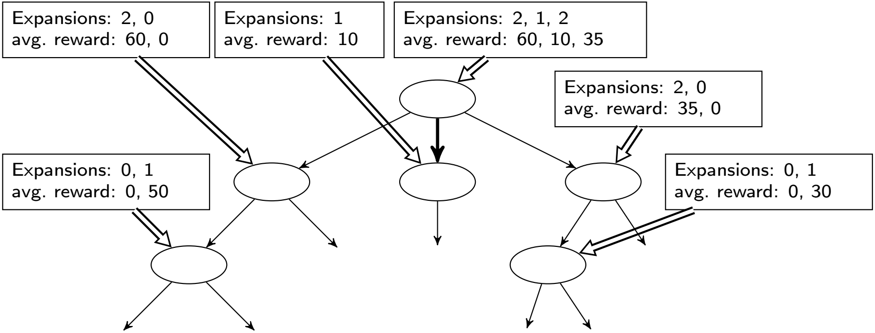
Expansions: 0, 1
avg. reward: 0, 50

Expansions: 0, 1
avg. reward: 0, 30

SOME RIGHTS RESERVED

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.63 (Redoing the previous example).**



Expansions: 2, 0
avg. reward: 60, 0

Expansions: 1
avg. reward: 10

Expansions: 2, 1, 2
avg. reward: 60, 10, 35

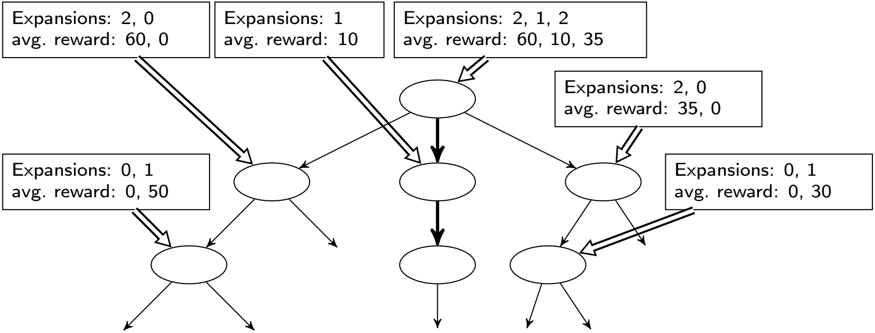
Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 50

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.64 (Redoing the previous example).**



Expansions: 2, 0
avg. reward: 60, 0

Expansions: 1
avg. reward: 10

Expansions: 2, 1, 2
avg. reward: 60, 10, 35

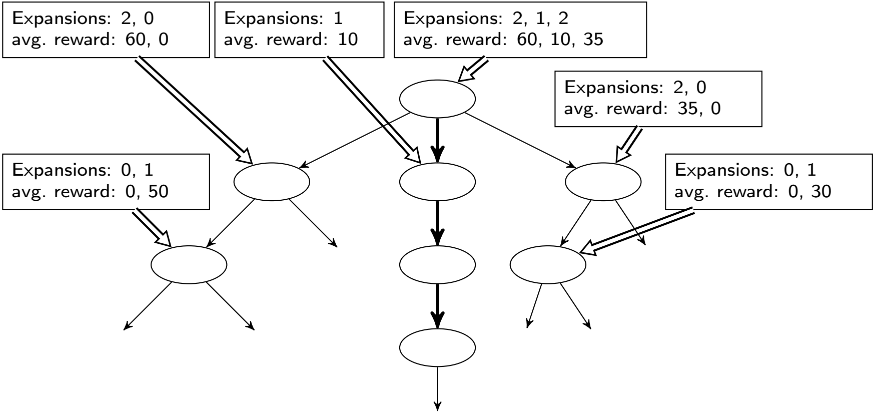
Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 50

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

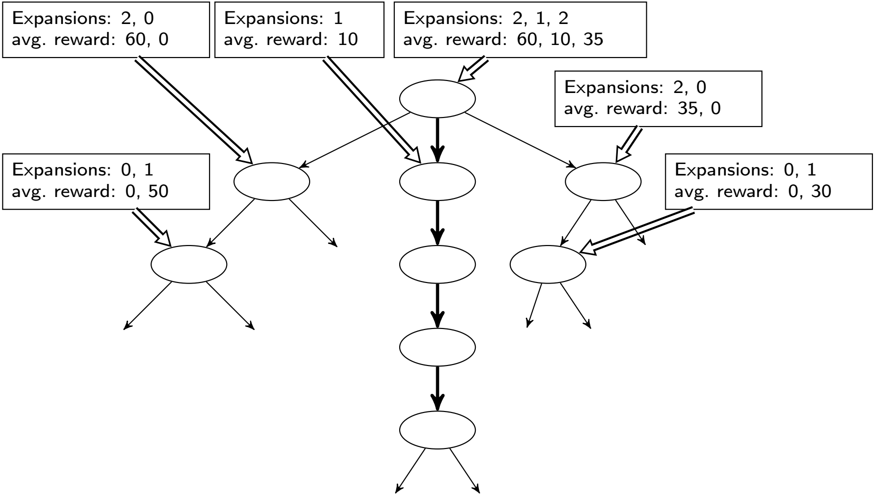
- **Idea:** We can save work by building the tree as we go along.
- **Example 5.65 (Redoing the previous example).**



SOME RIGHTS RESERVED

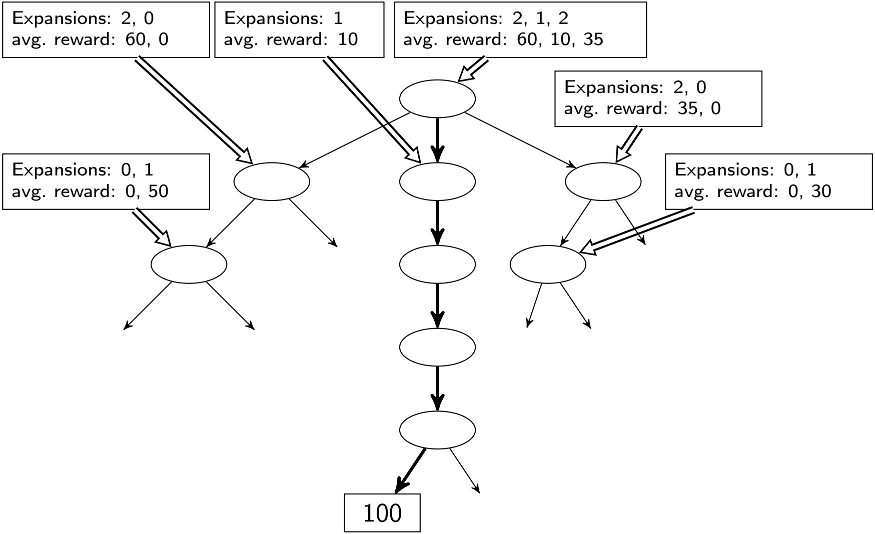
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.66 (Redoing the previous example).**



Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.67 (Redoing the previous example).**



Expansions: 2, 0
avg. reward: 60, 0

Expansions: 1
avg. reward: 10

Expansions: 2, 1, 2
avg. reward: 60, 10, 35

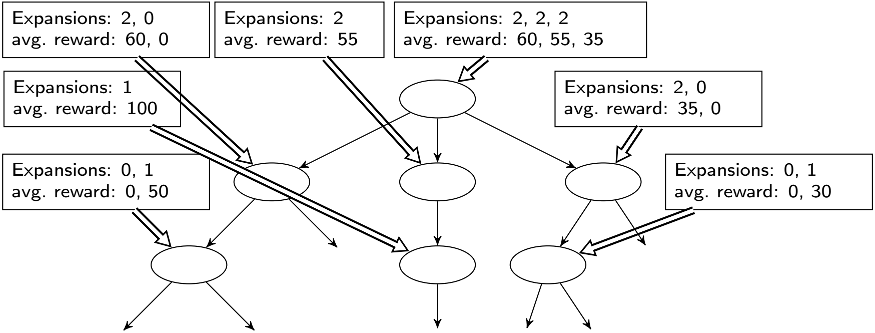
Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 50

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.68 (Redoing the previous example).**



Expansions: 2, 0
avg. reward: 60, 0

Expansions: 2
avg. reward: 55

Expansions: 2, 2, 2
avg. reward: 60, 55, 35

Expansions: 1
avg. reward: 100

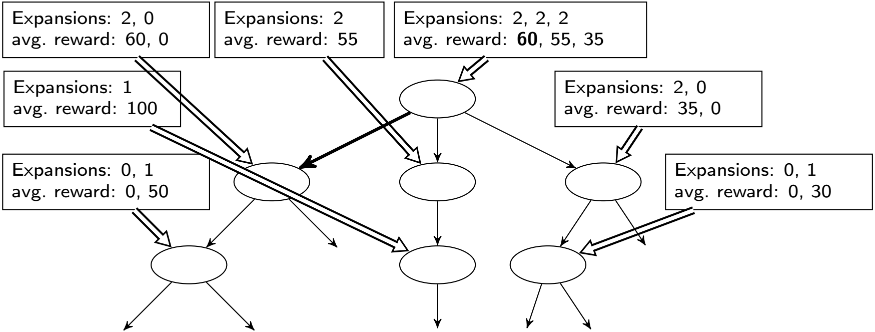
Expansions: 2, 0
avg. reward: 35, 0

Expansions: 0, 1
avg. reward: 0, 50

Expansions: 0, 1
avg. reward: 0, 30

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.69 (Redoing the previous example).**



Expansions: 2, 0
avg. reward: 60, 0

Expansions: 2
avg. reward: 55

Expansions: 2, 2, 2
avg. reward: **60**, 55, 35

Expansions: 1
avg. reward: 100

Expansions: 2, 0
avg. reward: 35, 0

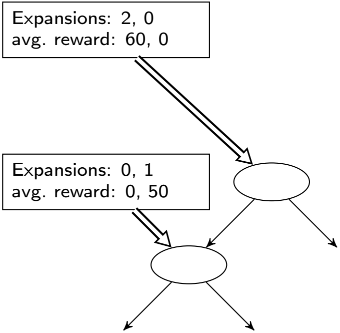
Expansions: 0, 1
avg. reward: 0, 50

Expansions: 0, 1
avg. reward: 0, 30

SOME RIGHTS RESERVED

Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.70 (Redoing the previous example).**



Expansions: 2, 0
avg. reward: 60, 0

Expansions: 0, 1
avg. reward: 0, 50

How to Guide the Search in MCTS?

- **How to sample?:** What exactly is "random"?
- **Classical formulation:** balance exploitation vs. exploration.
  - Exploitation: Prefer moves that have high average already (interesting regions of state space)
  - Exploration: Prefer moves that have not been tried a lot yet (don't overlook other, possibly better, options)
- UCT: "Upper Confidence bounds applied to Trees" [**kocsis:szepesvari:ecml-06**].
  - Inspired by Multi-Armed Bandit (as in: Casino) problems.
  - Basically a formula defining the balance. Very popular (buzzword).
  - Recent critics (e.g. [**feldman:domshlak:jair-14**]): Exploitation in search is very different from the Casino, as the "accumulated rewards" are fictitious (we're only thinking about the game, not actually playing and winning/losing all the time).

SOME RIGHTS RESERVED

AlphaGo: Overview

- **Definition 5.71 (Neural Networks in AlphaGo).**
  - Policy networks: Given a state s, output a probability distribution over the actions applicable in s.
  - Value networks: Given a state s, output a number estimating the game value of s.
- **Combination with MCTS:**
  - Policy networks bias the action choices within the MCTS tree (and hence the leaf state selection), and bias the random samples.
  - Value networks are an additional source of state values in the MCTS tree, along with the random samples.
- And now in a little more detail

SOME RIGHTS RESERVED

Neural Networks in AlphaGo

- **Neural network training pipeline and architecture:**

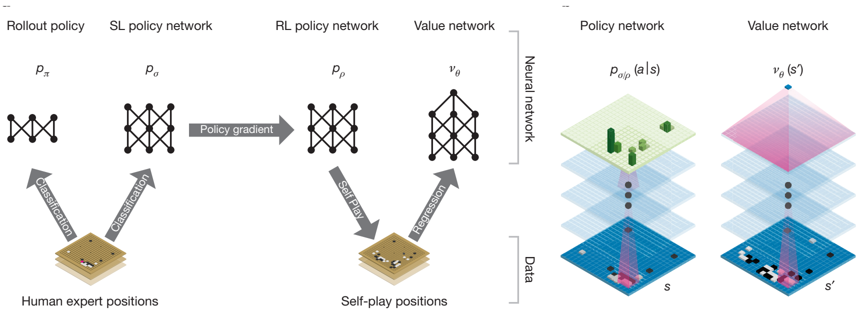


Illustration taken from [**silver:etal:nature-16**] .

- Rollout policy p_π: Simple but fast, ≈ prior work on Go.
- SL policy network p_σ: Supervised learning, human-expert data ("learn to choose an expert action").
- RL policy network p_ρ: Reinforcement learning, self-play ("learn to win").
- Value network v_θ: Use self-play games with p_ρ as training data for game-position evaluation v_θ ("predict which player will win in this state").

SOME RIGHTS RESERVED

Neural Networks + MCTS in AlphaGo

- **Monte Carlo tree search in AlphaGo:**

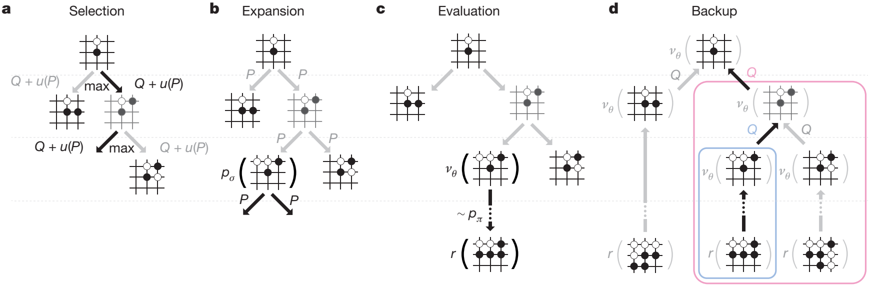


Illustration taken from [**silver:etal:nature-16**]

- *Rollout policy* p_π*:* Action choice in random samples.
- *SL policy network* p_σ*:* Action choice bias within the UCTS tree (stored as "P", gets smaller to "$u(P)$" with number of visits); along with quality Q.
- *RL policy network* p_ρ*:* Not used here (used only to learn v_θ).
- *Value network* v_θ*:* Used to evaluate leaf states s, in linear sum with the value returned by a random sample on s.

SOME RIGHTS RESERVED

7.6 State of the Art

State of the Art

- **Some well-known board games:**
  - **Chess**: Up next.
  - **Othello (Reversi)**: In 1997, "Logistello" beat the human world champion. Best computer players now are clearly better than best human players.
  - **Checkers (Dame)**: Since 1994, "Chinook" is the offical world champion. In 2007, it was shown to be *unbeatable*: Checkers is *solved*. (We know the exact value of, and optimal strategy for, the initial state.)
  - **Go**: In 2016, AlphaGo beat the Grandmaster Lee Sedol, cracking the "holy grail" of board games. In 2017, "AlphaZero" – a variant of AlphaGo with zero prior knowledge beat all reigning champion systems in all board games (including AlphaGo) 100/0 after 24h of self-play.
  - **Intuition:** Board Games are considered a "solved problem" from the AI perspective.

SOME RIGHTS RESERVED

Computer Chess: "Deep Blue" beat Garry Kasparov in 1997

Duell Kasparow gegen Deep Blue (1997): *Demütigende Niederlage*

- 6 games, final score 3.5 : 2.5.
- Specialized chess hardware, 30 nodes with 16 processors each.
- Alphabeta search plus human knowledge. (more details in a moment)
- Nowadays, standard PC hardware plays at world champion level.

SOME RIGHTS RESERVED

Computer Chess: Famous Quotes

- The chess machine is an ideal one to start with, since(Claude Shannon (1949))
  1. the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate),
  2. it is neither so simple as to be trivial nor too difficult for satisfactory solution,
  3. chess is generally considered to require “thinking” for skilful play, [. . .]
  4. the discrete structure of chess fits well into the digital nature of modern computers.
- Chess is the drosophila of artificial intelligence. (Alexander Kronrod (1965))

SOME RIGHTS RESERVED

Computer Chess: Another Famous Quote

- In 1965, the Russian mathematician Alexander Kronrod said, “Chess is the Drosophila of artificial intelligence.”
  However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophilae. We would have some science, but mainly we would have very fast fruit flies. (John McCarthy (1997))

SOME RIGHTS RESERVED

7.7 Conclusion

SOME RIGHTS RESERVED

Summary

- Games (2-player turn-taking zero-sum discrete and finite games) can be understood as a simple extension of classical search problems.
- Each player tries to reach a terminal state with the best possible utility (maximal vs. minimal).
- Minimax searches the game depth-first, max'ing and min'ing at the respective turns of each player. It yields perfect play, but takes time $\mathcal{O}(b^d)$ where b is the branching factor and d the search depth.
- Except in trivial games (Tic-Tac-Toe), minimax needs a depth limit and apply an evaluation function to estimate the value of the cut-off states.
- Alpha-beta search remembers the best values achieved for each player elsewhere in the tree already, and prunes out sub-trees that won't be reached in the game.
- Monte Carlo tree search (MCTS) samples game branches, and averages the findings. AlphaGo controls this using neural networks: evaluation function ("value network"), and action filter ("policy network").

SOME RIGHTS RESERVED

Chapter 8
Constraint Satisfaction Problems

8.1 Constraint Satisfaction Problems: Motivation

SOME RIGHTS RESERVED

A (Constraint Satisfaction) Problem

- **Example 1.1 (Tournament Schedule).** Who's going to play against who, when and where?

SOME RIGHTS RESERVED

Constraint Satisfaction Problems (CSPs)

- Standard search problem: state is a "black box" any old data structure that supports goal test, eval, successor state, ...
- **Definition 1.2.** A constraint satisfaction problem (CSP) is a triple $\langle V, D, C\rangle$ where
  1. V is a finite set V of variables,
  2. an V-indexed family $(D_v)_{v \in V}$ of domains, and
  3. for some subsets $\{v_1, \ldots, v_k\} \subseteq V$ a constraint $C_{\{v_1,\ldots,v_k\}} \subset D_{v_1} \times \ldots \times D_{v_k}$.

  A variable assignment $\varphi \in (v \in V) \to D_v$ is a solution for C, iff $\langle \varphi(v_1), \ldots, \varphi(v_k)\rangle \in C_{\{v_1,\ldots,v_k\}}$ for all $\{v_1, \ldots, v_k\} \subseteq V$.

  Definition 1.3. Let $\langle V, D, C\rangle$ be a CSP, then the order $\mathrm{ord}(C_V)$ of a constraint $C_V \in C$ is $\#(V)$, the order of $\langle V, D, C\rangle$ itself is $\max_{C_V \in C} \#(V)$.

  A constraint of order 1 is called unary, one of order 2 binary, and a constraint c is higher-order, iff $\mathrm{ord}(c) > 2$.
- **Definition 1.4.** A CSP γ is called satisfiable, iff it has a solution: a total variable assignment φ that satisfies all constraints.
- **Definition 1.5.** The process of finding solutions to CSPs is called constraint solving.
- *Remark 1.6.* We are using factored representation for world states now!
- Allows useful *general-purpose* algorithms with more power than standard tree search algorithm.

SOME RIGHTS RESERVED

Another Constraint Satisfaction Problem

- **Example 1.7 (SuDoKu).** Fill the cells with row/column/block-unique digits

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

$\leadsto$

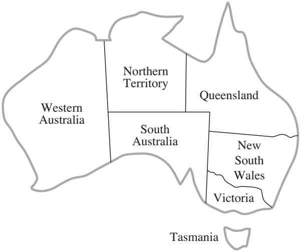
2	5	8	7	3	6	9	4	1
6	1	9	8	2	4	3	5	7
4	3	7	9	1	5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

  - Variables: The 81 cells.
  - Domains: Numbers 1, . . . , 9.
  - Constraints: Each number only once in each row, column, block.

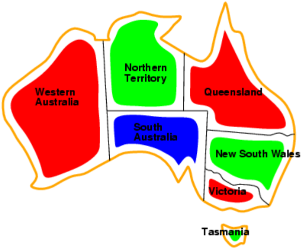
SOME RIGHTS RESERVED

CSP Example: Map-Coloring

- **Definition 1.8.** Given a map M, the map coloring problem is to assign colors to regions in a map so that no adjoining regions have the same color.
- **Example 1.9 (Map coloring in Australia).**



- Variables: $\mathrm{WA}, \mathrm{NT}, \mathrm{Q}, \mathrm{NSW}, \mathrm{V}, \mathrm{SA}, \mathrm{T}$
- Domains: $D_i = \{\mathrm{red}, \mathrm{green}, \mathrm{blue}\}$
- Constraints: adjacent regions must have different colors e.g., $\mathrm{WA} \neq \mathrm{NT}$ (if the language allows this), or $\langle \mathrm{WA}, \mathrm{NT} \rangle \in \{\langle \mathrm{red}, \mathrm{green} \rangle, \langle \mathrm{red}, \mathrm{blue} \rangle, \langle \mathrm{green}, \mathrm{red} \rangle, \ldots\}$



- **Intuition**: solutions map variables to domain values satisfying all constraints,
- e.g., $\{\mathrm{WA} = \mathrm{red}, \mathrm{NT} = \mathrm{green}, \ldots\}$

Bundesliga Constraints

- Variables: $v_{A\text{vs.}B}$ where A and B are teams, with domains $\{1,\ldots,34\}$: For each match, the index of the weekend where it is scheduled.
- (Some) constraints:

- If $\{A,B\} \cap \{C,D\} \neq \emptyset$: $v_{A\text{vs.}B} \neq v_{C\text{vs.}D}$ (each team only one match per day).
- If $\{A,B\} = \{C,D\}$: $v_{A\text{vs.}B} \leq 17 < v_{C\text{vs.}D}$ or $v_{C\text{vs.}D} \leq 17 < v_{A\text{vs.}B}$ (each pairing exactly once in each half-season).
- If $A = C$: $v_{A\text{vs.}B} + 1 \neq v_{C\text{vs.}D}$ (each team alternates between home matches and away matches).
- Leading teams of last season meet near the end of each half-season.
- …

SOME RIGHTS RESERVED

How to Solve the Bundesliga Constraints?

- 306 nested for-loops (for each of the 306 matches), each ranging from 1 to 306. Within the innermost loop, test whether the current values are (a) a permutation and, if so, (b) a legal Bundesliga schedule.
  - **Estimated running time**: End of this universe, and the next couple billion ones after it . . .
- Directly enumerate all permutations of the numbers $1, \ldots, 306$, test for each whether it's a legal Bundesliga schedule.
  - **Estimated running time**: Maybe only the time span of a few thousand universes.
- View this as variables/constraints and use backtracking (this chapter)
  - **Executed running time**: About 1 minute.
- **How do they actually do it?:** Modern computers and CSP methods: fractions of a second. 19th (20th/21st?) century: Combinatorics and manual work.
- **Try it yourself:** with an off-the shelf CSP solver, e.g. Minion [**minion:URL**]

SOME RIGHTS RESERVED

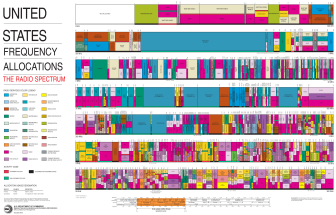
More Constraint Satisfaction Problems

Traveling Tournament Problem

Scheduling

Timetabling

Radio Frequency Assignment



Our Agenda for This Topic

- Our treatment of the topic “Constraint Satisfaction Problems” consists of Chapters 7 and 8. in [**RusNor:AIMA03**]
- **This Chapter**: Basic definitions and concepts; naïve backtracking search.
  - Sets up the framework. Backtracking underlies many successful algorithms for solving constraint satisfaction problems (and, naturally, we start with the simplest version thereof).
- **Next Chapter**: Constraint propagation and decomposition methods.
  - Constraint propagation reduces the search space of backtracking. Decomposition methods break the problem into smaller pieces. Both are crucial for efficiency in practice.

SOME RIGHTS RESERVED

Our Agenda for This Chapter

- How are constraint networks, and assignments, consistency, solutions: How are constraint satisfaction problems defined? What is a solution?
  - Get ourselves on firm ground.

Our Agenda for This Chapter

- How are constraint networks, and assignments, consistency, solutions: How are constraint satisfaction problems defined? What is a solution?
  - Get ourselves on firm ground.
- **Naïve Backtracking**: How does backtracking work? What are its main weaknesses?
  - Serves to understand the basic workings of this wide-spread algorithm, and to motivate its enhancements.

Our Agenda for This Chapter

- How are constraint networks, and assignments, consistency, solutions: How are constraint satisfaction problems defined? What is a solution?
  - Get ourselves on firm ground.
- **Naïve Backtracking**: How does backtracking work? What are its main weaknesses?
  - Serves to understand the basic workings of this wide-spread algorithm, and to motivate its enhancements.
- **Variable- and Value Ordering**: How should we guide backtracking searchs?
  - Simple methods for making backtracking aware of the structure of the problem, and thereby reduce search.

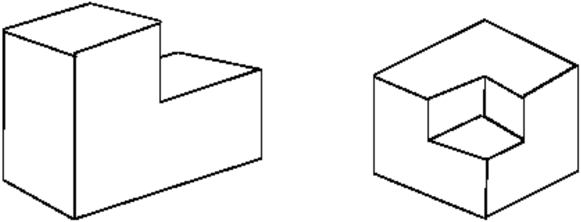
SOME RIGHTS RESERVED

8.2 The Waltz Algorithm

SOME RIGHTS RESERVED

The Waltz Algorithm

- **Remark:** One of the earliest examples of applied CSPs.
- **Motivation:** Interpret line drawings of polyhedra.

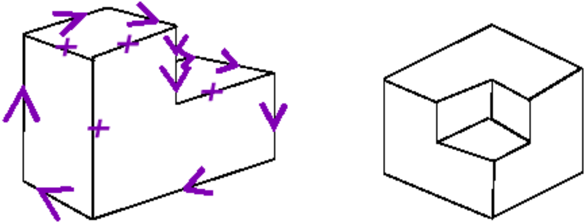


- **Problem:** Are intersections convex or concave? (interpret $\widehat{=}$ label as such)
- **Idea:** Adjacent intersections impose constraints on each other. Use CSP to find a unique set of labelings.

SOME RIGHTS RESERVED

Waltz Algorithm on Simple Scenes

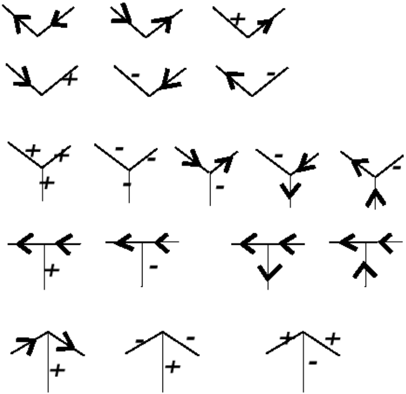
- **Assumptions:** All objects
  - have no shadows or cracks,
  - have only three-faced vertices,
  - are in “general position”, i.e. no junctions change with small movements of the eye.
- **Observation 2.1.** *Then each line on the images is one of the following:*
  - *a boundary line (edge of an object) (<) with right hand of arrow denoting “solid” and left hand denoting “space”*
  - *an interior convex edge* (label with “+”)
  - *an interior concave edge* (label with “-”)



SOME RIGHTS RESERVED

18 Legal Kinds of Junctions

- **Observation 2.2.** *There are only 18 "legal" kinds of junctions:*

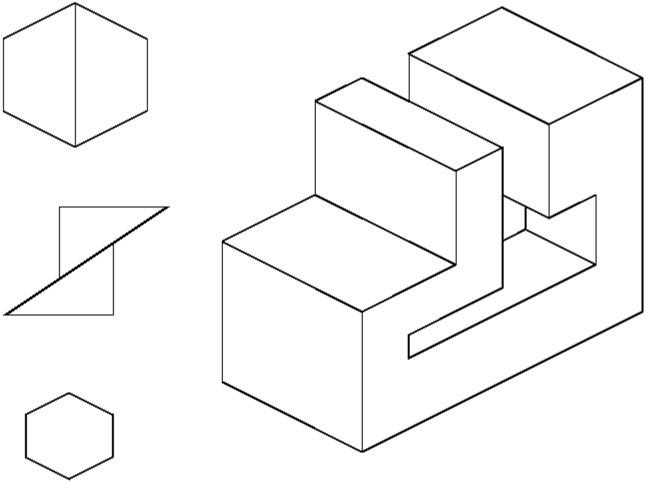


- **Idea:** given a representation of a diagram
  - label each junction in one of these manners (lots of possible ways)
  - junctions must be labeled, so that lines are labeled consistently
- **Fun Fact:** CSP always works perfectly! (early success story for CSP [**Waltz:uldss75**])

SOME RIGHTS RESERVED

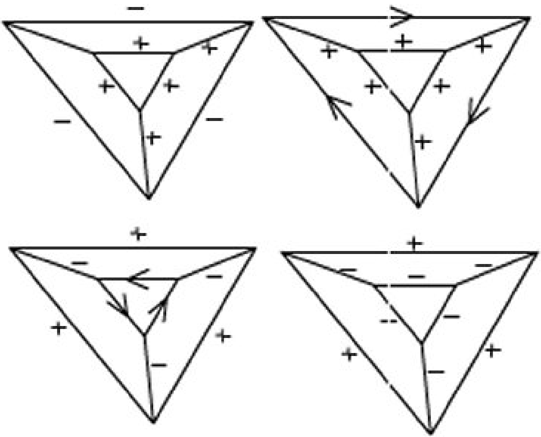
Waltz's Examples

- In his dissertation 1972 [**Waltz:uldss75**] David Waltz used the following examples

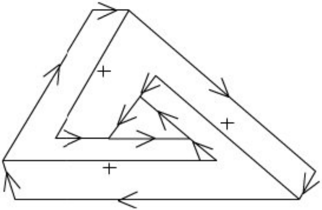


SOME RIGHTS RESERVED

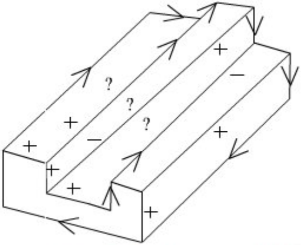
Waltz Algorithm (More Examples): Ambiguous Figures



Waltz Algorithm (More Examples): Impossible Figures



Consistent labelling for impossible figure



No consistent labelling possible

SOME RIGHTS RESERVED

8.3 CSP: Towards a Formal Definition

SOME RIGHTS RESERVED

Types of CSPs

- **Definition 3.1.** We call a CSP discrete, iff all of the variables have countable domains; we have two kinds:
  - finite domains (size $d \rightsquigarrow \mathcal{O}(d^n)$ solutions)
    - e.g., Boolean CSPs (solvability $\widehat{=}$ Boolean satisfiability $\rightsquigarrow$ NP-hard)
  - infinite domains (e.g. integers, strings, etc.)
    - e.g., job scheduling, variables are start/end days for each job
    - need a "constraint language", e.g., $StartJob_1 + 5 \leq StartJob_3$
    - linear constraints decidable, nonlinear ones undecidable
- **Definition 3.2.** We call a CSP continuous, iff one domain is uncountable.
- **Example 3.3.** Start/end times for Hubble Telescope observations form a continuous CSP.
- **Theorem 3.4.** *Linear constraints solvable in poly time by linear programming methods.*
- **Theorem 3.5.** *There cannot be optimal algorithms for nonlinear constraint systems.*

SOME RIGHTS RESERVED

Types of Constraints

- We classify the constraints by the number of variables they involve.
- **Definition 3.6.** Unary constraints involve a single variable, e.g., $\mathrm{SA} \neq$ *green*.
- **Definition 3.7.** Binary constraints involve pairs of variables, e.g., $\mathrm{SA} \neq \mathrm{WA}$.
- **Definition 3.8.** Higher-order constraints involve $n = 3$ or more variables, e.g., cryptarithmetic column constraints.
  The number n of variables is called the order of the constraint.
- **Definition 3.9.** Preferences (soft constraints) (e.g., red is better than green) are often representable by a cost for each variable assignment $\leadsto$ constrained optimization problems.

SOME RIGHTS RESERVED

Non-Binary Constraints, e.g. "Send More Money"

- **Example 3.10 (Send More Money).** A student writes home:

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

Puzzle: letters stand for digits, addition should work out (parents send MONEY€)

  - Variables: S, E, N, D, M, O, R, Y, each with domain $\{0, \ldots, 9\}$.
  - Constraints:
    1. all variables should have different values: $S \neq E$, $S \neq N$, ...
    2. first digits are non-zero: $S \neq 0$, $M \neq 0$.
    3. the addition scheme should work out: i.e.
       $1000 \cdot S + 100 \cdot E + 10 \cdot N + D + 1000 \cdot M + 100 \cdot O + 10 \cdot R + E = 10000 \cdot M + 1000 \cdot 0 + 100 \cdot N + 10 \cdot E + Y$.

  BTW: The solution is $S \mapsto 9, E \mapsto 5, N \mapsto 6, D \mapsto 7, M \mapsto 1, O \mapsto 0, R \mapsto 8, Y \mapsto 2 \leadsto$ parents send 10652€

- **Definition 3.11.** Problems like the one in 3.10 are called crypto-arithmetic puzzles.

SOME RIGHTS RESERVED

Encoding Higher-Order Constraints as Binary ones

- **Problem:** The last constraint is of order 8. ($n = 8$ variables involved)
- **Observation 3.12.** *We can write the addition scheme constraint column wise using auxiliary variables, i.e. variables that do not "occur" in the original problem.*

$$\begin{aligned} D + E &= Y + 10 \cdot X_1 \\ X_1 + N + R &= E + 10 \cdot X_2 \\ X_2 + E + O &= N + 10 \cdot X_3 \\ X_3 + S + M &= O + 10 \cdot M \end{aligned}$$

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

These constraints are of order ≤ 5.

- **General Recipe:** For $n \geq 3$, encode $C(v_1, \ldots, v_{n-1}, v_n)$ as

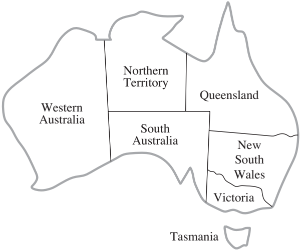
$$C(p_1(x), \ldots, p_{n-1}(x), v_n) \wedge v_1 = p_1(x) \wedge \ldots \wedge v_{n-1} = p_{n-1}(x)$$

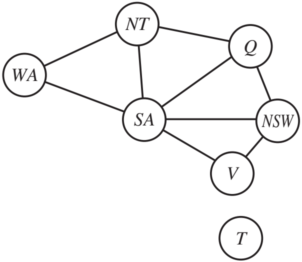
- **Problem:** The problem structure gets hidden. (search algorithms can get confused)

SOME RIGHTS RESERVED

Constraint Graph

- **Definition 3.13.** A binary CSP is a CSP where each constraint is unary or binary.
- **Observation 3.14.** *A binary CSP forms a graph called the constraint graph whose nodes are variables, and whose edges represent the constraints.*
- **Example 3.15.** Australia as a binary CSP





- **Intuition:** General-purpose CSP algorithms use the graph structure to speed up search. (E.g., Tasmania is an independent subproblem!)

SOME RIGHTS RESERVED

Real-world CSPs

- **Example 3.16 (Assignment problems).** e.g., who teaches what class
- **Example 3.17 (Timetabling problems).** e.g., which class is offered when and where?
- **Example 3.18 (Hardware configuration).**
- **Example 3.19 (Spreadsheets).**
- **Example 3.20 (Transportation scheduling).**
- **Example 3.21 (Factory scheduling).**
- **Example 3.22 (Floorplanning).**
- **Note:** many real-world problems involve real-valued variables $\rightsquigarrow$ continuous CSPs.

SOME RIGHTS RESERVED

8.4 Constraint Networks: Formalizing Binary CSPs

SOME RIGHTS RESERVED

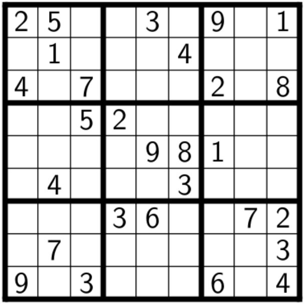
Constraint Networks (Formalizing binary CSPs)

- **Definition 4.1.** A constraint network is a constraint satisfaction problem of order 2. We will use C_v for $C_{\{u\}}$ and C_{uv} for $C_{\{u,v\}}$. Note that $C_{uv} = C_{vu}$
  Definition 4.2. We call the undirected graph $\langle V, \{(u,v) \in V^2 \mid C_{uv} \neq D_u \times D_v\}\rangle$, the constraint graph of γ.
- We will talk of CSPs and mean constraint networks.
- **Remarks:** The mathematical formulation gives us a lot of leverage:
  - $C_{uv} \subseteq D_u \times D_v \mathrel{\widehat{=}}$ possible assignments to variables u and v
  - Relations are the most general formalization, generally we use symbolic formulations, e.g. "$u = v$" for the relation $C_{uv} = \{(a,b) \mid a = b\}$ or "$u \neq v$".
  - We can express unary constraints C_u by restricting the domain of v: $D_v := C_v$.

SOME RIGHTS RESERVED

Example: SuDoKu as a Constraint Network

- **Example 4.3 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as 1.7.



2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

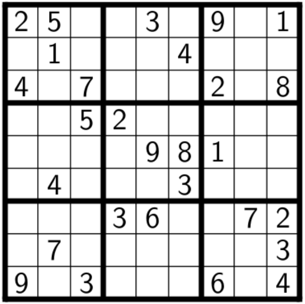
  - Variables:

Note that the ideas are still the same as 1.7, but in constraint networks we have a language to formulate things precisely.

SOME RIGHTS RESERVED

Example: SuDoKu as a Constraint Network

- **Example 4.4 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as 1.7.



2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

  - Variables: $V = \{v_{ij} \mid 1 \leq i, j \leq 9\}$: v_{ij} =cell in row i column j.
  - Domains

Note that the ideas are still the same as 1.7, but in constraint networks we have a language to formulate things precisely.

SOME RIGHTS RESERVED

Example: SuDoKu as a Constraint Network

- **Example 4.5 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as 1.7.

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

  - Variables: $V = \{v_{ij} \mid 1 \leq i, j \leq 9\}$: v_{ij} =cell in row i column j.
  - Domains For all $v \in V$: $D_v = D = \{1, \ldots, 9\}$.
  - Unary constraint:

Note that the ideas are still the same as 1.7, but in constraint networks we have a language to formulate things precisely.

SOME RIGHTS RESERVED

Example: SuDoKu as a Constraint Network

- **Example 4.6 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as 1.7.

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

  - Variables: $V = \{v_{ij} \mid 1 \leq i, j \leq 9\}$: v_{ij} =cell in row i column j.
  - Domains
  - Unary constraint: $C_{v_{ij}} = \{d\}$ if cell i, j is pre-filled with d.
  - (Binary) constraint:

  Note that the ideas are still the same as 1.7, but in constraint networks we have a language to formulate things precisely.

SOME RIGHTS RESERVED

Example: SuDoKu as a Constraint Network

- **Example 4.7 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as 1.7.

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

  - Variables: $V = \{v_{ij} \mid 1 \leq i,j \leq 9\}$: v_{ij} =cell in row i column j.
  - Domains
  - Unary constraint:
  - (Binary) constraint: $C_{v_{ij}v_{i'j'}} \widehat{=}$ "$v_{ij} \neq v_{i'j'}$", i.e. $C_{v_{ij}v_{i'j'}} = \{(d,d') \in D \times D \mid d \neq d'\}$, for: $i = i'$ (same row), or $j = j'$ (same column), or $(\lceil \frac{i}{3} \rceil, \lceil \frac{j}{3} \rceil) = (\lceil \frac{i'}{3} \rceil, \lceil \frac{j'}{3} \rceil)$ (same block).

Note that the ideas are still the same as 1.7, but in constraint networks we have a language to formulate things precisely.

SOME RIGHTS RESERVED

Constraint Networks (Solutions)

- Let $\gamma := \langle V, D, C, C, C, V, E\rangle$ be a constraint network.
- **Definition 4.8.** We call a partial function $a\colon\; V \rightharpoonup \bigcup_{u \in V} D_u$ a variable assignment if $a(u) \in D_u$ for all $u \in \mathrm{dom}(a)$.
- **Definition 4.9.** Let $\mathcal{C} := \langle V, D, C, C, C, V, E\rangle$ be a constraint network and $a\colon\; V \rightharpoonup \bigcup_{v \in V} D_v$ a variable assignment. We say that a satisfies (otherwise violates) a constraint C_{uv}, iff $u, v \in \mathrm{dom}(a)$ and $(a(u),a(v)) \in C_{uv}$. a is called consistent in $\mathcal{C}$, iff it satisfies all constraints in $\mathcal{C}$. A value $w \in D_u$ is legal for a variable u in $\mathcal{C}$, iff $\{(u,w)\}$ is a consistent assignment in $\mathcal{C}$. A variable with illegal value under a is called conflicted.
- **Example 4.10.** The empty assignment ϵ is (trivially) consistent in any constraint network.
- **Definition 4.11.** Let f and g be variable assignments, then we say that f extends (or is an extension of) g, iff $\mathrm{dom}(g) \subset \mathrm{dom}(f)$ and $f|_{\mathrm{dom}(g)} = g$.
- **Definition 4.12.** We call a consistent (total) assignment a solution for γ and γ itself solvable or satisfiable.

SOME RIGHTS RESERVED

How it all fits together

- **Lemma 4.13.** *Higher-order constraints can be transformed into equi-satisfiable binary constraints using auxiliary variables.*
- **Corollary 4.14.** *Any CSP can be represented by a constraint network.*
- **In other words** The notion of a constraint network is a refinement of a CSP.
- So we will stick to constraint networks in this course.
- **Observation 4.15.** *We can view a constraint network as a search problem, if we take the states as the variable assignments, the actions as assignment extensions, and the goal states as consistent assignments.*
- **Idea:** We will explore that idea for algorithms that solve constraint networks.

SOME RIGHTS RESERVED

8.5 CSP as Search

SOME RIGHTS RESERVED

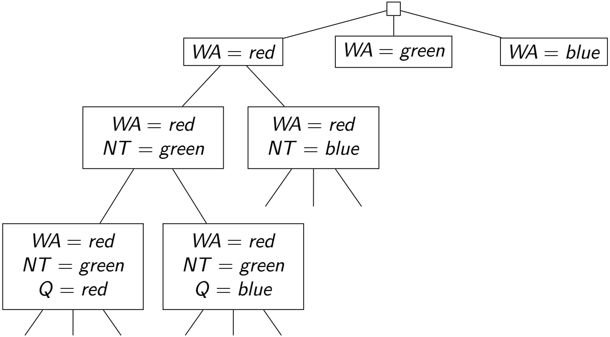
Standard search formulation (incremental)

- **Idea:** Every constraint network induces a single state problem.
- **Definition 5.1 (Let's do the math).** Given a constraint network $\gamma := \langle V, D, C, C, C, V, E \rangle$, then $\Pi_\gamma := \langle \mathcal{S}_\gamma, \mathcal{A}_\gamma, \mathcal{T}_\gamma, \mathcal{I}_\gamma, \mathcal{G}_\gamma \rangle$ is called the search problem induced by γ, iff
  - State $\mathcal{S}_\gamma$ are variable assignments
  - Action $\mathcal{A}_\gamma$: extend $\varphi \in \mathcal{S}_\gamma$ by a pair $x \mapsto v$ not conflicted with φ.
  - Transition model $\mathcal{T}_\gamma(a, \varphi) = \varphi, x \mapsto v$ (extended assignment)
  - Initial state $\mathcal{I}_\gamma$: the empty assignment ϵ.
  - Goal states $\mathcal{G}_\gamma$: the total, consistent assignments
- **What has just happened?:** We interpret a constraint network γ as a search problem Π_γ. A solution to Π_γ induces a solution to γ.
- **Idea:** We have algorithms for that: e.g. tree search.
- **Remark:** This is the same for all CSPs! ☺
  $\rightsquigarrow$ fail if no consistent assignments exist (not fixable!)

SOME RIGHTS RESERVED

Standard search formulation (incremental)

- **Example 5.2.** A search tree for $\Pi_{Australia}$:



- **Observation:** Every solution appears at depth n with n variables.
- **Idea:** Use depth first search!
- **Observation:** Path is irrelevant $\rightsquigarrow$ can use local search algorithms.
- Branching factor $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!! ☹

SOME RIGHTS RESERVED

Backtracking Search

- Assignments for different variables are independent!
  - e.g. first $\mathrm{WA} = \mathrm{red}$ then $\mathrm{NT} = \mathrm{green}$ vs. first $\mathrm{NT} = \mathrm{green}$ then $\mathrm{WA} = \mathrm{red}$
  - $\leadsto$ we only need to consider assignments to a single variable at each node
  - $\leadsto$ $b = d$ and there are d^n leaves.
- **Definition 5.3.** Depth first search for CSPs with single-variable assignment extensions actions is called backtracking search.
- Backtracking search is the basic uninformed algorithm for CSPs.
- It can solve the n-queens problem for $\approxeq n, 25$.

SOME RIGHTS RESERVED

Backtracking Search (Implementation)

- **Definition 5.4.** The generic backtracking search algorithm:

```
procedure Backtracking−Search(csp ) returns solution/failure
     return Recursive−Backtracking ($\emptyset$, csp)

procedure Recursive−Backtracking (assignment) returns soln/failure
  if assignment is complete then return assignment
  var := Select−Unassigned−Variable(Variables[csp], assignment, csp)
  foreach value in Order−Domain−Values(var, assignment, csp) do
      if value is consistent with assignment given Constraints[csp] then
          add {var = value} to assignment
          result := Recursive−Backtracking(assignment,csp)
          if result $\neq$ failure then return result
          remove {var= value} from assignment
  return failure
```

SOME RIGHTS RESERVED

Backtracking in Australia

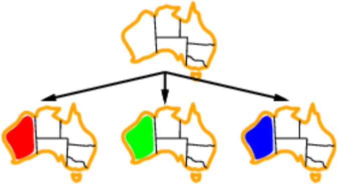
- **Example 5.5.** We apply backtracking search for a map coloring problem:

Step 1:

SOME RIGHTS RESERVED

Backtracking in Australia

- **Example 5.6.** We apply backtracking search for a map coloring problem:

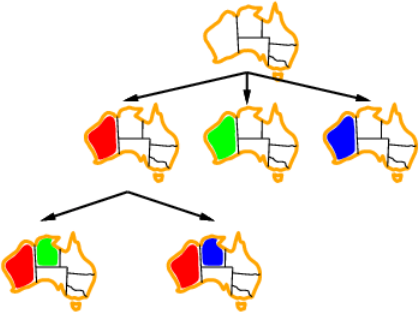


Step 2:

SOME RIGHTS RESERVED

Backtracking in Australia

- **Example 5.7.** We apply backtracking search for a map coloring problem:

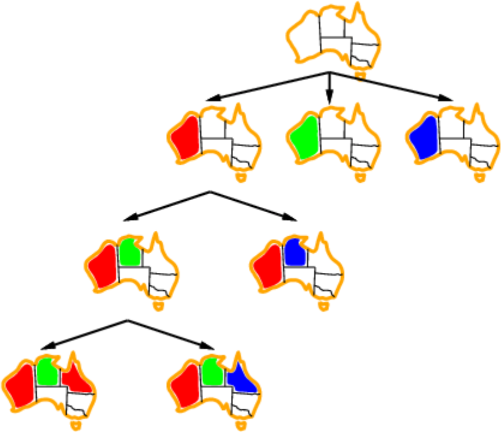


Step 3:

SOME RIGHTS RESERVED

Backtracking in Australia

- **Example 5.8.** We apply backtracking search for a map coloring problem:



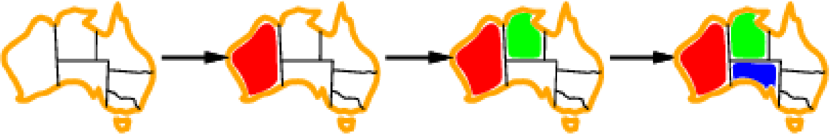
Step 4:

Improving Backtracking Efficiency

- General-purpose methods can give huge gains in speed for backtracking search.
- Answering the following questions well helps find powerful heuristics:
  1. Which variable should be assigned next? (i.e. a variable ordering heuristic)
  2. In what order should its values be tried? (i.e. a value ordering heuristic)
  3. Can we detect inevitable failure early? (for pruning strategies)
  4. Can we take advantage of problem structure? (⇝ inference)
- **Observation:** Questions 1/2 correspond to the missing subroutines Select−Unassigned−Variable and Order−Domain−Values from 5.4.

Heuristic: Minimum Remaining Values (Which Variable)

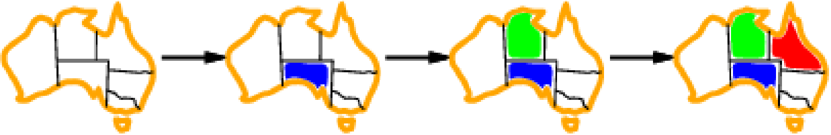
- **Definition 5.9.** The minimum remaining values (MRV) heuristic for backtracking search always chooses the variable with the fewest legal values, i.e. a variable v that given an initial assignment a minimizes $\#(\{d \in D_v \mid a \cup \{v \mapsto d\} \text{ is consistent}\})$.
- **Intuition:** By choosing a most constrained variable v first, we reduce the branching factor (number of sub trees generated for v) and thus reduce the size of our search tree.
- **Extreme case:** If $\#(\{d \in D_v \mid a \cup \{v \mapsto d\} \text{ is consistent}\}) = 1$, then the value assignment to v is forced by our previous choices.
- **Example 5.10.** In step 3 of 5.5, there is only one remaining value for SA!



SOME RIGHTS RESERVED

Degree Heuristic (Variable Order Tie Breaker)

- **Problem:** Need a tie-breaker among MRV variables! (there was no preference in step 1,2)
- **Definition 5.11.** The degree heuristic in backtracking search always chooses a most constraining variable, i.e. given an initial assignment a always pick a variable v with $\#(\{v \in (V \backslash \mathrm{dom}(a)) \,|\, C_{uv} \in C\})$ maximal.
- By choosing a most constraining variable first, we detect inconsistencies earlier on and thus reduce the size of our search tree.
- **Commonly used strategy combination:** From the set of most constrained variable, pick a most constraining variable.
- **Example 5.12.**

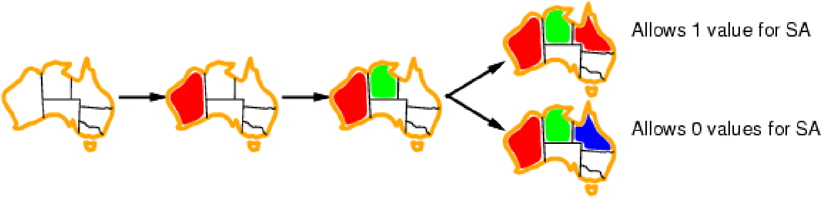


Degree heuristic: $\mathrm{SA} = 5$, $\mathrm{T} = 0$, all others 2 or 3.

SOME RIGHTS RESERVED

Least Constraining Value Heuristic (Value Ordering)

- **Definition 5.13.** Given a variable v, the least constraining value heuristic chooses the least constraining value for v: the one that rules out the fewest values in the remaining variables, i.e. for a given initial assignment a and a chosen variable v pick a value $d \in D_v$ that minimizes $\#(\{e \in D_u \mid u \notin \mathrm{dom}(a), C_{uv} \in C, \text{ and } (e,d) \notin C_{uv}\})$
- By choosing the least constraining value first, we increase the chances to not rule out the solutions below the current node.
- **Example 5.14.**



- Combining these heuristics makes 1000 queens feasible.

8.6 Conclusion & Preview

SOME RIGHTS RESERVED

Summary & Preview

- Summary of "CSP as Search":
  - Constraint networks γ consist of variables, associated with finite domains, and constraints which are binary relations specifying permissible value pairs.
  - A variable assignment a maps some variables to values. a is consistent if it complies with all constraints. A consistent total assignment is a solution.
  - The constraint satisfaction problem (CSP) consists in finding a solution for a constraint network. This has numerous applications including, e.g., scheduling and timetabling.
  - Backtracking search assigns variable one by one, pruning inconsistent variable assignments.
  - Variable orderings in backtracking can dramatically reduce the size of the search tree. Value orderings have this potential (only) in solvable sub trees.
- **Up next:** Inference and decomposition, for improved efficiency.

SOME RIGHTS RESERVED

Chapter 9
Constraint Propagation

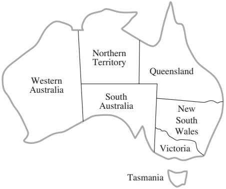
SOME RIGHTS RESERVED

9.1 Introduction

SOME RIGHTS RESERVED

Illustration: Constraint Propagation

- **Example 1.1.** A constraint network γ:

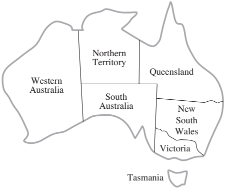


- **Question:** Can we add a constraint without losing any solutions?
- **Example 1.2.** $C_{\mathrm{WAQ}} :=$ "$=$". If WA and Q are assigned different colors, then NT must be assigned the 3rd color, leaving no color for SA.
- **Intuition:** Adding constraints without losing solutions
  $\widehat{=}$ obtaining an equivalent network with a "tighter description"
  $\leadsto$ a smaller number of consistent (partial) variable assignments
  $\leadsto$ more efficient search!

SOME RIGHTS RESERVED

Illustration: Decomposition

- **Example 1.3.** Constraint network γ:



- We can separate this into two independent constraint networks.
- Tasmania is not adjacent to any other state. Thus we can color Australia first, and assign an arbitrary color to Tasmania afterwards.
- Decomposition methods exploit the structure of the constraint network. They identify separate parts (sub-networks) whose inter-dependencies are "simple" and can be handled efficiently.
- **Example 1.4 (Extreme case).** No inter-dependencies at all, as for Tasmania above.

SOME RIGHTS RESERVED

Our Agenda for This Chapter

- Constraint propagation: How does inference work in principle? What are relevant practical aspects?
  - Fundamental concepts underlying inference, basic facts about its use.
- Forward checking: What is the simplest instance of inference?
  - Gets us started on this subject.
- Arc consistency: How to make inferences between variables whose value is not fixed yet?
  - Details a state of the art inference method.
- Decomposition: Constraint graphs, and two simple cases
  - How to capture dependencies in a constraint network? What are "simple cases"?
  - Basic results on this subject.
- Cutset conditioning: What if we're not in a simple case?
  - Outlines the most easily understandable technique for decomposition in the general case.

SOME RIGHTS RESERVED

9.2 Constraint Propagation/Inference

SOME RIGHTS RESERVED

Constraint Propagation/Inference: Basic Facts

- **Definition 2.1.** Constraint propagation (i.e inference in constraint networks) consists in deducing additional constraints, that follow from the already known constraints, i.e. that are satisfied in all solutions.
- **Example 2.2.** It's what you do all the time when playing SuDoKu:

	5	8	7		6	9	4	1
		9	8		4	3	5	7
4		7	9		5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

- **Formally:** Replace γ by an equivalent and strictly tighter constraint network γ'.

SOME RIGHTS RESERVED

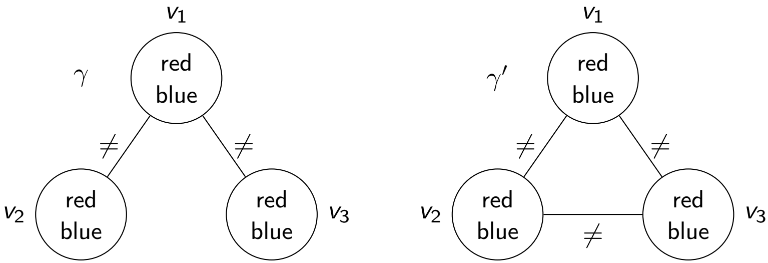
Equivalent Constraint Networks

- **Definition 2.3.** We say that two constraint networks $\gamma := \langle V, D, C, C, C, V, E\rangle$ and $\gamma' := \langle V, D', C'\rangle$ sharing the same set of variables are equivalent, (write $\gamma' \equiv \gamma$), if they have the same solutions.

SOME RIGHTS RESERVED

Equivalent Constraint Networks

- **Definition 2.5.** We say that two constraint networks $\gamma := \langle V, D, C, C, C, V, E \rangle$ and $\gamma' := \langle V, D', C' \rangle$ sharing the same set of variables are equivalent, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- **Example 2.6.**

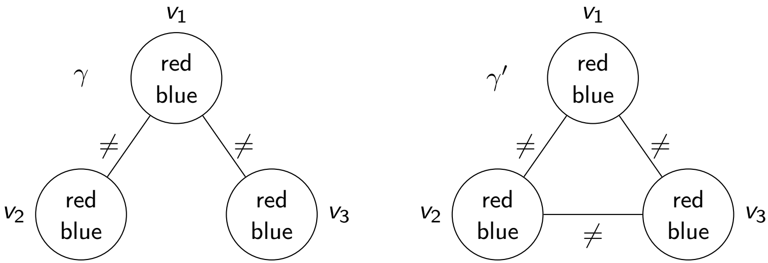


Are these constraint networks equivalent?

SOME RIGHTS RESERVED

Equivalent Constraint Networks

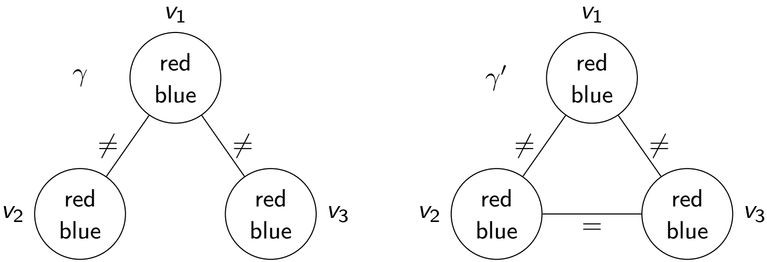
- **Definition 2.7.** We say that two constraint networks $\gamma := \langle V, D, C, C, C, V, E \rangle$ and $\gamma' := \langle V, D', C' \rangle$ sharing the same set of variables are equivalent, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- **Example 2.8.**



Are these constraint networks equivalent? No.

Equivalent Constraint Networks

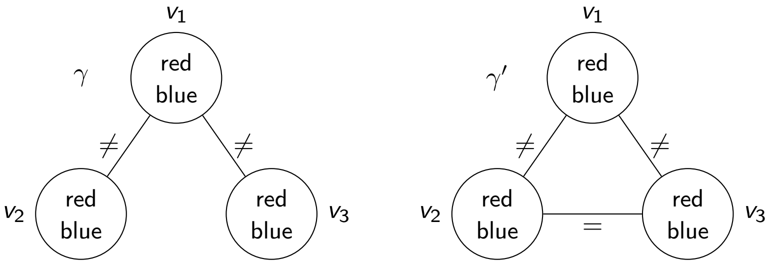
- **Definition 2.9.** We say that two constraint networks $\gamma := \langle V, D, C, C, C, V, E\rangle$ and $\gamma' := \langle V, D', C'\rangle$ sharing the same set of variables are equivalent, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- **Example 2.10.**



Are these constraint networks equivalent?

Equivalent Constraint Networks

- **Definition 2.11.** We say that two constraint networks $\gamma := \langle V, D, C, C, C, V, E \rangle$ and $\gamma' := \langle V, D', C' \rangle$ sharing the same set of variables are equivalent, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- **Example 2.12.**



Are these constraint networks equivalent? Yes.

Tightness

- **Definition 2.13 (Tightness).** Let $\gamma := \langle V, D, C, C, C, V, E\rangle$ and $\gamma' = \langle V, D', C', C_{\gamma'}, C_{\gamma'}, V_{\gamma'}, E_{\gamma'}\rangle$ be constraint networks sharing the same set of variables, then γ' is tighter than γ, (write $\gamma' \sqsubseteq \gamma$), if:
  (i) For all $v \in V$: $D'_v \subseteq D_v$.
  (ii) For all $u \neq v \in V$ and $C'_{uv} \in C'$: either $C'_{uv} \notin C$ or $C'_{uv} \subseteq C_{uv}$.

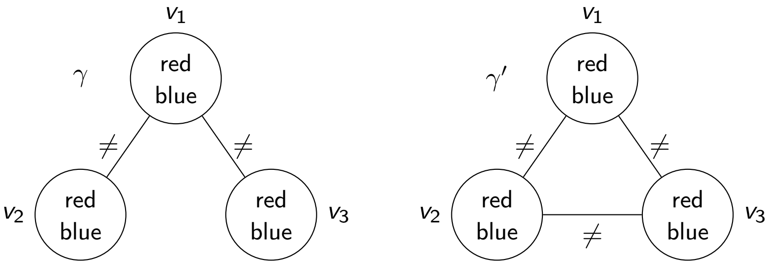
  γ' is strictly tighter than γ, (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is proper.

SOME RIGHTS RESERVED

Tightness

- **Definition 2.15 (Tightness).** Let $\gamma := \langle V, D, C, C, C, V, E\rangle$ and $\gamma' = \langle V, D', C', C_{\gamma'}, C_{\gamma'}, V_{\gamma'}, E_{\gamma'}\rangle$ be constraint networks sharing the same set of variables, then γ' is tighter than γ, (write $\gamma' \sqsubseteq \gamma$), if:
  (i) For all $v \in V$: $D'_v \subseteq D_v$.
  (ii) For all $u \neq v \in V$ and $C'_{uv} \in C'$: either $C'_{uv} \notin C$ or $C'_{uv} \subseteq C_{uv}$.

  γ' is strictly tighter than γ, (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is proper.
- **Example 2.16.**

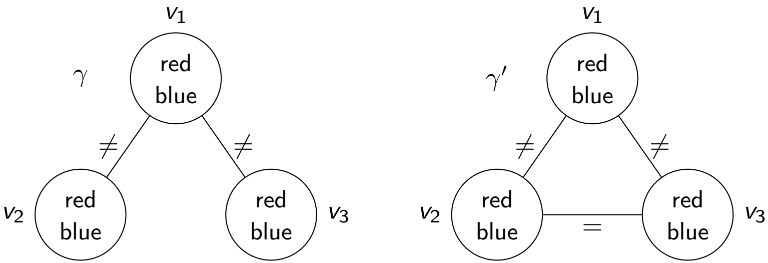


Here, we do have $\gamma' \sqsubseteq \gamma$.

Tightness

- **Definition 2.17 (Tightness).** Let $\gamma := \langle V, D, C, C, C, V, E \rangle$ and $\gamma' = \langle V, D', C', C_{\gamma'}, C_{\gamma'}, V_{\gamma'}, E_{\gamma'} \rangle$ be constraint networks sharing the same set of variables, then γ' is tighter than γ, (write $\gamma' \sqsubseteq \gamma$), if:
  (i) For all $v \in V$: $D'_v \subseteq D_v$.
  (ii) For all $u \neq v \in V$ and $C'_{uv} \in C'$: either $C'_{uv} \notin C$ or $C'_{uv} \subseteq C_{uv}$.

  γ' is strictly tighter than γ, (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is proper.
- **Example 2.18.**

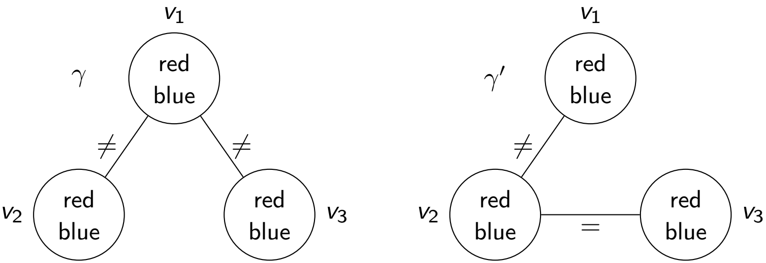


Here, we do have $\gamma' \sqsubseteq \gamma$.

Tightness

- **Definition 2.19 (Tightness).** Let $\gamma := \langle V, D, C, C, C, V, E\rangle$ and $\gamma' = \langle V, D', C', C_{\gamma'}, C_{\gamma'}, V_{\gamma'}, E_{\gamma'}\rangle$ be constraint networks sharing the same set of variables, then γ' is tighter than γ, (write $\gamma' \sqsubseteq \gamma$), if:
  - (i) For all $v \in V$: $D'_v \subseteq D_v$.
  - (ii) For all $u \neq v \in V$ and $C'_{uv} \in C'$: either $C'_{uv} \notin C$ or $C'_{uv} \subseteq C_{uv}$.

  γ' is strictly tighter than γ, (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is proper.
- **Example 2.20.**



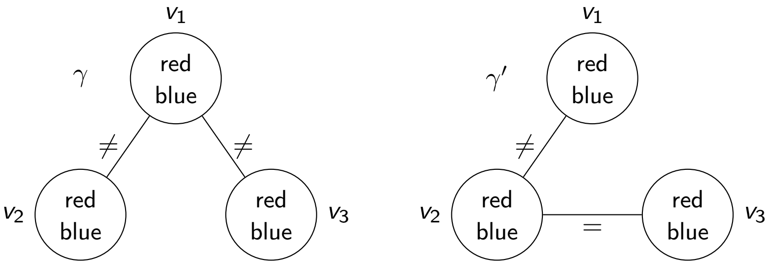
Here, we do not have $\gamma' \sqsubseteq \gamma$!.

SOME RIGHTS RESERVED

Tightness

- **Definition 2.21 (Tightness).** Let $\gamma := \langle V, D, C, C, C, V, E \rangle$ and $\gamma' = \langle V, D', C', C_{\gamma'}, C_{\gamma'}, V_{\gamma'}, E_{\gamma'} \rangle$ be constraint networks sharing the same set of variables, then γ' is tighter than γ, (write $\gamma' \sqsubseteq \gamma$), if:
  (i) For all $v \in V$: ${D'}_v \subseteq D_v$.
  (ii) For all $u \neq v \in V$ and ${C'}_{uv} \in C'$: either ${C'}_{uv} \notin C$ or ${C'}_{uv} \subseteq C_{uv}$.

  γ' is strictly tighter than γ, (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is proper.
- **Example 2.22.**

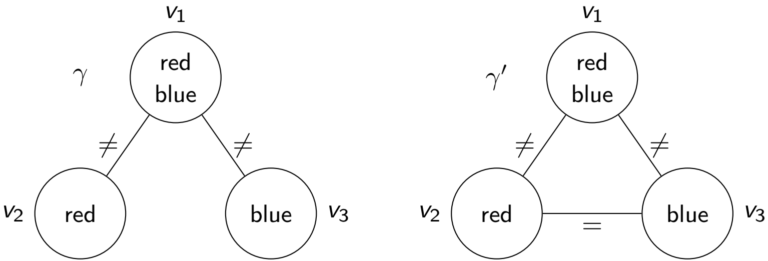


Here, we do not have $\gamma' \sqsubseteq \gamma$!.

- **Intuition:** Strict tightness $\widehat{=}$ "γ' *has the same constraints as* γ, *plus some*".

Equivalence + Tightness = Inference

- **Theorem 2.23.** *Let γ and γ' be constraint networks such that $\gamma' \equiv \gamma$ and $\gamma' \sqsubseteq \gamma$. Then γ' has the same solutions as, but fewer consistent assignments than, γ.*
- $\leadsto \gamma'$ is a better encoding of the underlying problem.
- **Example 2.24.** Two equivalent constraint networks (one obviously unsolvable)



ϵ cannot be extended to a solution (neither in γ nor in γ' because they're equivalent); this is obvious (red $\neq$ blue) in γ', but not in γ.

SOME RIGHTS RESERVED

How to Use Constraint Propagation in CSP Solvers?

- **Simple:** Constraint propagation as a pre-process:
  - **When**: Just once before search starts.
  - **Effect**: Little running time overhead, little pruning power. (not considered here)
- **More Advanced:** Constraint propagation during search:
  - **When**: At every recursive call of backtracking.
  - **Effect**: Strong pruning power, may have large running time overhead.
- **Search vs. Inference:** The more complex the inference, the *smaller* the number of search nodes, but the *larger* the running time needed at each node.
- **Idea:** Encode variable assignments as unary constraints (i.e., for $a(v) = d$, set the unary constraint $D_v = \{d\}$), so that inference reasons about *the network restricted to the commitments already made in the search*.

SOME RIGHTS RESERVED

Backtracking With Inference

- **Definition 2.25.** The general algorithm for backtracking with inference is

```
1  **function** BacktrackingWithInference($\gamma$,$a$) **returns** a solution, or ``inconsistent''
2    **if** $a$ is inconsistent **then return** ``inconsistent''
3    **if** $a$ is a total assignment **then return** $a$
4    $\gamma'$ := a copy of $\gamma$ /* $\gamma' = (V_{\gamma'}, D_{\gamma'}, C_{\gamma'})$ */
5    $\gamma'$ := **Inference**($\gamma'$)
6    **if** exists $v$ with $D_{\gamma'}{}_v = \emptyset$ **then return** ``inconsistent''
7    select some variable $v$ **for** which $a$ is not defined
8    **for** each $d \in$ copy of $D_{\gamma'}{}_v$ **in** some order **do**
9      $a' := a \cup \{v = d\}$; $D_{\gamma'}{}_v := \{d\}$ /* makes $a$ explicit as a constraint */
10     $a''$ := BacktrackingWithInference($\gamma'$,$a'$)
11     **if** $a'' \neq$ "inconsistent" **then return** $a''$
12   **return** ``inconsistent''
```

- Exactly the same as 5.4, only line 5 new!
- **Inference**(): Any procedure delivering a (tighter) equivalent network.
- **Inference**() typically prunes domains; indicate unsolvability by $D_{\gamma'}{}_v = \emptyset$.
- When backtracking out of a search branch, retract the inferred constraints: these were dependent on a, the search commitments so far.

9.3 Forward Checking

Forward Checking

- **Definition 3.1.** Forward checking propagates information about illegal values: Whenever a variable u is assigned by a, delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.

SOME RIGHTS RESERVED

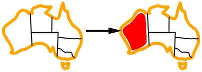
Forward Checking

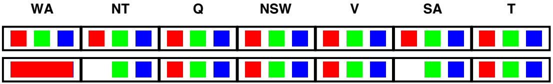
- **Definition 3.4.** Forward checking propagates information about illegal values: Whenever a variable u is assigned by a, delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- **Example 3.5.** Forward checking in Australia

WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦

Forward Checking

- **Definition 3.7.** Forward checking propagates information about illegal values: Whenever a variable u is assigned by a, delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- **Example 3.8.** Forward checking in Australia



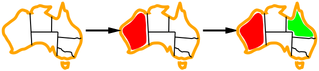


WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

SOME RIGHTS RESERVED

Forward Checking

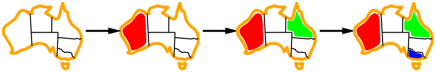
- **Definition 3.10.** Forward checking propagates information about illegal values: Whenever a variable u is assigned by a, delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- **Example 3.11.** Forward checking in Australia

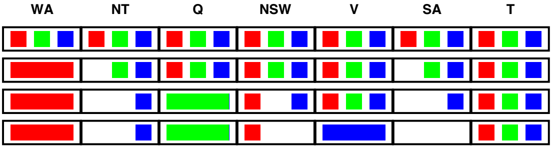


WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦

Forward Checking

- **Definition 3.13.** Forward checking propagates information about illegal values: Whenever a variable u is assigned by a, delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- **Example 3.14.** Forward checking in Australia

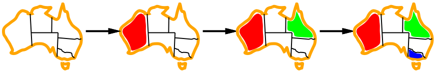


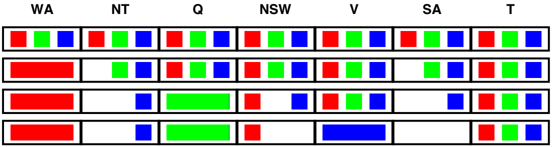


WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

Forward Checking

- **Definition 3.16.** Forward checking propagates information about illegal values: Whenever a variable u is assigned by a, delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- **Example 3.17.** Forward checking in Australia





WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥	🟦		🟥 🟩 🟦

- **Definition 3.18 (Inference, Version 1).** Forward checking implemented

```
function ForwardChecking($\gamma$,$a$) returns modified $\gamma$
  for each $v$ where $a(v) = d'$ is defined do
    for each $u$ where $a(u)$ is undefined and $C_{uv} \in C$ do
      $D_u := \{d \in D_u \mid (d,d') \in C_uv\}$
  return $\gamma$
```

Forward Checking: Discussion

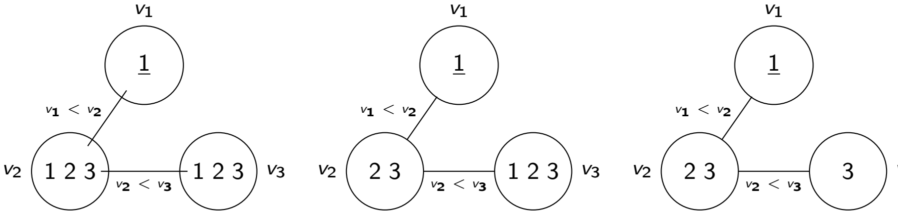
- **Definition 3.19.** An inference procedure is called sound, iff for any input γ the output γ' have the same solutions.
- **Lemma 3.20.** *Forward checking is sound.*
  Proof sketch: Recall here that the assignment a is represented as unary constraints inside γ.
- **Corollary 3.21.** *γ and γ' are equivalent.*
- Incremental computation: Instead of the first for-loop in 3.3, use only the inner one every time a new assignment $a(v) = d'$ is added.
- **Practical Properties:**
  - Cheap but useful inference method.
  - Rarely a good idea to not use forward checking (or a stronger inference method subsuming it).
- **Up next:** A stronger inference method (subsuming forward checking).
- **Definition 3.22.** Let p and q be inference procedures, then p subsumes q, if $p(\gamma) \sqsubseteq q(\gamma)$ for any input γ.

SOME RIGHTS RESERVED

9.4 Arc Consistency

When Forward Checking is Not Good Enough

- **Problem:** Forward checking makes inferences only from assigned to unassigned variables.
- **Example 4.1.**



We could do better here: value 3 for v_2 is not consistent with any remaining value for v_3 $\rightsquigarrow$ it can be removed!
But forward checking does not catch this.

SOME RIGHTS RESERVED

Arc Consistency: Definition

- **Definition 4.2 (Arc Consistency).** Let $\gamma := \langle V, D, C, C, C, V, E\rangle$ be a constraint network.
  1. A variable $u \in V$ is arc consistent relative to another variable $v \in V$ if either $C_{uv} \notin C$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d,d') \in C_{uv}$.
  2. The constraint network γ is arc consistent if every variable $u \in V$ is arc consistent relative to every other variable $v \in V$.

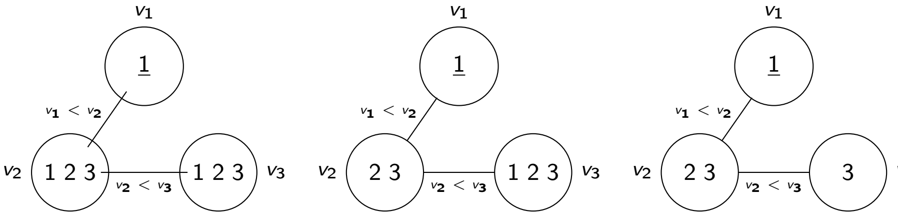
  The concept of arc consistency concerns both levels.
- **Intuition:** Arc consistency $\widehat{=}$ for every domain value and constraint, at least one value on the other side of the constraint "works".
- **Note** the asymmetry between u and v: arc consistency is directed.

SOME RIGHTS RESERVED

Arc Consistency: Example

- **Definition 4.3 (Arc Consistency).** Let $\gamma := \langle V, D, C, C, C, V, E \rangle$ be a constraint network.
  1. A variable $u \in V$ is arc consistent relative to another variable $v \in V$ if either $C_{uv} \notin C$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d,d') \in C_{uv}$.
  2. The constraint network γ is arc consistent if every variable $u \in V$ is arc consistent relative to every other variable $v \in V$.

  The concept of arc consistency concerns both levels.
- **Example 4.4 (Arc Consistency).**



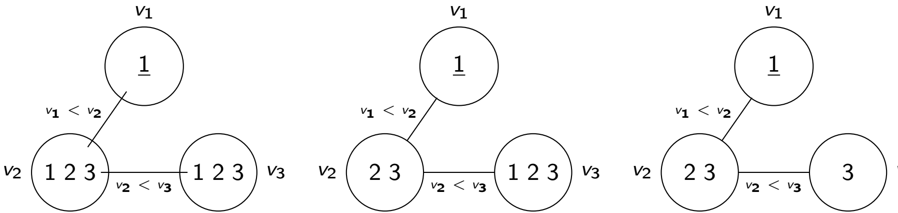
  - **Question**: On top, middle, is v_3 arc consistent relative to v_2?

SOME RIGHTS RESERVED

Arc Consistency: Example

- **Definition 4.5 (Arc Consistency).** Let $\gamma := \langle V, D, C, C, C, V, E \rangle$ be a constraint network.
  1. A variable $u \in V$ is arc consistent relative to another variable $v \in V$ if either $C_{uv} \notin C$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d,d') \in C_{uv}$.
  2. The constraint network γ is arc consistent if every variable $u \in V$ is arc consistent relative to every other variable $v \in V$.

  The concept of arc consistency concerns both levels.
- **Example 4.6 (Arc Consistency).**



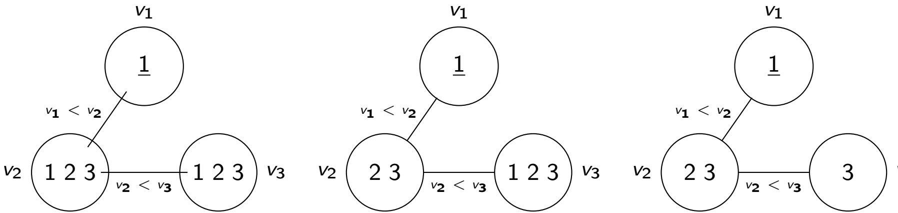
  - **Question**: On top, middle, is v_3 arc consistent relative to v_2?
  - **Answer**: No. For values 1 and 2, D_{v_2} does not have a value that works.
  - **Note**: Enforcing arc consistency for one variable may lead to further reductions on another variable!
  - **Question**: And on the right?

SOME RIGHTS RESERVED

Arc Consistency: Example

- **Definition 4.7 (Arc Consistency).** Let $\gamma := \langle V, D, C, C, C, V, E \rangle$ be a constraint network.
  1. A variable $u \in V$ is arc consistent relative to another variable $v \in V$ if either $C_{uv} \notin C$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d,d') \in C_{uv}$.
  2. The constraint network γ is arc consistent if every variable $u \in V$ is arc consistent relative to every other variable $v \in V$.

  The concept of arc consistency concerns both levels.
- **Example 4.8 (Arc Consistency).**



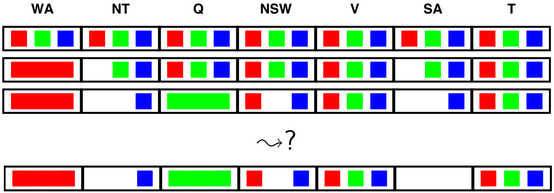
  - **Question**: On top, middle, is v_3 arc consistent relative to v_2?
  - **Answer**: No. For values 1 and 2, D_{v_2} does not have a value that works.
  - **Note**: Enforcing arc consistency for one variable may lead to further reductions on another variable!
  - **Question**: And on the right?

SOME RIGHTS RESERVED

Arc Consistency: Example

- **Definition 4.9 (Arc Consistency).** Let $\gamma := \langle V, D, C, C, C, V, E \rangle$ be a constraint network.
  1. A variable $u \in V$ is arc consistent relative to another variable $v \in V$ if either $C_{uv} \notin C$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d,d') \in C_{uv}$.
  2. The constraint network γ is arc consistent if every variable $u \in V$ is arc consistent relative to every other variable $v \in V$.

  The concept of arc consistency concerns both levels.
- **Example 4.10.**



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

$\rightsquigarrow ?$

🟥	🟦	🟩	🟥🟦	🟥🟩🟦		🟥🟩🟦

- **Note**: SA is not arc consistent relative to NT in 3rd row.

SOME RIGHTS RESERVED

Enforcing Arc Consistency: General Remarks

- **Inference, version 2:** "Enforcing Arc Consistency" = removing domain values until γ is arc consistent. (Up next)
- **Note:** Assuming such an inference method $\mathrm{AC}(\gamma)$.
- **Lemma 4.11.** $\mathrm{AC}(\gamma)$ *is sound: guarantees to deliver an equivalent network.*
- *Proof sketch:* If, for $d \in D_u$, there does not exist a value $d' \in D_v$ such that $(d,d') \in C_{uv}$, then $u = d$ cannot be part of any solution.
- **Observation 4.12.** $\mathrm{AC}(\gamma)$ *subsumes forward checking:* $\mathrm{AC}(\gamma) \sqsubseteq \mathrm{ForwardChecking}(\gamma)$.
- *Proof:* Recall from slide 286 that $\gamma' \sqsubseteq \gamma$ means γ' is tighter than γ.
  1. Forward checking removes d from D_u only if there is a constraint C_{uv} such that $D_v = \{d'\}$ (i.e. when v was assigned the value d'), and $(d,d') \notin C_{uv}$.
  2. Clearly, enforcing arc consistency of u relative to v removes d from D_u as well.

$\square$

SOME RIGHTS RESERVED

Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.13 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

```
function Revise($\gamma$,$u$,$v$) returns modified $\gamma$
  for each $d \in D_u$ do
    if there is no $d' \in D_v$ with $(d,d') \in C_{uv}$ then $D_u := D_u \backslash \{d\}$
  return $\gamma$
```

- **Lemma 4.14.** *If d is maximal domain size in γ and the test "$(d,d') \in C_{uv}$?" has time complexity $\mathcal{O}(1)$, then the running time of $\mathrm{Revise}(\gamma, u, v)$ is $\mathcal{O}(d^2)$.*

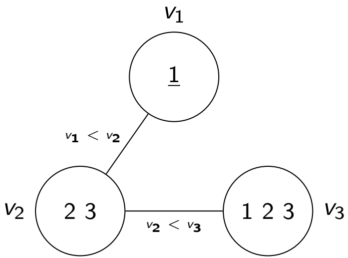
SOME RIGHTS RESERVED

Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.16 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

```
function Revise($\gamma$,$u$,$v$) returns modified $\gamma$
  for each $d \in D_u$ do
    if there is no $d' \in D_v$ with $(d,d') \in C_{uv}$ then $D_u := D_u \backslash \{d\}$
  return $\gamma$
```

- **Lemma 4.17.** *If d is maximal domain size in γ and the test "$(d,d') \in C_{uv}$?" has time complexity $\mathcal{O}(1)$, then the running time of $\mathrm{Revise}(\gamma, u, v)$ is $\mathcal{O}(d^2)$.*
- **Example 4.18.** $\mathrm{Revise}(\gamma, v_3, v_2)$



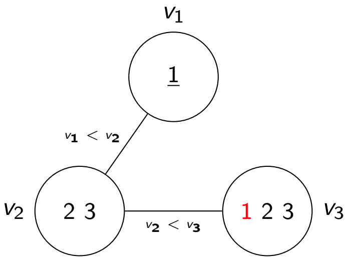
Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.19 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

```
function Revise($\gamma$,$u$,$v$) returns modified $\gamma$
  for each $d \in D_u$ do
    if there is no $d' \in D_v$ with $(d,d') \in C_{uv}$ then $D_u := D_u \backslash \{d\}$
  return $\gamma$
```

- **Lemma 4.20.** *If d is maximal domain size in γ and the test "$(d,d') \in C_{uv}$?" has time complexity $\mathcal{O}(1)$, then the running time of $\mathrm{Revise}(\gamma, u, v)$ is $\mathcal{O}(d^2)$.*

- **Example 4.21.** $\mathrm{Revise}(\gamma, v_3, v_2)$

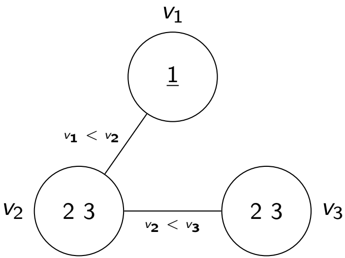


Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.22 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

```
function Revise($\gamma$,$u$,$v$) returns modified $\gamma$
  for each $d \in D_u$ do
    if there is no $d' \in D_v$ with $(d,d') \in C_{uv}$ then $D_u := D_u \backslash \{d\}$
  return $\gamma$
```

- **Lemma 4.23.** *If d is maximal domain size in γ and the test "$(d,d') \in C_{uv}$?" has time complexity $\mathcal{O}(1)$, then the running time of $\mathrm{Revise}(\gamma, u, v)$ is $\mathcal{O}(d^2)$.*
- **Example 4.24.** $\mathrm{Revise}(\gamma, v_3, v_2)$

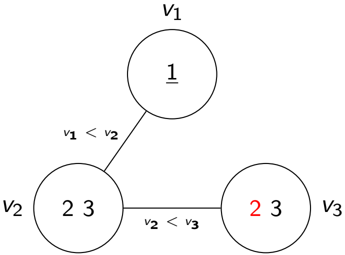


Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.25 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

```
function Revise($\gamma$,$u$,$v$) returns modified $\gamma$
  for each $d \in D_u$ do
    if there is no $d' \in D_v$ with $(d,d') \in C_{uv}$ then $D_u := D_u \backslash \{d\}$
  return $\gamma$
```

- **Lemma 4.26.** *If d is maximal domain size in γ and the test "$(d,d') \in C_{uv}$?" has time complexity $\mathcal{O}(1)$, then the running time of $\mathrm{Revise}(\gamma, u, v)$ is $\mathcal{O}(d^2)$.*
- **Example 4.27.** $\mathrm{Revise}(\gamma, v_3, v_2)$

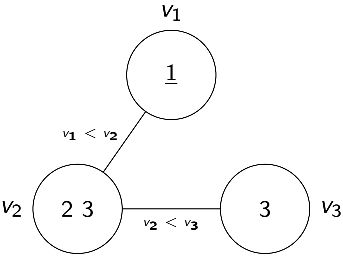


Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.28 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

```
function Revise($\gamma$,$u$,$v$) returns modified $\gamma$
  for each $d \in D_u$ do
    if there is no $d' \in D_v$ with $(d,d') \in C_{uv}$ then $D_u := D_u \backslash \{d\}$
  return $\gamma$
```

- **Lemma 4.29.** *If d is maximal domain size in γ and the test "$(d,d') \in C_{uv}$?" has time complexity $\mathcal{O}(1)$, then the running time of $\mathrm{Revise}(\gamma, u, v)$ is $\mathcal{O}(d^2)$.*
- **Example 4.30.** $\mathrm{Revise}(\gamma, v_3, v_2)$

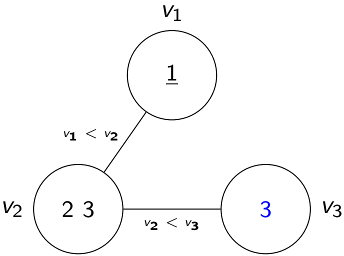


Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.31 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

```
function Revise($\gamma$,$u$,$v$) returns modified $\gamma$
  for each $d \in D_u$ do
    if there is no $d' \in D_v$ with $(d,d') \in C_{uv}$ then $D_u := D_u \backslash \{d\}$
  return $\gamma$
```

- **Lemma 4.32.** *If d is maximal domain size in γ and the test "$(d,d') \in C_{uv}$?" has time complexity $\mathcal{O}(1)$, then the running time of $\mathrm{Revise}(\gamma, u, v)$ is $\mathcal{O}(d^2)$.*
- **Example 4.33.** $\mathrm{Revise}(\gamma, v_3, v_2)$



AC-1: Enforcing Arc Consistency (Version 1)

- **Idea:** Apply Revise pairwise up to a fixed point.
- **Definition 4.34.** AC-1 enforces arc consistency in constraint networks:

```
function AC−1($\gamma$) returns modified $\gamma$
  repeat
    changesMade := False
    for each constraint $C_{uv}$ do
      Revise($\gamma$,$u$,$v$) /∗ if $D_u$ reduces, set changesMade := True ∗/
      Revise($\gamma$,$v$,$u$) /∗ if $D_v$ reduces, set changesMade := True ∗/
  until changesMade = False
  return $\gamma$
```

SOME RIGHTS RESERVED

AC-1: Enforcing Arc Consistency (Version 1)

- **Idea:** Apply Revise pairwise up to a fixed point.
- **Definition 4.36.** AC-1 enforces arc consistency in constraint networks:

```
function AC−1($\gamma$) returns modified $\gamma$
  repeat
    changesMade := False
    for each constraint $C_{uv}$ do
      Revise($\gamma$,$u$,$v$) /* if $D_u$ reduces, set changesMade := True */
      Revise($\gamma$,$v$,$u$) /* if $D_v$ reduces, set changesMade := True */
  until changesMade = False
  return $\gamma$
```

- **Observation:** Obviously, this does indeed enforce arc consistency for γ.
- **Lemma 4.37.** *If γ has n variables, m constraints, and maximal domain size d, then the time complexity of $\mathrm{AC1}(\gamma)$ is $\mathcal{O}(md^2nd)$.*
- *Proof sketch:* $\mathcal{O}(md^2)$ for each inner loop, fixed point reached at the latest once all nd variable values have been removed.

SOME RIGHTS RESERVED

AC-1: Enforcing Arc Consistency (Version 1)

- **Idea:** Apply Revise pairwise up to a fixed point.
- **Definition 4.38.** AC-1 enforces arc consistency in constraint networks:

```
function AC−1($\gamma$) returns modified $\gamma$
  repeat
    changesMade := False
    for each constraint $C_{uv}$ do
      Revise($\gamma$,$u$,$v$) /* if $D_u$ reduces, set changesMade := True */
      Revise($\gamma$,$v$,$u$) /* if $D_v$ reduces, set changesMade := True */
  until changesMade = False
  return $\gamma$
```

- **Observation:** Obviously, this does indeed enforce arc consistency for γ.
- **Lemma 4.39.** *If γ has n variables, m constraints, and maximal domain size d, then the time complexity of $\mathrm{AC1}(\gamma)$ is $\mathcal{O}(md^2nd)$.*
- *Proof sketch:* $\mathcal{O}(md^2)$ for each inner loop, fixed point reached at the latest once all nd variable values have been removed.
- **Problem:** There are redundant computations.
- **Question:** Do you see what these redundant computations are?

SOME RIGHTS RESERVED

AC-1: Enforcing Arc Consistency (Version 1)

- **Idea:** Apply Revise pairwise up to a fixed point.
- **Definition 4.40.** AC-1 enforces arc consistency in constraint networks:

```
function AC−1($\gamma$) returns modified $\gamma$
  repeat
    changesMade := False
    for each constraint $C_{uv}$ do
      Revise($\gamma$,$u$,$v$) /∗ if $D_u$ reduces, set changesMade := True ∗/
      Revise($\gamma$,$v$,$u$) /∗ if $D_v$ reduces, set changesMade := True ∗/
  until changesMade = False
  return $\gamma$
```

- **Observation:** Obviously, this does indeed enforce arc consistency for γ.
- **Lemma 4.41.** *If γ has n variables, m constraints, and maximal domain size d, then the time complexity of $\mathrm{AC1}(\gamma)$ is $\mathcal{O}(md^2nd)$.*
- *Proof sketch:* $\mathcal{O}(md^2)$ for each inner loop, fixed point reached at the latest once all nd variable values have been removed.
- **Problem:** There are redundant computations.
- **Question:** Do you see what these redundant computations are?
- **Redundant computations:** u and v are revised even if theirdomains haven't changed since the last time.
- Better algorithm avoiding this: AC 3 (coming up)

SOME RIGHTS RESERVED

AC-3: Enforcing Arc Consistency (Version 3)

- **Idea:** Remember the potentially inconsistent variable pairs.
- **Definition 4.42.** AC-3 optimizes AC-1 for enforcing arc consistency.

```
function AC−3($\gamma$) returns modified $\gamma$
  $M := \emptyset$
  for each constraint $C_{uv} \in C$ do
    $M := M \cup \{(u,v),(v,u)\}$
  while $M \neq \emptyset$ do
    remove any element $(u,v)$ from $M$
    Revise($\gamma, u, v$)
    if $D_u$ has changed in the call to Revise then
      for each constraint $C_{wu} \in C$ where $w \neq v$ do
        $M := M \cup \{(w,u)\}$
  return $\gamma$
```

- **Question:** $\mathrm{AC}-3(\gamma)$ enforces arc consistency because?

SOME RIGHTS RESERVED

AC-3: Enforcing Arc Consistency (Version 3)

- **Idea:** Remember the potentially inconsistent variable pairs.
- **Definition 4.43.** AC-3 optimizes AC-1 for enforcing arc consistency.

```
function AC−3($\gamma$) returns modified $\gamma$
  $M := \emptyset$
  for each constraint $C_{uv} \in C$ do
    $M := M \cup \{(u,v),(v,u)\}$
  while $M \neq \emptyset$ do
    remove any element $(u,v)$ from $M$
    Revise($\gamma, u, v$)
    if $D_u$ has changed in the call to Revise then
      for each constraint $C_{wu} \in C$ where $w \neq v$ do
        $M := M \cup \{(w,u)\}$
  return $\gamma$
```

- **Question:** $\mathrm{AC}-3(\gamma)$ enforces arc consistency because?
- **Answer:** At any time during the while-loop, if $(u,v) \notin M$ then u is arc consistent relative to v.
- **Question:** Why only "where $w \neq v$"?

SOME RIGHTS RESERVED

AC-3: Enforcing Arc Consistency (Version 3)

- **Idea:** Remember the potentially inconsistent variable pairs.
- **Definition 4.44.** AC-3 optimizes AC-1 for enforcing arc consistency.

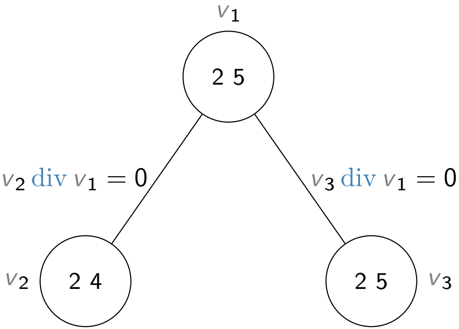
```
function AC−3($\gamma$) returns modified $\gamma$
  $M := \emptyset$
  for each constraint $C_{uv} \in C$ do
    $M := M \cup \{(u,v),(v,u)\}$
  while $M \neq \emptyset$ do
    remove any element $(u,v)$ from $M$
    Revise($\gamma, u, v$)
    if $D_u$ has changed in the call to Revise then
      for each constraint $C_{wu} \in C$ where $w \neq v$ do
        $M := M \cup \{(w,u)\}$
  return $\gamma$
```

- **Question:** $\mathrm{AC}-3(\gamma)$ enforces arc consistency because?
- **Answer:** At any time during the while-loop, if $(u,v) \notin M$ then u is arc consistent relative to v.
- **Question:** Why only "where $w \neq v$"?
- **Answer:** If $w = v$ is the reason why D_u changed, then w is still arc consistent relative to u: the values just removed from D_u did not match any values from D_w anyway.

SOME RIGHTS RESERVED

AC-3: Example

- **Example 4.45.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



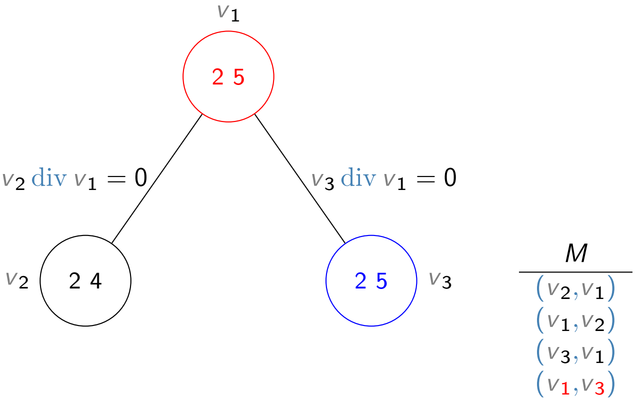
v_1
2 5
$v_2 \operatorname{div} v_1 = 0$
$v_3 \operatorname{div} v_1 = 0$
v_2
2 4
2 5
v_3

M
(v_2, v_1)
(v_1, v_2)
(v_3, v_1)
(v_1, v_3)

SOME RIGHTS RESERVED

AC-3: Example

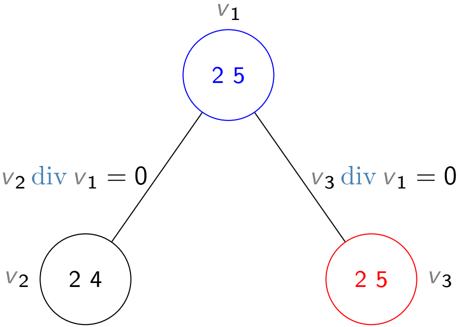
- **Example 4.46.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



SOME RIGHTS RESERVED

AC-3: Example

- **Example 4.47.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x

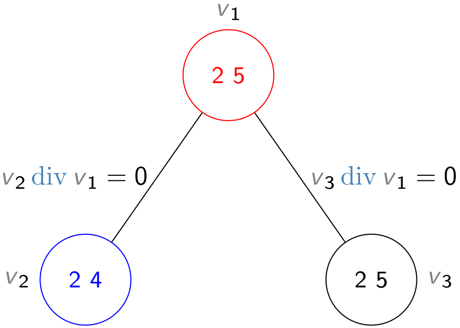

v_1
2 5
$v_2 \operatorname{div} v_1 = 0$
$v_3 \operatorname{div} v_1 = 0$
v_2
2 4
2 5
v_3

M

(v_2, v_1)
(v_1, v_2)
(v_3, v_1)

AC-3: Example

- **Example 4.48.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x


v_1
2 5
$v_2 \operatorname{div} v_1 = 0$
$v_3 \operatorname{div} v_1 = 0$
v_2
2 4
2 5
v_3

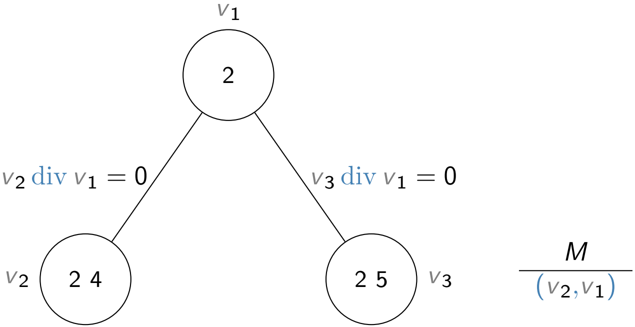
M

(v_2, v_1)
(v_1, v_2)

SOME RIGHTS RESERVED

AC-3: Example

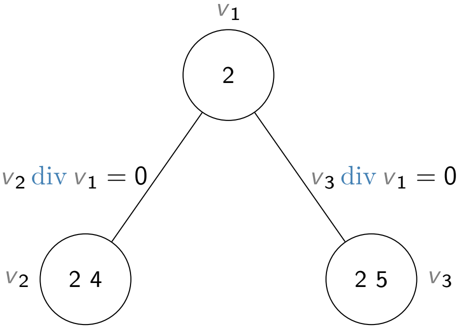
- **Example 4.49.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x


v_1
2
$v_2 \operatorname{div} v_1 = 0$
$v_3 \operatorname{div} v_1 = 0$
v_2
2 4
2 5
v_3
M
(v_2,v_1)

SOME RIGHTS RESERVED

AC-3: Example

- **Example 4.50.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x

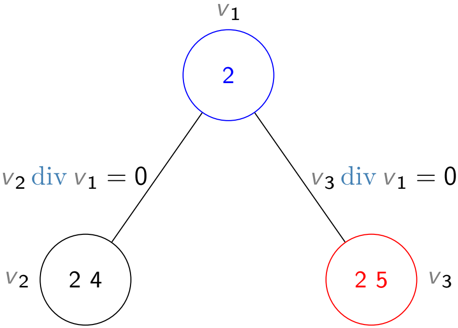

v_1
2
$v_2 \operatorname{div} v_1 = 0$
$v_3 \operatorname{div} v_1 = 0$
v_2
2 4
2 5
v_3

M
(v_2,v_1)
(v_3,v_1)

SOME RIGHTS RESERVED

AC-3: Example

- **Example 4.51.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x


v_1
2
$v_2 \operatorname{div} v_1 = 0$
$v_3 \operatorname{div} v_1 = 0$
v_2
2 4
2 5
v_3

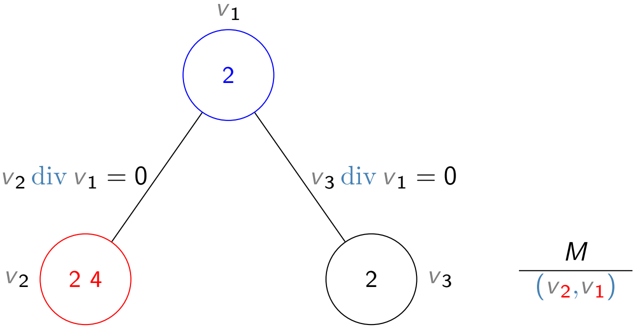
M

(v_2, v_1)

(v_3, v_1)

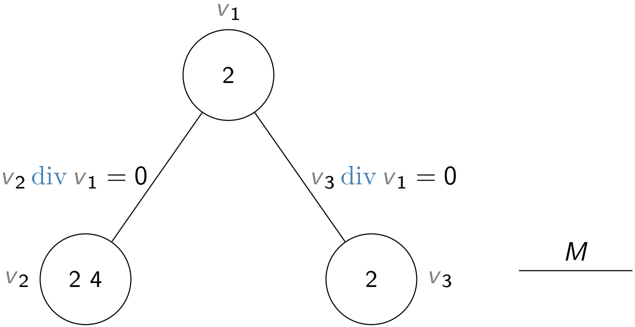
AC-3: Example

- **Example 4.52.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x


v_1
2
$v_2 \operatorname{div} v_1 = 0$
$v_3 \operatorname{div} v_1 = 0$
v_2
2 4
2
v_3
M
(v_2,v_1)

AC-3: Example

- **Example 4.53.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x


v_1
2
$v_2 \operatorname{div} v_1 = 0$
$v_3 \operatorname{div} v_1 = 0$
v_2
2 4
2
v_3
M

SOME RIGHTS RESERVED

AC-3: Runtime

- **Theorem 4.54 (Runtime of AC-3).** *Let* $\gamma := \langle V, D, C, C, C, V, E\rangle$ *be a constraint network with* m *constraints, and maximal domain size* d. *Then* $\mathrm{AC}-3(\gamma)$ *runs in time* $\mathcal{O}(md^3)$.
- *Proof:* by counting how often Revise is called.
  1. Each call to $\mathrm{Revise}(\gamma, u, v)$ takes time $\mathcal{O}(d^2)$ so it suffices to prove that at most $\mathcal{O}(md)$ of these calls are made.
  2. The number of calls to $\mathrm{Revise}(\gamma, u, v)$ is the number of iterations of the while-loop, which is at most the number of insertions into M.
  3. Consider any constraint C_{uv}.
  4. Two variable pairs corresponding to C_{uv} are inserted in the for-loop. In the while loop, if a pair corresponding to C_{uv} is inserted into M, then
  5. beforehand the domain of either u or v was reduced, which happens at most $2d$ times.
  6. Thus we have $\mathcal{O}(d)$ insertions per constraint, and $\mathcal{O}(md)$ insertions overall, as desired.

□

SOME RIGHTS RESERVED

9.5 Decomposition: Constraint Graphs, and Three Simple Cases

SOME RIGHTS RESERVED

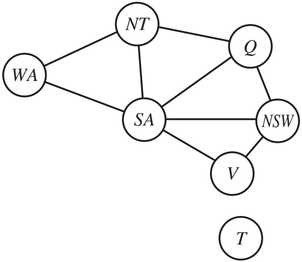
Reminder: The Big Picture

- Say γ is a constraint network with n variables and maximal domain size d.
  - d^n total assignments must be tested in the worst case to solve γ.
- **Inference:** One method to try to avoid/ameliorate this combinatorial explosion in practice.
  - Often, from an assignment to some variables, we can easily make inferences regarding other variables.
- **Decomposition:** Another method to avoid/ameliorate this combinatorial explosion in practice.
  - Often, we can exploit the *structure* of a network to *decompose* it into smaller parts that are easier to solve.
  - **Question**: What is "structure", and how to "decompose"?

SOME RIGHTS RESERVED

Problem Structure

- **Idea:** Tasmania and mainland are "independent subproblems"
- **Definition 5.1.** Independent subproblems are identified as connected components of constraint graphs.
- Suppose each independent subproblem has c variables out of n total. (d is max domain size)
- Worst-case solution cost is $n \operatorname{div} c \cdot d^c$ (linear in n)
- E.g., $n = 80$, $d = 2$, $c = 20$
  - $2^{80} \widehat{=}$ 4 billion years at 10 million nodes/sec
  - $4 \cdot 2^{20} \widehat{=}$ 0.4 seconds at 10 million nodes/sec



"Decomposition" 1.0: Disconnected Constraint Graphs

- **Theorem 5.2 (Disconnected Constraint Graphs).** *Let* $\gamma := \langle V, D, C, C, C, V, E\rangle$ *be a constraint network. Let* a_i *be a solution to each connected component* γ_i *of the constraint graph of* γ*. Then* $a := \bigcup_i a_i$ *is a solution to* γ*.*

SOME RIGHTS RESERVED

“Decomposition” 1.0: Disconnected Constraint Graphs

- **Theorem 5.6 (Disconnected Constraint Graphs).** *Let* $\gamma := \langle V, D, C, C, C, V, E \rangle$ *be a constraint network. Let* a_i *be a solution to each connected component* γ_i *of the constraint graph of* γ*. Then* $a := \bigcup_i a_i$ *is a solution to* γ*.*
- *Proof:*
  1. a satisfies all C_{uv} where u and v are inside the same connected component.
  2. The latter is the case for all C_{uv}.
  3. If two parts of γ are not connected, then they are independent.

☐

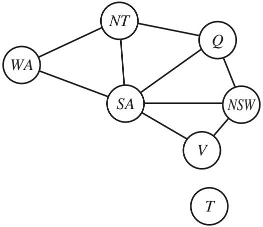
SOME RIGHTS RESERVED

"Decomposition" 1.0: Disconnected Constraint Graphs

- **Theorem 5.10 (Disconnected Constraint Graphs).** *Let $\gamma := \langle V, D, C, C, C, V, E\rangle$ be a constraint network. Let a_i be a solution to each connected component γ_i of the constraint graph of γ. Then $a := \bigcup_i a_i$ is a solution to γ.*
- *Proof:*
  1. a satisfies all C_{uv} where u and v are inside the same connected component.
  2. The latter is the case for all C_{uv}.
  3. If two parts of γ are not connected, then they are independent.

☐

- **Example 5.11.** Color Tasmania separately in Australia



SOME RIGHTS RESERVED

“Decomposition” 1.0: Disconnected Constraint Graphs

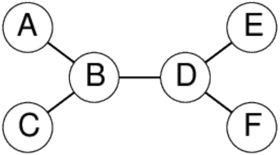
- **Theorem 5.14 (Disconnected Constraint Graphs).** *Let* $\gamma := \langle V, D, C, C, C, V, E \rangle$ *be a constraint network. Let* a_i *be a solution to each connected component* γ_i *of the constraint graph of* γ*. Then* $a := \bigcup_i a_i$ *is a solution to* γ*.*
- *Proof:*
  1. a satisfies all C_{uv} where u and v are inside the same connected component.
  2. The latter is the case for all C_{uv}.
  3. If two parts of γ are not connected, then they are independent.

  ☐

- **Example 5.15.** Color Tasmania separately in Australia
- **Example 5.16 (Doing the Numbers).**
  - γ with $n = 40$ variables, each domain size $k = 2$. Four separate connected components each of size 10.
  - Reduction of worst-case when using decomposition:
    - No decomposition: 2^{40}. With: $4 \cdot 2^{10}$. Gain: $2^{28} \approxeq 280.000.000$.
- **Definition 5.17.** The process of decomposing a constraint network into components is called decomposition. There are various decomposition algorithms.

SOME RIGHTS RESERVED

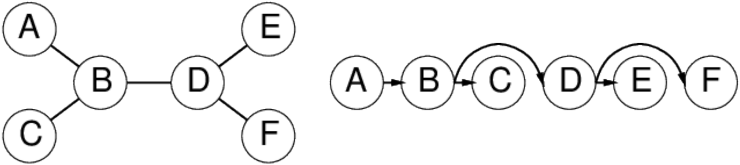
Tree-structured CSPs



- **Definition 5.18.** We call a CSP tree-structured, iff its constraint graph is acyclic
- **Theorem 5.19.** *Tree-structured CSP can be solved in $\mathcal{O}(nd^2)$ time.*
- Compare to general CSPs, where worst case time is $\mathcal{O}(d^n)$.
- This property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and the complexity of reasoning.

Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



2. For j from n down to 2, apply

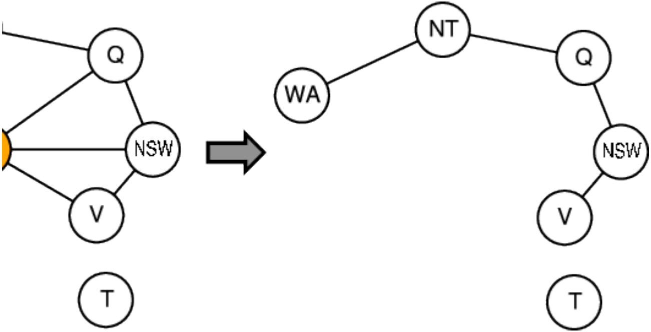
   RemoveInconsistent(Parent(X_j,X_j))

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

SOME RIGHTS RESERVED

Nearly tree-structured CSPs

- **Definition 5.20.** Conditioning: instantiate a variable, prune its neighbors' domains.
- **Example 5.21.**



- **Definition 5.22.** Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree.
- Cutset size $c \rightsquigarrow$ running time $\mathcal{O}(d^c(n-c)d^2)$, very fast for small c.

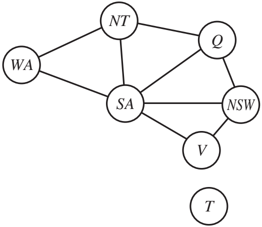
SOME RIGHTS RESERVED

"Decomposition" 2.0: Acyclic Constraint Graphs

- **Theorem 5.23 (Acyclic Constraint Graphs).** *Let* $\gamma := \langle V, D, C, C, C, V, E\rangle$ *be a constraint network with* n *variables and maximal domain size* k*, whose constraint graph is acyclic. Then we can find a solution for* γ*, or prove* γ *to be unsatisfiable, in time* $\mathcal{O}(nk^2)$*.*
- *Proof sketch:* See the algorithm on the next slide
- Constraint networks with acyclic constraint graphs can be solved in (low order) polynomial time.

"Decomposition" 2.0: Acyclic Constraint Graphs

- **Theorem 5.26 (Acyclic Constraint Graphs).** *Let* $\gamma := \langle V, D, C, C, C, V, E\rangle$ *be a constraint network with* n *variables and maximal domain size* k*, whose constraint graph is acyclic. Then we can find a solution for* γ*, or prove* γ *to be unsatisfiable, in time* $\mathcal{O}(nk^2)$*.*
- *Proof sketch:* See the algorithm on the next slide
- Constraint networks with acyclic constraint graphs can be solved in (low order) polynomial time.
- **Example 5.27.** Australia is not acyclic. (But see next section)



"Decomposition" 2.0: Acyclic Constraint Graphs

- **Theorem 5.29 (Acyclic Constraint Graphs).** *Let $\gamma := \langle V, D, C, C, C, V, E \rangle$ be a constraint network with n variables and maximal domain size k, whose constraint graph is acyclic. Then we can find a solution for γ, or prove γ to be unsatisfiable, in time $\mathcal{O}(nk^2)$.*
- *Proof sketch:* See the algorithm on the next slide
- Constraint networks with acyclic constraint graphs can be solved in (low order) polynomial time.
- **Example 5.30.** Australia is not acyclic. (But see next section)
- **Example 5.31 (Doing the Numbers).**
  - γ with $n = 40$ variables, each domain size $k = 2$. Acyclic constraint graph.
  - Reduction of worst-case when using decomposition:
    - No decomposition: 2^{40}.
    - With decomposition: $40 \cdot 2^2$. Gain: 2^{32}.

SOME RIGHTS RESERVED

Acyclic Constraint Graphs: How To

- **Definition 5.32.** Algorithm $\mathrm{AcyclicCG}(\gamma)$:
  1. Obtain a (directed) tree from γ's constraint graph, picking an arbitrary variable *v* as the root, and directing edges outwards.[1]

---

[1] We assume here that γ's constraint graph is connected. If it is not, do this and the following for each component separately.

SOME RIGHTS RESERVED

Acyclic Constraint Graphs: How To

- **Definition 5.34.** Algorithm $\mathrm{AcyclicCG}(\gamma)$:
  1. Obtain a (directed) tree from γ's constraint graph, picking an arbitrary variable v as the root, and directing edges outwards.[1]
  2. Order the variables topologically, i.e., such that each node is ordered before its children; denote that order by $v_1, \ldots, v_n$.

---

[1] We assume here that γ's constraint graph is connected. If it is not, do this and the following for each component separately.

SOME RIGHTS RESERVED

Acyclic Constraint Graphs: How To

- **Definition 5.36.** Algorithm $\mathrm{AcyclicCG}(\gamma)$:
  1. Obtain a (directed) tree from γ's constraint graph, picking an arbitrary variable v as the root, and directing edges outwards.[1]
  2. Order the variables topologically, i.e., such that each node is ordered before its children; denote that order by $v_1, \ldots, v_n$.
  3. **for** $i := n, n-1, \ldots, 2$ **do**:
     - 3.1 $\mathrm{Revise}(\gamma, v_{parent(i)}, v_i)$.
     - 3.2 **if** $D_{v_{parent(i)}} = \emptyset$ **then return** "inconsistent"

     Now, every variable is arc consistent relative to its children.
  4. Run BacktrackingWithInference with forward checking, using the variable order $v_1, \ldots, v_n$.
- **Lemma 5.37.** *This algorithm will find a solution without ever having to backtrack!*

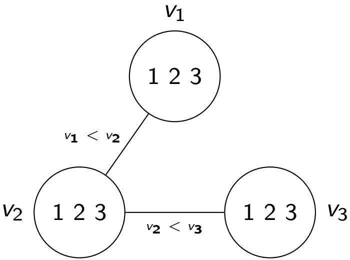
---

[1] We assume here that γ's constraint graph is connected. If it is not, do this and the following for each component separately.

SOME RIGHTS RESERVED

AcyclicCG(γ): Example

- **Example 5.38 (AcyclicCG() execution).**

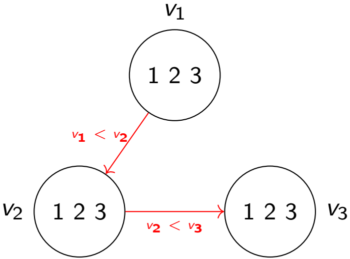


Input network γ.

SOME RIGHTS RESERVED

AcyclicCG(γ): Example

- **Example 5.39 (AcyclicCG() execution).**

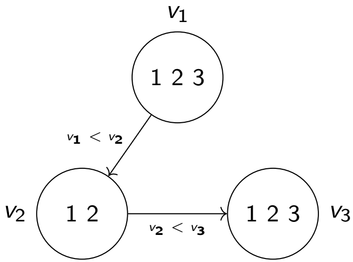


Step 1: Directed tree for root v_1.
Step 2: Order v_1, v_2, v_3.

SOME RIGHTS RESERVED

AcyclicCG(γ): Example

- **Example 5.40 (AcyclicCG() execution).**

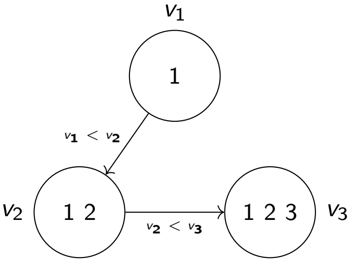


Step 3: After Revise(γ, v_2, v_3).

SOME RIGHTS RESERVED

AcyclicCG(γ): Example

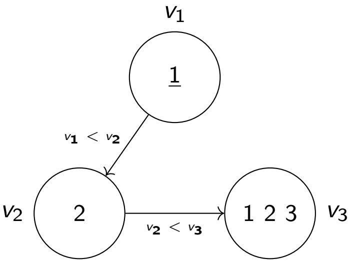
- **Example 5.41 (AcyclicCG() execution).**



Step 3: After Revise(γ, v_1, v_2).

AcyclicCG(γ): Example

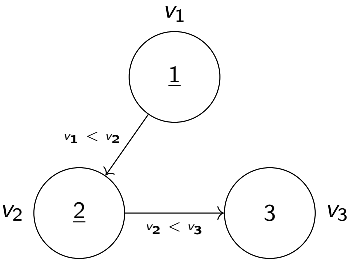
- **Example 5.42 (AcyclicCG() execution).**



Step 4: After $a(v_1) := 1$ and forward checking.

AcyclicCG(γ): Example

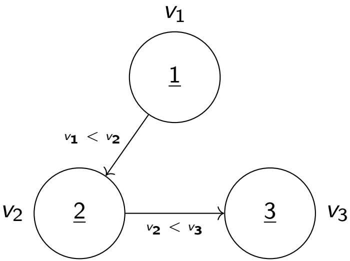
- **Example 5.43 (AcyclicCG() execution).**



Step 4: After $a(v_2) := 2$ and forward checking.

AcyclicCG(γ): Example

- **Example 5.44 (AcyclicCG() execution).**



Step 4: After $a(v_3) := 3$ (and forward checking).

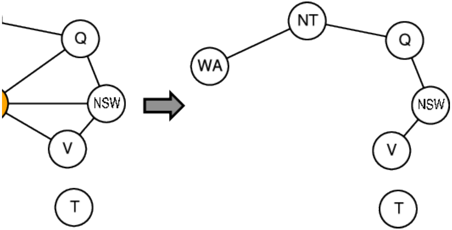
SOME RIGHTS RESERVED

9.6 Cutset Conditioning

SOME RIGHTS RESERVED

"Almost" Acyclic Constraint Graphs

- **Example 6.1 (Coloring Australia).**



- **Cutset Conditioning: Idea:**
  1. Recursive call of backtracking search on a s.t. the subgraph of the constraint graph induced by $\{v \in V \mid a(v) \text{ is undefined}\}$ is acyclic.
     - Then we can solve the remaining sub-problem with $\mathrm{AcyclicCG}()$.
  2. Choose the variable ordering so that removing the first d variables renders the constraint graph acyclic.
     - Then with (1) we won't have to search deeper than d . . . !

SOME RIGHTS RESERVED

"Decomposition" 3.0: Cutset Conditioning

- **Definition 6.2 (Cutset).** Let $\gamma := \langle V, D, C, C, C, V, E\rangle$ be a constraint network, and $V_0 \subseteq V$. Then V_0 is a cutset for γ if the subgraph of γ's constraint graph induced by $V \backslash V_0$ is acyclic. V_0 is called optimal if its size is minimal among all cutsets for γ.
- **Definition 6.3.** The cutset conditioning algorithm, computes an optimal cutset, from γ and an existing cutset V_0.

```
function CutsetConditioning($\gamma$,$V_0$,$a$) returns a solution, or ``inconsistent''
  $\gamma'$ := a copy of $\gamma$; $\gamma'$ := ForwardChecking($\gamma'$,$a$)
  if ex. $v$ with $D_{\gamma'_v} = \emptyset$ then return ``inconsistent''
  if ex. $v \in V_0$ s.t. $a(v)$ is undefined then select such $v$
  else $a'$ := AcyclicCG($\gamma'$);
  if $a' \neq$ "inconsistent" then return $a \cup a'$ else return ``inconsistent''
  for each $d \in$ copy of $D_{\gamma'_v}$ in some order do
    $a'$ := $a \cup \{v = d\}$; $D_{\gamma'_v}$ := $\{d\}$;
    $a''$ := CutsetConditioning($\gamma'$,$V_0$,$a'$)
  if $a'' \neq$ "inconsistent" then return $a''$ else return ``inconsistent''
```

- Forward checking is required so that "$a \cup \mathrm{AcyclicCG}(\gamma')$" is consistent in γ.
- **Observation 6.4.** *Running time is exponential only in $\#(V_0)$, not in $\#(V)$!*
- *Remark 6.5.* Finding optimal cutsets is NP hard, but good approximations exist.

SOME RIGHTS RESERVED

9.7 Constraint Propagation with Local Search

SOME RIGHTS RESERVED

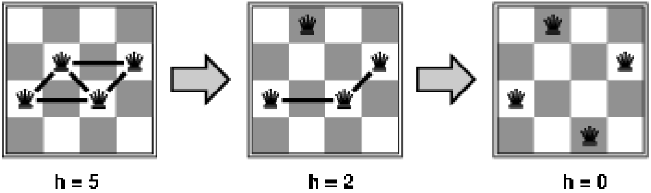
Iterative algorithms for CSPs

- Local search algorithms like hill climbing and simulated annealing typically work with “complete” states, i.e., all variables are assigned
- To apply to CSPs: allow states with unsatisfied constraints, actions reassign variable values.
- **Variable selection:** Randomly select any conflicted variable.
- **Value selection** by min conflicts heuristic: choose value that violates the fewest constraints i.e., hill climb with $h(n)$:=total number of violated constraints.

SOME RIGHTS RESERVED

Example: 4-Queens

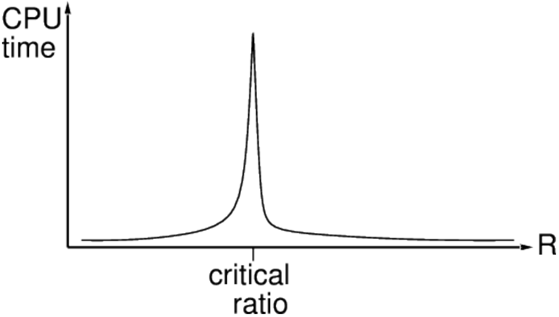
- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Actions: Move queen in column
- Goal state: No conflicts
- Heuristic: $h(n) \mathrel{\widehat{=}}$ number of conflict



Performance of min-conflicts

- Given a random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10{,}000{,}000$)
- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



SOME RIGHTS RESERVED

9.8 Conclusion & Summary

SOME RIGHTS RESERVED

Conclusion & Summary

- γ and γ' are equivalent if they have the same solutions. γ' is tighter than γ if it is more constrained.
- Inference tightens γ without losing equivalence, during backtracking search. This reduces the amount of search needed; that benefit must be traded off against the running time overhead for making the inferences.
- Forward checking removes values conflicting with an assignment already made.
- Arc consistency removes values that do not comply with any value still available at the other end of a constraint. This subsumes forward checking.
- The constraint graph captures the dependencies between variables. Separate connected components can be solved independently. Networks with acyclic constraint graphs can be solved in low order polynomial time.
- A cutset is a subset of variables removing which renders the constraint graph acyclic. Cutset conditioning backtracks only on such a cutset, and solves a sub-problem with acyclic constraint graph at each search leaf.

SOME RIGHTS RESERVED

Topics We Didn't Cover Here

- **Path consistency, *k*-consistency:** Generalizes arc consistency to size k subsets of variables. Path consistency $\widehat{=}$ 3-consistency.
- **Tree decomposition:** Instead of instantiating variables until the leaf nodes are trees, distribute the variables and constraints over sub-CSPs whose connections form a tree.
- **Backjumping:** Like backtracking search, but with ability to back up *across several levels* (to a previous variable assignment identified to be responsible for failure).
- **No-Good Learning:** Inferring additional constraints based on information gathered during backtracking search.
- **Local search:** In space of total (but not necessarily consistent) assignments. (E.g., 8 queens in)
- **Tractable CSP:** Classes of CSPs that can be solved in P.
- **Global Constraints:** Constraints over many/all variables, with associated specialized inference methods.
- **Constraint Optimization Problems (COP):** Utility function over solutions, need an optimal one.

SOME RIGHTS RESERVED

Part 3

Knowledge and Inference

SOME RIGHTS RESERVED

Chapter 10
Propositional Logic & Reasoning, Part I: Principles

SOME RIGHTS RESERVED

10.1 Introduction: Inference with Structured State Representations

State Representations in Agents and Algorithms

- **Recall:** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attribute and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.

SOME RIGHTS RESERVED

State Representations in Agents and Algorithms

- **Recall:** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attribute and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.
- **Recall:** We have used atomic representations in search problems and tree search algorithms.
- **But:** We already allowed peeking into state in
  - informed search to compute heuristics
  - adversarial search $\leftsquigarrow$ too many state!

SOME RIGHTS RESERVED

State Representations in Agents and Algorithms

- **Recall:** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attribute and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.
- **Recall:** We have used atomic representations in search problems and tree search algorithms.
- **But:** We already allowed peeking into state in
  - informed search to compute heuristics
  - adversarial search ⇜ too many state!
- **Recall:** We have used factored representations in
  - backtracking search for CSPs ⇝ universally useful heuristics
  - constraint propagation: inference ⇝ lifting search to the CSP description level.

SOME RIGHTS RESERVED

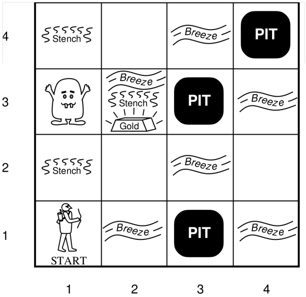
State Representations in Agents and Algorithms

- **Recall:** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attribute and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.
- **Recall:** We have used atomic representations in search problems and tree search algorithms.
- **But:** We already allowed peeking into state in
  - informed search to compute heuristics
  - adversarial search ⇜ too many state!
- **Recall:** We have used factored representations in
  - backtracking search for CSPs ⇝ universally useful heuristics
  - constraint propagation: inference ⇝ lifting search to the CSP description level.
- **Up Next:** Inference for structured state representations.

10.1.1 A Running Example: The Wumpus World

SOME RIGHTS RESERVED

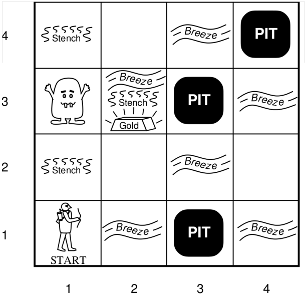
The Wumpus World



Definition 1.1. The Wumpus world is a simple game where an agent explores a cave with 16 cells that can contain pits, gold, and the Wumpus with the goal of getting back out alive with the gold.
The agent cannot observe the locations of pits, gold, and the Wumpus, but some of their effects in the cell it currently visits.

- **Definition 1.2 (Actions).** The agent can perform the following actions: goForward, turnRight (by 90°), turnLeft (by 90°), shoot arrow in direction you're facing (you got exactly one arrow), grab an object in current cell, leave cave if you're in cell $[1,1]$.
- **Definition 1.3 (Initial and Terminal States).** Initially, the agent is in cell $[1,1]$ facing east. If the agent falls down a pit or meets live Wumpus it dies.

The Wumpus World



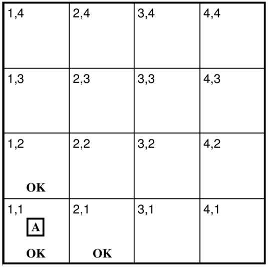
Definition 1.5. The Wumpus world is a simple game where an agent explores a cave with 16 cells that can contain pits, gold, and the Wumpus with the goal of getting back out alive with the gold.
The agent cannot observe the locations of pits, gold, and the Wumpus, but some of their effects in the cell it currently visits.

- **Definition 1.8 (Percepts).** The agent can experience the following percepts: stench, breeze, glitter, bump, scream, none.
  - Cell adjacent (i.e. north, south, west, east) to Wumpus: stench (else: none).
  - Cell adjacent to pit: breeze (else: none).
  - Cell that contains gold: glitter (else: none).
  - You walk into a wall: bump (else: none).
  - Wumpus shot by arrow: scream (else: none).

SOME RIGHTS RESERVED

Reasoning in the Wumpus World

- **Example 1.9 (Reasoning in the Wumpus World).**
  As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



(1) Initial state

- **Note:** The agent has more knowledge than just the percepts ⇝ inference!

SOME RIGHTS RESERVED

Reasoning in the Wumpus World

- **Example 1.10 (Reasoning in the Wumpus World).**
  As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

(1) Initial state

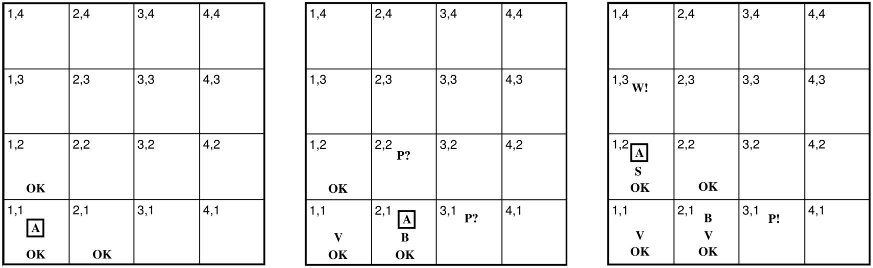
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(2) One step to right

- **Note:** The agent has more knowledge than just the percepts ⇝ inference!

Reasoning in the Wumpus World

- **Example 1.11 (Reasoning in the Wumpus World).**
  As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



(1) Initial state (2) One step to right (3) Back, and up to [1,2]

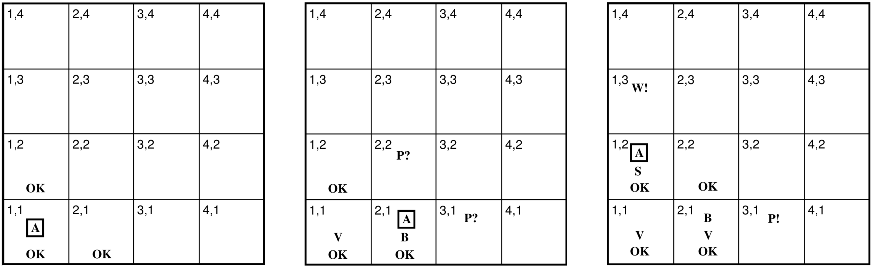
  - "*The Wumpus is in [1,3]*"! How do we know?

- **Note:** The agent has more knowledge than just the percepts ⇝ inference!

SOME RIGHTS RESERVED

Reasoning in the Wumpus World

- **Example 1.12 (Reasoning in the Wumpus World).**
  As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



(1) Initial state (2) One step to right (3) Back, and up to [1,2]

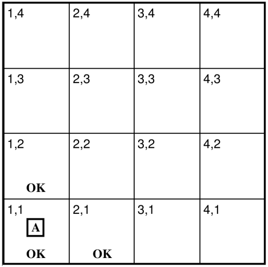
  - "*The Wumpus is in [1,3]*"! How do we know?
  - No stench in [2,1], so the stench in [1,2] can only come from [1,3].
  - "*There's a pit in [3,1]*"! How do we know?

- **Note:** The agent has more knowledge than just the percepts ⇝ inference!

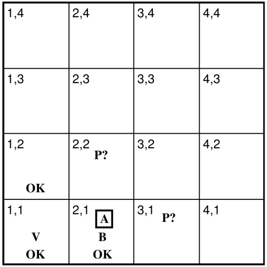
SOME RIGHTS RESERVED

Reasoning in the Wumpus World

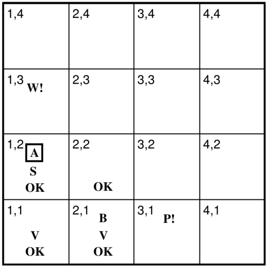
- **Example 1.13 (Reasoning in the Wumpus World).**
  As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



(1) Initial state



(2) One step to right



(3) Back, and up to [1,2]

  - "*The Wumpus is in [1,3]*"! How do we know?
  - No stench in [2,1], so the stench in [1,2] can only come from [1,3].
  - "*There's a pit in [3,1]*"! How do we know?
  - No breeze in [1,2], so the breeze in [2,1] can only come from [3,1].
- **Note:** The agent has more knowledge than just the percepts ⇝ inference!

Agents that Think Rationally

- **Problem:** But how can we build an agent that can do the necessary inferences?

SOME RIGHTS RESERVED

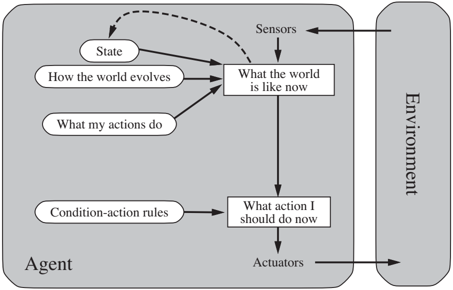
Agents that Think Rationally

- **Problem:** But how can we build an agent that can do the necessary inferences?
- **Idea:** Think Before You Act!
  “Thinking” = Inference about knowledge represented using logic.

SOME RIGHTS RESERVED

Agents that Think Rationally

- **Problem:** But how can we build an agent that can do the necessary inferences?
- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.
- **Definition 1.16.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about world state and its own actions. Agent schema:



The formal language of the logical system acts as a world description language.

SOME RIGHTS RESERVED

Agents that Think Rationally

- **Problem:** But how can we build an agent that can do the necessary inferences?
- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.
- **Definition 1.17.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about world state and its own actions. Agent function:

```
function KB−AGENT (percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time
  TELL(KB, MAKE−PERCEPT−SENTENCE(percept,t))
  action := ASK(KB, MAKE−ACTION−QUERY(t))
  TELL(KB, MAKE−ACTION−SENTENCE(action,t))
   t := t+1
  return action
```

  Its agent function maintains a knowledge base about the world, which is updated with percept descriptions (formalizations of the percepts) and action descriptions. The next action is the result of a suitable inference-based query to the knowledge base.

SOME RIGHTS RESERVED

10.1.2 Propositional Logic: Preview

SOME RIGHTS RESERVED

Logic: Basic Concepts (Representing Knowledge)

- We now preview some of the concepts involved in logic so that you have an intuition for the formal definitions below.

SOME RIGHTS RESERVED

Logic: Basic Concepts (Representing Knowledge)

- We now preview some of the concepts involved in logic so that you have an intuition for the formal definitions below.
- **Definition 1.24.** Syntax: What are legal formulae A in the logic?
- **Example 1.25.** "W" and "$W \Rightarrow S$".
  ($W\widehat{=}$ "*Wumpus is here*", $S\widehat{=}$ "*it stinks*", $W \Rightarrow S \widehat{=}$ If W, then S)

SOME RIGHTS RESERVED

Logic: Basic Concepts (Representing Knowledge)

- We now preview some of the concepts involved in logic so that you have an intuition for the formal definitions below.
- **Definition 1.30.** Syntax: What are legal formulae A in the logic?
- **Example 1.31.** "W" and "$W \Rightarrow S$".
  ($W \widehat{=}$ "*Wumpus is here*", $S \widehat{=}$ "*it stinks*", $W \Rightarrow S \widehat{=}$ If W, then S)
- **Definition 1.32.** Semantics: Which formulae A are true?
- **Observation:** Whether $W \Rightarrow S$ is true depends on whether W and S are!
- **Idea:** Capture the state of W and S… in a variable assignment.
- **Definition 1.33.** For a variable assignment φ, write $\varphi \models$ A if φ is true in the Wumpus world described by φ.
- **Example 1.34.** If $\varphi := \{W \mapsto \mathsf{T}, S \mapsto \mathsf{F}\}$, then $\varphi \models W$ but $\varphi \not\models (W \Rightarrow S)$.
- **Intuition:** Knowledge about the state of the world is described by formulae, interpretations evaluate them in the current world (they should turn out true!)

SOME RIGHTS RESERVED

Logic: Basic Concepts (Representing Knowledge)

- We now preview some of the concepts involved in logic so that you have an intuition for the formal definitions below.
- **Definition 1.36.** Syntax: What are legal formulae A in the logic?
- **Example 1.37.** "W" and "$W \Rightarrow S$".
  ($W \mathrel{\widehat{=}}$ "*Wumpus is here*", $S \mathrel{\widehat{=}}$ "*it stinks*", $W \Rightarrow S \mathrel{\widehat{=}}$ If W, then S)
- **Definition 1.38.** Semantics: Which formulae A are true?
- **Observation:** Whether $W \Rightarrow S$ is true depends on whether W and S are!
- **Idea:** Capture the state of W and S... in a variable assignment.
- **Definition 1.39.** For a variable assignment φ, write $\varphi \models$ A if φ is true in the Wumpus world described by φ.
- **Example 1.40.** If $\varphi := \{W \mapsto \mathsf{T}, S \mapsto \mathsf{F}\}$, then $\varphi \models W$ but $\varphi \not\models (W \Rightarrow S)$.
- **Intuition:** Knowledge about the state of the world is described by formulae, interpretations evaluate them in the current world (they should turn out true!)
- **Definition 1.41.** The process of representing a natural language text in the formal language of a logical system is called formalization.
- **Observation:** Formalizing a NL text or utterance makes it machine-actionable. (the ultimate purpose of AI)
- **Observation:** Formalization is an art/skill, not a science!

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.42.** Entailment: Which B follow from A, written A $\models$ B, meaning that, for all φ with $\varphi \models$ A, we have $\varphi \models$ B? E.g., $P \wedge (P \Rightarrow Q) \models Q$.
- **Intuition:** Entailment $\widehat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information

SOME RIGHTS RESERVED

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.47.** Entailment: Which B follow from A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? E.g., $P \wedge (P \Rightarrow Q) \models Q$.
- **Intuition:** Entailment $\widehat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information
- **Definition 1.48.** Deduction: Which formulas B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.49.** If $\mathcal{C}$ contains $\frac{A \quad A \Rightarrow B}{B}$ then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.52.** Entailment: Which B follow from A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? E.g., $P \wedge (P \Rightarrow Q) \models Q$.
- **Intuition:** Entailment $\widehat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information
- **Definition 1.53.** Deduction: Which formulas B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.54.** If $\mathcal{C}$ contains $\frac{A \quad A \Rightarrow B}{B}$ then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$
- **Intuition:** Deduction $\widehat{=}$ process in an actual computer trying to reason about entailment. E.g. a mechanical process attempting to determine Wumpus position.

SOME RIGHTS RESERVED

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.57.** Entailment: Which B follow from A, written $A \vDash B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? E.g., $P \wedge (P \Rightarrow Q) \vDash Q$.
- **Intuition:** Entailment $\widehat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information
- **Definition 1.58.** Deduction: Which formulas B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.59.** If $\mathcal{C}$ contains $\frac{A \quad A \Rightarrow B}{B}$ then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$
- **Intuition:** Deduction $\widehat{=}$ process in an actual computer trying to reason about entailment. E.g. a mechanical process attempting to determine Wumpus position.
- **Critical Insight:** Entailment is purely semantical and gives a mathematical foundation of reasoning in PL^0, while Deduction is purely syntactic and can be implemented well. (but this only helps if they are related)
- **Definition 1.60.** Soundness: whenever $A \vdash_{\mathcal{C}} B$, we also have $A \vDash B$.
- **Definition 1.61.** Completeness: whenever $A \vDash B$, we also have $A \vdash_{\mathcal{C}} B$.

SOME RIGHTS RESERVED

General Problem Solving using Logic

- **Idea:** Any problem that can be formulated as reasoning about logic. $\leadsto$ use off-the-shelf reasoning tool.
- *Very* successful using propositional logic and modern SAT solvers! (Propositional satisfiability testing;)

SOME RIGHTS RESERVED

Propositional Logic and its Applications

- Propositional logic = canonical form of knowledge + reasoning.
  - Syntax: Atomic propositions that can be either true or false, connected by "and, or, and not".
  - Semantics: Assign value to every proposition, evaluate connectives.
- **Applications:** Despite its simplicity, widely applied!
  - **Product configuration** (e.g., Mercedes). Check consistency of customized combinations of components.
  - **Hardware verification** (e.g., Intel, AMD, IBM, Infineon). Check whether a circuit has a desired property p.
  - **Software verification**: Similar.
  - **CSP applications**: Propositional logic can be (successfully!) used to formulate and solve constraint satisfaction problems. (see)
- gives an example for verification.

SOME RIGHTS RESERVED

10.1.3 Propositional Logic: Agenda

SOME RIGHTS RESERVED

Our Agenda for This Topic

- **This subsection**: Basic definitions and concepts; tableaux, resolution.
  - Sets up the framework. Resolution is the quintessential reasoning procedure underlying most successful SAT solvers.
- **Next Section ()**: The Davis Putnam procedure and clause learning; practical problem structure.
  - State-of-the-art algorithms for reasoning about propositional logic, and an important observation about how they behave.

SOME RIGHTS RESERVED

Our Agenda for This Chapter

- Propositional logic: What's the syntax and semantics? How can we capture deduction?
  - We study this logic formally.
- **Tableaux, Resolution**: How can we make deduction mechanizable? What are its properties?
  - Formally introduces the most basic machine-oriented reasoning algorithm.
- **Killing a Wumpus**: How can we use all this to figure out where the Wumpus is?
  - Coming back to our introductory example.

SOME RIGHTS RESERVED

10.2 Propositional Logic (Syntax/Semantics)

Propositional Logic (Syntax)

- **Definition 2.1 (Syntax).** The formulae of propositional logic (write PL^0) are made up from
  - propositional variables: $\mathcal{V}_0 := \{P, Q, R, P^1, P^2, \ldots\}$ (countably infinite)
  - A propositional signature: constants/constructors called connectives: $\Sigma_0 := \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots\}$

  We define the set $wff_0(\mathcal{V}_0)$ of well-formed propositional formulae (wffs) as
  - propositional variables,
  - the logical constants T and F,
  - negations $\neg$A,
  - conjunctions A $\wedge$ B(A and B are called conjuncts),
  - disjunctions A $\vee$ B (A and B are called disjuncts),
  - implications A $\Rightarrow$ B, and
  - equivalences (or biimplications). A $\Leftrightarrow$ B,

  where A, B $\in wff_0(\mathcal{V}_0)$ themselves.
- **Example 2.2.** $P \wedge Q, P \vee Q, \neg P \vee Q \Leftrightarrow P \Rightarrow Q \in wff_0(\mathcal{V}_0)$
- **Definition 2.3.** Propositional formulae without connectives are called atomic (or an atoms) and complex otherwise.

SOME RIGHTS RESERVED

Propositional Logic Grammar Overview

- **Grammar for Propositional Logic:**

propositional variables	X	$::=$	$\mathcal{V}_0 = \{P, Q, R, \ldots, \ldots\}$	variables
propositional formulae	A	$::=$	X	variable
		$\mid$	$T \mid F$	truth values
		$\mid$	$\neg A$	negation
		$\mid$	$A_1 \wedge A_2$	conjunction
		$\mid$	$A_1 \vee A_2$	disjunction
		$\mid$	$A_1 \Rightarrow A_2$	implication
		$\mid$	$A_1 \Leftrightarrow A_2$	equivalence

SOME RIGHTS RESERVED

Alternative Notations for Connectives

Here	**Elsewhere**		
$\neg A$	$\sim A$	$\overline{A}$	
$A \wedge B$	$A \& B$	$A \bullet B$	A, B
$A \vee B$	$A + B$	$A \mid B$	$A ; B$
$A \Rightarrow B$	$A \rightarrow B$	$A \supset B$	
$A \Leftrightarrow B$	$A \leftrightarrow B$	$A \equiv B$	
F	$\bot$	0	
T	$\top$	1	

SOME RIGHTS RESERVED

Semantics of PL^0 (Models)

- **Warning:** For the official semantics of PL^0 we will separate the tasks of giving meaning to connectives and propositional variables to different mappings.
- This will generalize better to other logical systems. (and thus applications)

Semantics of PL^0 (Models)

- **Warning:** For the official semantics of PL^0 we will separate the tasks of giving meaning to connectives and propositional variables to different mappings.
- This will generalize better to other logical systems. (and thus applications)
- **Definition 2.5.** A model $\mathcal{M} := \langle \mathcal{D}_0, \mathcal{I} \rangle$ for propositional logic consists of
  - the universe $\mathcal{D}_0 = \{\mathsf{T}, \mathsf{F}\}$
  - the interpretation $\mathcal{I}$ that assigns values to essential connectives.
  - $\mathcal{I}(\neg)\colon \mathcal{D}_0 \to \mathcal{D}_0; \mathsf{T} \mapsto \mathsf{F}, \mathsf{F} \mapsto \mathsf{T}$
  - $\mathcal{I}(\wedge)\colon \mathcal{D}_0 \times \mathcal{D}_0 \to \mathcal{D}_0; \langle \alpha, \beta \rangle \mapsto \mathsf{T}$, iff $\alpha = \beta = \mathsf{T}$

  We call a constant a logical constant, iff its value is fixed by the interpretation.
- Treat the other connectives as abbreviations, e.g. $\mathsf{A} \vee \mathsf{B} \widehat{=} \neg(\neg \mathsf{A} \wedge \neg \mathsf{B})$ and $\mathsf{A} \Rightarrow \mathsf{B} \widehat{=} \neg \mathsf{A} \vee \mathsf{B}$, and $T \widehat{=} P \vee \neg P$ (only need to treat $\neg, \wedge$ directly)

Semantics of PL^0 (Models)

- **Warning:** For the official semantics of PL^0 we will separate the tasks of giving meaning to connectives and propositional variables to different mappings.
- This will generalize better to other logical systems. (and thus applications)
- **Definition 2.6.** A model $\mathcal{M} := \langle \mathcal{D}_0, \mathcal{I} \rangle$ for propositional logic consists of
  - the universe $\mathcal{D}_0 = \{\mathsf{T}, \mathsf{F}\}$
  - the interpretation $\mathcal{I}$ that assigns values to essential connectives.
  - $\mathcal{I}(\neg)\colon \mathcal{D}_0 \to \mathcal{D}_0; \mathsf{T} \mapsto \mathsf{F}, \mathsf{F} \mapsto \mathsf{T}$
  - $\mathcal{I}(\wedge)\colon \mathcal{D}_0 \times \mathcal{D}_0 \to \mathcal{D}_0; \langle \alpha, \beta \rangle \mapsto \mathsf{T}$, iff $\alpha = \beta = \mathsf{T}$

  We call a constant a logical constant, iff its value is fixed by the interpretation.
- Treat the other connectives as abbreviations, e.g. $\mathsf{A} \vee \mathsf{B} \widehat{=} \neg(\neg \mathsf{A} \wedge \neg \mathsf{B})$ and $\mathsf{A} \Rightarrow \mathsf{B} \widehat{=} \neg \mathsf{A} \vee \mathsf{B}$, and $T \widehat{=} P \vee \neg P$ (only need to treat $\neg, \wedge$ directly)
- **Note:** PL^0 is a single-model logical system with canonical model $\langle \mathcal{D}_0, \mathcal{I} \rangle$.

SOME RIGHTS RESERVED

Semantics of PL^0 (Evaluation)

- **Problem:** The interpretation function $\mathcal{I}$ only assigns meaning to connectives.
- **Definition 2.7.** A variable assignment $\varphi\colon \mathcal{V}_0 \to \mathcal{D}_0$ assigns values to propositional variables.
- **Definition 2.8.** The value function $\mathcal{I}_\varphi\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_0$ assigns values to PL^0 formulae. It is recursively defined,
  - $\mathcal{I}_\varphi(P) = \varphi(P)$ (base case)
  - $\mathcal{I}_\varphi(\neg \mathrm{A}) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathrm{A}))$.
  - $\mathcal{I}_\varphi(\mathrm{A} \wedge \mathrm{B}) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathrm{A}), \mathcal{I}_\varphi(\mathrm{B}))$.
- **Note:** $\mathcal{I}_\varphi(\mathrm{A} \vee \mathrm{B}) = \mathcal{I}_\varphi(\neg(\neg \mathrm{A} \wedge \neg \mathrm{B}))$ is only determined by $\mathcal{I}_\varphi(\mathrm{A})$ and $\mathcal{I}_\varphi(\mathrm{B})$, so we think of the defined connectives as logical constants as well.
- **Alternative Notation:** write $[\![\mathrm{A}]\!]_\varphi$ for $\mathcal{I}_\varphi(\mathrm{A})$. (and $[\![\mathrm{A}]\!]$, if A is ground)
- **Definition 2.9.** Two formulae A and B are called equivalent, iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathcal{I}_\varphi(\mathrm{B})$ for all variable assignments φ.

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.10.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

  $\mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.11.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned} & \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\ = \quad & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.12.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4)))
\end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.13.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{array}{rl} & \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\ = & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\ = & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\ = & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \end{array}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.14.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4))))
\end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.15.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F})))
\end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.16.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F}))
\end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.17.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F}))
\end{aligned}$$

Computing Semantics

- **Example 2.18.** Let $\varphi := [\mathsf{T}/P_1],[\mathsf{F}/P_2],[\mathsf{T}/P_3],[\mathsf{F}/P_4],\ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2),\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1),\mathcal{I}_\varphi(P_2)),\mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)),\mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1),\varphi(P_2)),\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)),\mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3),\mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T},\mathsf{F}),\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1),\mathcal{I}_\varphi(P_2))),\mathcal{I}(\wedge)(\varphi(P_3),\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)),\varphi(P_2))),\mathcal{I}(\wedge)(\mathsf{T},\mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)),\mathsf{F})),\mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}),\mathsf{F})),\mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F},\mathsf{F})),\mathsf{F}))
\end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.19.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F}, \mathsf{F})), \mathsf{F})) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathsf{F}), \mathsf{F}))
\end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.20.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F}, \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathsf{F}), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}))
\end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.21.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F}, \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathsf{F}), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathsf{T}, \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathsf{T})
\end{aligned}$$

SOME RIGHTS RESERVED

Computing Semantics

- **Example 2.22.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F}, \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathsf{F}), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathsf{T}, \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathsf{T}) \\
= \; & \mathsf{T}
\end{aligned}
$$

- What a mess!

SOME RIGHTS RESERVED

Propositional Identities

- **Definition 2.23.** We have the following identities in propositional logic:

Name	for $\wedge$	for $\vee$
Idempotence	$\varphi \wedge \varphi = \varphi$	$\varphi \vee \varphi = \varphi$
Identity	$\varphi \wedge T = \varphi$	$\varphi \vee F = \varphi$
Absorption 1	$\varphi \wedge F = F$	$\varphi \vee T = T$
Commutativity	$\varphi \wedge \psi = \psi \wedge \varphi$	$\varphi \vee \psi = \psi \vee \varphi$
Associativity	$\varphi \wedge (\psi \wedge \theta) = (\varphi \wedge \psi) \wedge \theta$	$\varphi \vee (\psi \vee \theta) = (\varphi \vee \psi) \vee \theta$
Distributivity	$\varphi \wedge (\psi \vee \theta) = \varphi \wedge \psi \vee \varphi \wedge \theta$	$\varphi \vee \psi \wedge \theta = (\varphi \vee \psi) \wedge (\varphi \vee \theta)$
Absorption 2	$\varphi \wedge (\varphi \vee \theta) = \varphi$	$\varphi \vee \varphi \wedge \theta = \varphi$
De Morgan rule	$\neg(\varphi \wedge \psi) = \neg\varphi \vee \neg\psi$	$\neg(\varphi \vee \psi) = \neg\varphi \wedge \neg\psi$
double negation	$\neg\neg\varphi = \varphi$	
Definitions	$\varphi \Rightarrow \psi = \neg\varphi \vee \psi$	$\varphi \Leftrightarrow \psi = (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$

- **Idea:** How about using these as inference component (simplification) to simplify calculations like the one in ???. (see below)

Semantic Properties of Propositional Formulae

- **Definition 2.24.** Let $\mathcal{M} := \langle \mathcal{D}_0, \mathcal{I} \rangle$ be our model, then we say that A is
  - true under φ in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathsf{T}$, (write $\mathcal{M} \models^\varphi \mathrm{A}$)
  - falsifies φ in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathsf{F}$, (write $\mathcal{M} \not\models^\varphi \mathrm{A}$)
  - satisfiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathsf{T}$ for some assignment φ,
  - valid in $\mathcal{M}$, iff $\mathcal{M} \models^\varphi \mathrm{A}$ for all variable assignments φ,
  - falsifiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathsf{F}$ for some assignments φ, and
  - unsatisfiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathsf{F}$ for all assignments φ.
- **Example 2.25.** $x \vee x$ is satisfiable and falsifiable.
- **Example 2.26.** $x \vee \neg x$ is valid and $x \wedge \neg x$ is unsatisfiable.
- **Note:** As PL^0 is a single-model logical system, we can elide the reference to the canonical model and regain the notation $\varphi \models \mathrm{A}$ (φ true under A) from the preview for $\mathcal{M} \models^\varphi \mathrm{A}$.
- **Definition 2.27 (Entailment).** (aka. logical consequence)
  We say that A entails B (write $\mathrm{A} \models \mathrm{B}$), iff $\mathcal{I}_\varphi(\mathrm{B}) = \mathsf{T}$ for all φ with $\mathcal{I}_\varphi(\mathrm{A}) = \mathsf{T}$ (i.e. all assignments that make A true also make B true)

SOME RIGHTS RESERVED

A better mouse-trap: Truth Tables

- Truth tables visualize truth functions:

$\neg$	
$\top$	F
$\bot$	T

$\wedge$	$\top$	$\bot$
$\top$	T	F
$\bot$	F	F

$\vee$	$\top$	$\bot$
$\top$	T	T
$\bot$	T	F

- If we are interested in values for all assignments (e.g $z \wedge x \vee \neg(z \wedge y)$)

assignments			intermediate results			full
x	y	z	$e_1 := z \wedge y$	$e_2 := \neg e_1$	$e_3 := z \wedge x$	$e_3 \vee e_2$
F	F	F	F	T	F	T
F	F	T	F	T	F	T
F	T	F	F	T	F	T
F	T	T	T	F	F	F
T	F	F	F	T	F	T
T	F	T	F	T	T	T
T	T	F	F	T	F	T
T	T	T	T	F	T	T

SOME RIGHTS RESERVED

Hair Color in Propositional Logic

- There are three persons, Stefan, Nicole, and Jochen.
  1. Their hair colors are black, red, or green.
  2. Their study subjects are AI, Physics, or Chinese at least one studies AI.
     - 2.1 Persons with red or green hair do not study AI.
     - 2.2 Neither the Physics nor the Chinese students have black hair.
     - 2.3 Of the two male persons, one studies Physics, and the other studies Chinese.
- **Question:** Who studies AI?

  (A) Stefan (B) Nicole (C) Jochen (D) Nobody

SOME RIGHTS RESERVED

Hair Color in Propositional Logic

- There are three persons, Stefan, Nicole, and Jochen.
  1. Their hair colors are black, red, or green.
  2. Their study subjects are AI, Physics, or Chinese at least one studies AI.
     - 2.1 Persons with red or green hair do not study AI.
     - 2.2 Neither the Physics nor the Chinese students have black hair.
     - 2.3 Of the two male persons, one studies Physics, and the other studies Chinese.
- **Question:** Who studies AI?

  (A) Stefan (B) Nicole (C) Jochen (D) Nobody
- **Answer:** You can solve this using PL^0, if we accept $bla(S)$, etc. as propositional variables.

  We first express what we know: For every $x \in \{S, N, J\}$ (Stefan, Nicole, Jochen) we have
  1. $bla(x) \vee red(x) \vee gre(x)$; (note: three formulae)
  2. $ai(x) \vee phy(x) \vee chi(x)$ and $ai(S) \vee ai(N) \vee ai(J)$
     - 2.1 $ai(x) \Rightarrow \neg red(x) \wedge \neg gre(x)$.
     - 2.2 $phy(x) \Rightarrow \neg bla(x)$ and $chi(x) \Rightarrow \neg bla(x)$.
     - 2.3 $phy(S) \wedge chi(J) \vee phy(J) \wedge chi(S)$.

  Now, we obtain new knowledge via entailment steps:
  3. 1. together with 2.1 entails that $ai(x) \Rightarrow bla(x)$ for every $x \in \{S, N, J\}$,
  4. thus $\neg bla(S) \wedge \neg bla(J)$ by 2.3 and 2.2 and
  5. so $\neg ai(S) \wedge \neg ai(J)$ by 3. and 4.
  6. With 2. the latter entails $ai(N)$.

SOME RIGHTS RESERVED

10.3 Inference in Propositional Logics

SOME RIGHTS RESERVED

Agents that Think Rationally

- **Problem:** But how can we build an agent that can do the necessary inferences?

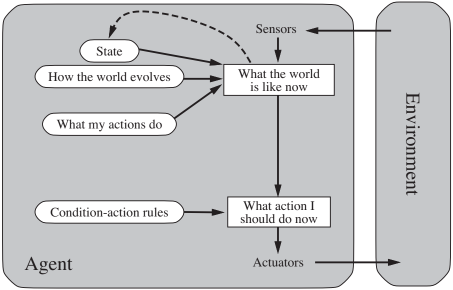
Agents that Think Rationally

- **Problem:** But how can we build an agent that can do the necessary inferences?
- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.

SOME RIGHTS RESERVED

Agents that Think Rationally

- **Problem:** But how can we build an agent that can do the necessary inferences?
- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.
- **Definition 3.3.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about world state and its own actions. Agent schema:



The formal language of the logical system acts as a world description language.

Agents that Think Rationally

- **Problem:** But how can we build an agent that can do the necessary inferences?
- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.
- **Definition 3.4.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about world state and its own actions. Agent function:

```
function KB−AGENT (percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time
  TELL(KB, MAKE−PERCEPT−SENTENCE(percept,t))
  action := ASK(KB, MAKE−ACTION−QUERY(t))
  TELL(KB, MAKE−ACTION−SENTENCE(action,t))
  t := t+1
  return action
```

Its agent function maintains a knowledge base about the world, which is updated with percept descriptions (formalizations of the percepts) and action descriptions. The next action is the result of a suitable inference-based query to the knowledge base.

SOME RIGHTS RESERVED

A Simple Formal System: Prop. Logic with Hilbert-Calculus

- **Formulae:** Built from propositional variables: $P, Q, R\ldots$ and implication: $\Rightarrow$
- **Semantics:** $\mathcal{I}_\varphi(P) = \varphi(P)$ and $\mathcal{I}_\varphi(\mathrm{A} \Rightarrow \mathrm{B}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathsf{F}$ or $\mathcal{I}_\varphi(\mathrm{B}) = \mathsf{T}$.
- **Definition 3.5.** The Hilbert calculus $\mathcal{H}^0$ consists of the inference rules:

$$\frac{}{P \Rightarrow Q \Rightarrow P}\,\mathrm{K} \qquad \frac{}{(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R}\,\mathrm{S}$$

$$\frac{\mathrm{A} \Rightarrow \mathrm{B} \quad \mathrm{A}}{\mathrm{B}}\,\mathrm{MP} \qquad \frac{\mathrm{A}}{[\mathrm{B}/X](\mathrm{A})}\,\mathrm{Subst}$$

- **Example 3.6.** A $\mathcal{H}^0$ theorem $\mathrm{C} \Rightarrow \mathrm{C}$ and its proof

  Proof: We show that $\emptyset \vdash_{\mathcal{H}^0} \mathrm{C} \Rightarrow \mathrm{C}$

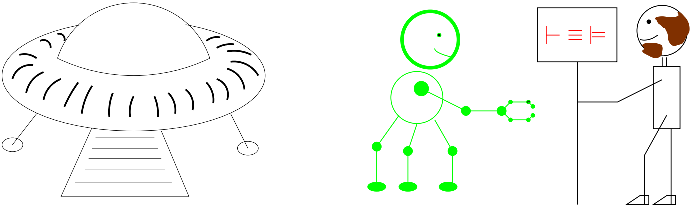
  1. $(\mathrm{C} \Rightarrow (\mathrm{C} \Rightarrow \mathrm{C}) \Rightarrow \mathrm{C}) \Rightarrow (\mathrm{C} \Rightarrow \mathrm{C} \Rightarrow \mathrm{C}) \Rightarrow \mathrm{C} \Rightarrow \mathrm{C}$ (S with $[\mathrm{C}/P], [\mathrm{C} \Rightarrow \mathrm{C}/Q], [\mathrm{C}/R]$)
  2. $\mathrm{C} \Rightarrow (\mathrm{C} \Rightarrow \mathrm{C}) \Rightarrow \mathrm{C}$ (K with $[\mathrm{C}/P], [\mathrm{C} \Rightarrow \mathrm{C}/Q]$)
  3. $(\mathrm{C} \Rightarrow \mathrm{C} \Rightarrow \mathrm{C}) \Rightarrow \mathrm{C} \Rightarrow \mathrm{C}$ (MP on P.1 and P.2)
  4. $\mathrm{C} \Rightarrow \mathrm{C} \Rightarrow \mathrm{C}$ (K with $[\mathrm{C}/P], [\mathrm{C}/Q]$)
  5. $\mathrm{C} \Rightarrow \mathrm{C}$ (MP on P.3 and P.4)

  □

SOME RIGHTS RESERVED

Soundness and Completeness

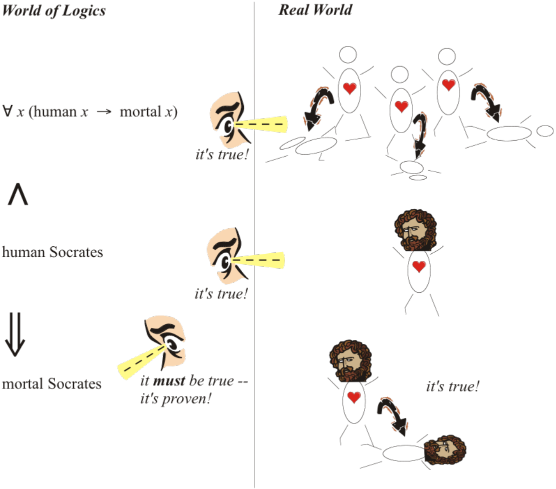
- **Definition 3.7.** Let $\mathcal{L} := \langle \mathcal{L}, \mathcal{M}, \models \rangle$ be a logical system, then we call a calculus $\mathcal{C}$ for $\mathcal{L}$,
  - sound (or correct), iff $\mathcal{H} \models \mathrm{A}$, whenever $\mathcal{H} \vdash_{\mathcal{C}} \mathrm{A}$, and
  - complete, iff $\mathcal{H} \vdash_{\mathcal{C}} \mathrm{A}$, whenever $\mathcal{H} \models \mathrm{A}$.
- **Goal:** Find calculi C, such that $\vdash_C \mathrm{A}$ iff $\models \mathrm{A}$ (provability and validity coincide)
  - To TRUTH through PROOF (CALCULEMUS [Leibniz ∼1680])
-



SOME RIGHTS RESERVED

The Miracle of Logic

- Purely formal derivations are true in the real world!



10.4 Propositional Natural Deduction Calculus

Calculi: Natural Deduction ($\mathcal{ND}_0$; Gentzen [**Gentzen:uudlsi35**])

- **Idea:** $\mathcal{ND}_0$ tries to mimic human argumentation for theorem proving.
- **Definition 4.1.** The propositional natural deduction calculus $\mathcal{ND}_0$ has inference rules for the introduction and elimination of connectives:

Introduction Elimination Axiom

$$\frac{A \quad B}{A \wedge B}\;\wedge I \qquad \frac{A \wedge B}{A}\;\wedge E_l \qquad \frac{A \wedge B}{B}\;\wedge E_r \qquad \frac{}{A \vee \neg A}\;\mathrm{Ax}$$

$$\frac{\begin{array}{c}[A]^1\\ \vdots \\ B\end{array}}{A \Rightarrow B}\;\Rightarrow I^1 \qquad \frac{A \Rightarrow B \quad A}{B}\;\Rightarrow E$$

$\Rightarrow I^a$ proves $A \Rightarrow B$ by exhibiting a $\mathcal{ND}_0$ derivation $\mathcal{D}$ (depicted by the double horizontal lines) of B from the local hypothesis A; $\Rightarrow I^a$ then discharges (get rid of A, which can only be used in $\mathcal{D}$) the local hypothesis and concludes $A \Rightarrow B$. This mode of reasoning is called hypothetical reasoning.

- **Definition 4.2.** Given a set $\mathcal{H} \subseteq wff_0(\mathcal{V}_0)$ of assumptions and a conclusion C, we write $\mathcal{H} \vdash_{\mathcal{ND}_0} C$, iff there is a $\mathcal{ND}_0$ derivation tree whose leaves are in $\mathcal{H}$.
- **Note:** Ax is used only in classical logic. (otherwise constructive/intuitionistic)

SOME RIGHTS RESERVED

Natural Deduction: Examples

- **Example 4.3 (Inference with Local Hypotheses).**

$$\dfrac{\dfrac{\dfrac{[(A\wedge B)]^1}{B}\,\wedge E_r \qquad \dfrac{[(A\wedge B)]^1}{A}\,\wedge E_l}{B\wedge A}\,\wedge I}{A\wedge B\Rightarrow B\wedge A}\,\Rightarrow I^1$$

$$\dfrac{\dfrac{\begin{array}{c}[A]^1\\ [B]^2\\ A\end{array}}{B\Rightarrow A}\,\Rightarrow I^2}{A\Rightarrow B\Rightarrow A}\,\Rightarrow I^1$$

SOME RIGHTS RESERVED

A Deduction Theorem for $\mathcal{ND}_0$

- **Theorem 4.4.** *$\mathcal{H}, \mathrm{A} \vdash_{\mathcal{ND}_0} \mathrm{B}$, iff $\mathcal{H} \vdash_{\mathcal{ND}_0} \mathrm{A} \Rightarrow \mathrm{B}$.*
- *Proof:* We show the two directions separately
  1. If $\mathcal{H}, \mathrm{A} \vdash_{\mathcal{ND}_0} \mathrm{B}$, then $\mathcal{H} \vdash_{\mathcal{ND}_0} \mathrm{A} \Rightarrow \mathrm{B}$ by $\Rightarrow I$, and
  2. If $\mathcal{H} \vdash_{\mathcal{ND}_0} \mathrm{A} \Rightarrow \mathrm{B}$, then $\mathcal{H}, \mathrm{A} \vdash_{\mathcal{ND}_0} \mathrm{A} \Rightarrow \mathrm{B}$ by weakening and $\mathcal{H}, \mathrm{A} \vdash_{\mathcal{ND}_0} \mathrm{B}$ by $\Rightarrow E$.

☐

SOME RIGHTS RESERVED

More Rules for Natural Deduction

- **Note:** $\mathcal{ND}_0$ does not try to be minimal, but comfortable to work in!
- **Definition 4.5.** $\mathcal{ND}_0$ has the following additional inference rules for the remaining connectives.

Introduction	Elimination
$\dfrac{A}{A \lor B}\ \lor I_l \qquad \dfrac{B}{A \lor B}\ \lor I_r$	$\dfrac{A \lor B \quad \begin{array}{c}[A]^1\\ \vdots\\ C\end{array} \quad \begin{array}{c}[B]^1\\ \vdots\\ C\end{array}}{C}\ \lor E^1$
$\dfrac{\begin{array}{c}[A]^1\\ \vdots\\ C\end{array} \quad \begin{array}{c}[A]^1\\ \vdots\\ \neg C\end{array}}{\neg A}\ \neg I^1$	$\dfrac{\neg\neg A}{A}\ \neg E$
$\dfrac{\neg A \quad A}{F}\ FI$	$\dfrac{F}{A}\ FE$

- **Again:** $\neg E$ is used only in classical logic (otherwise constructive/intuitionistic)

SOME RIGHTS RESERVED

Natural Deduction in Sequent Calculus Formulation

- **Idea:** Represent hypotheses explicitly. (lift calculus to judgments)
- **Definition 4.6.** A judgment is a meta-statement about the provability of propositions.
- **Definition 4.7.** A sequent is a judgment of the form $\mathcal{H} \vdash A$ about the provability of the formula A from the set $\mathcal{H}$ of hypotheses. We write $\vdash A$ for $\emptyset \vdash A$.
- **Idea:** Reformulate $\mathcal{ND}_0$ inference rules so that they act on sequents.
- **Example 4.8.**We give the sequent style version of 4.3:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{A \wedge B \vdash A \wedge B}\,\mathrm{Ax}}{A \wedge B \vdash B}\,\wedge E_r \qquad \cfrac{\cfrac{}{A \wedge B \vdash A \wedge B}\,\mathrm{Ax}}{A \wedge B \vdash A}\,\wedge E_l}{A \wedge B \vdash B \wedge A}\,\wedge I}{\vdash A \wedge B \Rightarrow B \wedge A}\,\Rightarrow I \qquad\qquad \cfrac{\cfrac{\cfrac{}{A, B \vdash A}\,\mathrm{Ax}}{A \vdash B \Rightarrow A}\,\Rightarrow I}{\vdash A \Rightarrow B \Rightarrow A}\,\Rightarrow I$$

- **Note:** Even though the antecedent of a sequent is written like a sequences, it is actually a set. In particular, we can permute and duplicate members at will.

Sequent-Style Rules for Natural Deduction

- **Definition 4.9.** The following inference rules make up the propositional sequent style natural deduction calculus $\mathcal{ND}^0_\vdash$:

$$\frac{}{\Gamma, A \vdash A}\,\mathrm{Ax} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\,\mathrm{weaken} \qquad \frac{}{\Gamma \vdash A \vee \neg A}\,\mathrm{TND}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge E_l \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,\wedge E_r$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee I_l \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee I_r \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\,\vee E$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\,\Rightarrow I \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\,\Rightarrow E$$

$$\frac{\Gamma, A \vdash F}{\Gamma \vdash \neg A}\,\neg I \qquad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}\,\neg E$$

$$FI\,\frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash F} \qquad FE\,\frac{\Gamma \vdash F}{\Gamma \vdash A}$$

Linearized Notation for (Sequent-Style) ND Proofs

- **Definition 4.10.** Linearized notation for sequent-style ND proofs after **[Lemmon:BL65]**

$$\begin{array}{llll} 1. & \mathcal{H}_1 & \vdash & A_1 \quad (\mathcal{J}_1) \\ 2. & \mathcal{H}_2 & \vdash & A_2 \quad (\mathcal{J}_2) \\ 3. & \mathcal{H}_3 & \vdash & A_3 \quad (\mathcal{J}_3 1, 2) \end{array} \qquad \text{corresponds to} \qquad \dfrac{\mathcal{H}_1 \vdash A_1 \quad \mathcal{H}_2 \vdash A_2}{\mathcal{H}_3 \vdash A_3}\mathcal{R}$$

- **Example 4.11.** We show a linearized version of the $\mathcal{ND}_0$ examples ???

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\text{Ax}}{A \wedge B \vdash B}\wedge E_r \qquad \dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\text{Ax}}{A \wedge B \vdash A}\wedge E_l}{A \wedge B \vdash B \wedge A}\wedge I}{\vdash A \wedge B \Rightarrow B \wedge A}\Rightarrow I \qquad\qquad \dfrac{\dfrac{\dfrac{}{A, B \vdash A}\text{Ax}}{A \vdash B \Rightarrow A}\Rightarrow I}{\vdash A \Rightarrow B \Rightarrow A}\Rightarrow I$$

#	*hyp*	$\vdash$	*formula*	*NDjust*
1.	1	$\vdash$	$A \wedge B$	Ax
2.	1	$\vdash$	B	$\wedge E_r$ 1
3.	1	$\vdash$	A	$\wedge E_l$ 1
4.	1	$\vdash$	$B \wedge A$	$\wedge I$ 2, 3
5.		$\vdash$	$A \wedge B \Rightarrow B \wedge A$	$\Rightarrow I$ 4

#	*hyp*	$\vdash$	*formula*	*NDjust*
1.	1	$\vdash$	A	Ax
2.	2	$\vdash$	B	Ax
3.	1, 2	$\vdash$	A	weaken 1, 2
4.	1	$\vdash$	$B \Rightarrow A$	$\Rightarrow I$ 3
5.		$\vdash$	$A \Rightarrow B \Rightarrow A$	$\Rightarrow I$ 4

SOME RIGHTS RESERVED

10.5 Predicate Logic Without Quantifiers

SOME RIGHTS RESERVED

Issues with Propositional Logic

- **Awkward to write for humans:** E.g., to model the Wumpus world we had to make a copy of the rules for every cell . . .
  $R_1 := \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$
  $R_2 := \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}$
  $R_3 := \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}$
  Compared to

  > "*Cell adjacent to Wumpus: Stench (else: None)*"

  that is not a very nice description language . . .
- **Can we design a more human-like logic?:** Yep!
- **Idea:** Introduce explict representations for
  - individuals, e.g. the wumpus, the gold, numbers, . . .
  - functions on individuals, e.g. the cell at i, j, . . .
  - relations between them, e.g. being in a cell, being adjacent, . . .

  This is essentially the same as PL^0, so we can reuse the calculi. (up next)

Individuals and their Properties/Relationships

- **Observation:** We want to talk about individuals like Stefan, Nicole, and Jochen
  and their properties, e.g. being blond, or studying AI
  and relationships, e.g. that "*Stefan loves Nicole*".
- **Idea:** Re-use PL^0, but replace propositional variables with something more expressive! (instead of fancy variable name trick)

Individuals and their Properties/Relationships

- **Observation:** We want to talk about individuals like Stefan, Nicole, and Jochen
  and their properties, e.g. being blond, or studying AI
  and relationships, e.g. that "*Stefan loves Nicole*".
- **Idea:** Re-use PL^0, but replace propositional variables with something more expressive! (instead of fancy variable name trick)
- **Definition 5.2.** A first-order signature $\langle\Sigma^f,\Sigma^p\rangle$ consists of
  - $\Sigma^f := \bigcup_{k\in\mathbb{N}}\Sigma^f_k$ of function constants, where members of Σ^f_k denote k-ary functions on individuals,
  - $\Sigma^p := \bigcup_{k\in\mathbb{N}}{\Sigma^p}_k$ of predicate constants, where members of ${\Sigma^p}_k$ denote k-ary relations among individuals,

  where Σ^f_k and ${\Sigma^p}_k$ are pairwise disjoint, countable sets of symbols for each $k\in\mathbb{N}$.
  A 0-ary function constant refers to a single individual, therefore we call it a individual constant.

SOME RIGHTS RESERVED

A Grammar for $\mathrm{PL}^{\mathrm{nq}}$

- **Definition 5.3.** The formulae of $\mathrm{PL}^{\mathrm{nq}}$ are given by the following grammar

function constants	f^k	$\in$	Σ^f_k	
predicate constants	p^k	$\in$	$\Sigma^{p}{}_{k}$	
terms	t	$::=$	f^0	individualconstant
		$\mid$	$f^k(t_1,\ldots,t_k)$	application
formulae	A	$::=$	$p^k(t_1,\ldots,t_k)$	atomic
		$\mid$	$\neg A$	negation
		$\mid$	$A_1 \wedge A_2$	conjunction

SOME RIGHTS RESERVED

$\mathrm{PL}^{\mathrm{nq}}$ Semantics

- **Definition 5.4.** Domains $\mathcal{D}_0 = \{\mathsf{T}, \mathsf{F}\}$ of truth values and $\mathcal{D}_\iota \neq \emptyset$ of individuals.
- **Definition 5.5.** Interpretation $\mathcal{I}$ assigns values to constants, e.g.
  - $\mathcal{I}(\neg)\colon \mathcal{D}_0 \to \mathcal{D}_0; \mathsf{T} \mapsto \mathsf{F}; \mathsf{F} \mapsto \mathsf{T}$ and $\mathcal{I}(\wedge) = \ldots$ (as in PL^0)
  - $\mathcal{I}\colon \Sigma_0^f \to \mathcal{D}_\iota$ (interpret individual constants as individuals)
  - $\mathcal{I}\colon \Sigma_k^f \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (interpret function constants as functions)
  - $\mathcal{I}\colon {\Sigma^p}_k \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicate constants as relations)
- **Definition 5.6.** The value function $\mathcal{I}$ assigns values to formulae: (recursively)
  - $\mathcal{I}(f(\mathsf{A}^1, \ldots, \mathsf{A}^k)) := \mathcal{I}(f)(\mathcal{I}(\mathsf{A}^1), \ldots, \mathcal{I}(\mathsf{A}^k))$
  - $\mathcal{I}(p(\mathsf{A}^1, \ldots, \mathsf{A}^k)) := \mathsf{T}$, iff $\langle \mathcal{I}(\mathsf{A}^1), \ldots, \mathcal{I}(\mathsf{A}^k) \rangle \in \mathcal{I}(p)$
  - $\mathcal{I}(\neg\mathsf{A}) = \mathcal{I}(\neg)(\mathcal{I}(\mathsf{A}))$ and $\mathcal{I}(\mathsf{A} \wedge \mathsf{B}) = \mathcal{I}(\wedge)(\mathcal{I}(\mathsf{A}), \mathcal{I}(\mathsf{G}))$ (just as in PL^0)
- **Definition 5.7.** Model: $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$ varies in $\mathcal{D}_\iota$ and $\mathcal{I}$.
- **Theorem 5.8.** $\mathrm{PL}^{\mathrm{nq}}$ *is isomorphic to* PL^0 *(interpret atoms as prop. variables)*

SOME RIGHTS RESERVED

A Model for $\mathrm{PL}^{\mathrm{nq}}$

- **Example 5.9.** Let $L := \{a, b, c, d, e, P, Q, R, S\}$, we set the universe $\mathcal{D} := \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$, and specify the interpretation function $\mathcal{I}$ by setting
  - $a \mapsto \clubsuit$, $b \mapsto \spadesuit$, $c \mapsto \heartsuit$, $d \mapsto \diamondsuit$, and $e \mapsto \diamondsuit$ for constants,
  - $P \mapsto \{\clubsuit, \spadesuit\}$ and $Q \mapsto \{\spadesuit, \diamondsuit\}$, for unary predicate constants.
  - $R \mapsto \{\langle\heartsuit, \diamondsuit\rangle, \langle\diamondsuit, \heartsuit\rangle\}$, and $S \mapsto \{\langle\diamondsuit, \spadesuit\rangle, \langle\spadesuit, \clubsuit\rangle\}$ for binary predicate constants.
- **Example 5.10 (Computing Meaning in this Model).**
  - $\mathcal{I}(R(a,b) \wedge P(c)) = \mathsf{T}$, iff
  - $\mathcal{I}(R(a,b)) = \mathsf{T}$ and $\mathcal{I}(P(c)) = \mathsf{T}$, iff
  - $\langle\mathcal{I}(a), \mathcal{I}(b)\rangle \in \mathcal{I}(R)$ and $\mathcal{I}(c) \in \mathcal{I}(P)$, iff
  - $\langle\clubsuit, \spadesuit\rangle \in \{\langle\heartsuit, \diamondsuit\rangle, \langle\diamondsuit, \heartsuit\rangle\}$ and $\heartsuit \in \{\clubsuit, \spadesuit\}$

  So, $\mathcal{I}(R(a,b) \wedge P(c)) = \mathsf{F}$.

SOME RIGHTS RESERVED

$\mathrm{PL}^{\mathrm{nq}}$ and PL^0 are Isomorphic

- **Observation:** For every choice of Σ of signature, the set $\mathcal{A}_\Sigma$ of atomic $\mathrm{PL}^{\mathrm{nq}}$ formulae is countable, so there is a $\mathcal{V}_\Sigma \subseteq \mathcal{V}_0$ and a bijection $\theta_\Sigma \colon \mathcal{A}_\Sigma \to \mathcal{V}_\Sigma$.
  θ_Σ can be extended to a bijection on formulae as $\mathrm{PL}^{\mathrm{nq}}$ and PL^0 share connectives.
- **Lemma 5.11.** *For every model $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$, there is a variable assignment $\varphi_\mathcal{M}$, such that $\mathcal{I}_{\varphi_\mathcal{M}}(\mathsf{A}) = \mathcal{I}(\mathsf{A})$.*
- *Proof sketch:* We just define $\varphi_\mathcal{M}(X) := \mathcal{I}(\theta_\Sigma^{-1}(X))$, then the assertion follows by induction on A.
- **Lemma 5.12.** *For every variable assignment $\psi \colon \mathcal{V}_\Sigma \to \{\mathsf{T}, \mathsf{F}\}$ there is a model $\mathcal{M}^\psi = \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle$, such that $\mathcal{I}_\psi(\mathsf{A}) = \mathcal{I}^\psi(\mathsf{A})$.*
- *Proof sketch:* see next slide
- **Corollary 5.13.** *$\mathrm{PL}^{\mathrm{nq}}$ is isomorphic to PL^0, i.e. the following diagram commutes:*

$$\begin{tikzcd}
\langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle & \mathcal{V}_\Sigma \to \{\mathsf{T}, \mathsf{F}\} \arrow[l, "\psi \mapsto \mathcal{M}^\psi"'] \\
\mathrm{PL}^{\mathrm{nq}}(\Sigma) \arrow[u, "\mathcal{I}^\psi()"] \arrow[r, "\theta_\Sigma"] & \mathrm{PL}^0(\mathcal{A}_\Sigma) \arrow[u, "\mathcal{I}_{\varphi_\mathcal{M}}()"']
\end{tikzcd}$$

- **Note:** This constellation with a language isomorphism and a corresponding model isomorphism (in converse direction) is typical for a logic isomorphism.

SOME RIGHTS RESERVED

Valuation and Satisfiability

- **Lemma 5.14.** *For every variable assignment* $\psi\colon \mathcal{V}_\Sigma \to \{\mathsf{T},\mathsf{F}\}$ *there is a model* $\mathcal{M}^\psi = \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle$*, such that* $\mathcal{I}_\psi(\mathsf{A}) = \mathcal{I}^\psi(\mathsf{A})$*.*
- *Proof:* We construct $\mathcal{M}^\psi = \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle$ and show that it works as desired.

  1. Let $\mathcal{D}^\psi$ be the set of $\mathrm{PL}^{\mathrm{nq}}$ terms over Σ, and
     - $\mathcal{I}^\psi(f)\colon {\mathcal{D}^\psi}^k \to \mathcal{D}^\psi\,;\, \langle \mathsf{A}_1,\ldots,\mathsf{A}_k \rangle \mapsto f(\mathsf{A}_1,\ldots,\mathsf{A}_k)$ for $f \in \Sigma^f_k$
     - $\mathcal{I}^\psi(p) := \{\langle \mathsf{A}_1,\ldots,\mathsf{A}_k \rangle \mid \psi(\theta_\psi^{-1} p(\mathsf{A}_1,\ldots,\mathsf{A}_k)) = \mathsf{T}\}$ for $p \in {\Sigma^p}_k$.

  2. We show $\mathcal{I}^\psi(\mathsf{A}) = \mathsf{A}$ for terms A by induction on A

     2.1. If $\mathsf{A} = c$, then $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}^\psi(c) = c = \mathsf{A}$

     2.2. If $\mathsf{A} = f(\mathsf{A}_1,\ldots,\mathsf{A}_n)$ then
     $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}^\psi(f)(\mathcal{I}(\mathsf{A}_1),\ldots,\mathcal{I}(\mathsf{A}_n)) = \mathcal{I}^\psi(f)(\mathsf{A}_1,\ldots,\mathsf{A}_k) = \mathsf{A}$.

  4. For a $\mathrm{PL}^{\mathrm{nq}}$ formula A we show that $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}_\psi(\mathsf{A})$ by induction on A.

     4.1. If $\mathsf{A} = p(\mathsf{A}_1,\ldots,\mathsf{A}_k)$, then $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}^\psi(p)(\mathcal{I}(\mathsf{A}_1),\ldots,\mathcal{I}(\mathsf{A}_n)) = \mathsf{T}$, iff $\langle \mathsf{A}_1,\ldots,\mathsf{A}_k \rangle \in \mathcal{I}^\psi(p)$, iff $\psi(\theta_\psi^{-1}\mathsf{A}) = \mathsf{T}$, so $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}_\psi(\mathsf{A})$ as desired.

     4.2. If $\mathsf{A} = \neg\mathsf{B}$, then $\mathcal{I}^\psi(\mathsf{A}) = \mathsf{T}$, iff $\mathcal{I}^\psi(\mathsf{B}) = \mathsf{F}$, iff $\mathcal{I}^\psi(\mathsf{B}) = \mathcal{I}_\psi(\mathsf{B})$, iff $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}_\psi(\mathsf{A})$.

     4.3. If $\mathsf{A} = \mathsf{B} \wedge \mathsf{C}$ then we argue similarly

  6. Hence $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}_\psi(\mathsf{A})$ for all $\mathrm{PL}^{\mathrm{nq}}$ formulae and we have concluded the proof.

$\square$

SOME RIGHTS RESERVED

10.6 Conclusion

Summary

- Sometimes, it pays off to think before acting.
- In AI, “thinking” is implemented in terms of reasoning to deduce new knowledge from a knowledge base represented in a suitable logic.
- Logic prescribes a syntax for formulas, as well as a semantics prescribing which interpretations satisfy them. A entails B if all interpretations that satisfy A also satisfy B. Deduction is the process of deriving new entailed formulae.
- Propositional logic formulae are built from atomic propositions, with the connectives “*and*”, “*or*”, “*not*”.

SOME RIGHTS RESERVED

Issues with Propositional Logic

- **Time:** For things that change (e.g., Wumpus moving according to certain rules), we need time-indexed propositions (like, $S_{2,1}^{t=7}$) to represent validity over time $\leadsto$ further expansion of the rules.
- **Can we design a more human-like logic?:** Yep
  - Predicate logic: quantification of variables ranging over individuals. (cf. and)
  - ...and a whole zoo of logics much more powerful still.
  - Note: In applications, propositional CNF are generated by computer programs. This mitigates (but does not remove!) the inconveniences of propositional modeling.

SOME RIGHTS RESERVED

Chapter 11
Formal Systems: Syntax, Semantics, Entailment, and Derivation in General

Recap: General Aspects of Propositional Logic

- **There are many ways to define Propositional Logic:**
  - We chose $\wedge$ and $\neg$ as primitive, and many others as defined.
  - We could have used $\vee$ and $\neg$ just as well.
  - We could even have used only one connective e.g. negated conjunction $\uparrow$ or disjunction $\downarrow$ and defined $\wedge$, $\vee$, and $\neg$ via $\uparrow$ and $\downarrow$ respectively.

$\uparrow$	$\top$	$\bot$
$\top$	F	T
$\bot$	T	T

$\downarrow$	$\top$	$\bot$
$\top$	F	F
$\bot$	F	T

$\neg a$	$a \uparrow a$	$a \downarrow a$
ab	$a \uparrow b \uparrow a \uparrow b$	$a \downarrow ab \downarrow b$
ab	$a \uparrow a \uparrow b \uparrow b$	$a \downarrow b \downarrow a \downarrow b$

- **Observation:** The set $wf\!f_0(\mathcal{V}_0)$ of well-formed propositional formulae is a formal language over the alphabet given by $\mathcal{V}_0$, the connectives, and brackets.
- **Recall:** We are mostly interested in
  - satisfiability i.e. whether $\mathcal{M} \models$ A, and
  - entailment i.e whether A $\models$ B.
- **Observation:** In particular, the inductive/compositional nature of $wf\!f_0(\mathcal{V}_0)$ and $\mathcal{I}_\varphi \colon wf\!f_0(\mathcal{V}_0) \to \mathcal{D}_0$ are secondary.
- **Idea:** Concentrate on language, models $(\mathcal{M}, \varphi)$, and satisfiability.

SOME RIGHTS RESERVED

Logical Systems

- **Definition 0.1.** A logical system (or simply a logic) is a triple $\mathcal{S} := \langle \mathcal{L}, \mathcal{M}, \models \rangle$, where
  1. $\mathcal{L}$ is a set of propositions,
  2. $\mathcal{M}$ a set of models, and
  3. a relation $\models \subseteq \mathcal{M} \times \mathcal{L}$ called the satisfaction relation. We read $\mathcal{M} \models A$ as $\mathcal{M}$ satisfies A and correspondingly $\mathcal{M} \not\models A$ as $\mathcal{M}$ falsifies A.
- **Example 0.2 (Propositional Logic).** $\langle wff(\Sigma_{PL^0}, \mathcal{V}_{PL^0}), \mathcal{K}_o, \models \rangle$ is a logical system, if we define $\mathcal{K}_o := \mathcal{V}_0 \rightharpoonup \mathcal{D}_0$ (the set of variable assignments) and $\varphi \models \mathrm{A}$ iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathsf{T}$.
- **Definition 0.3.** Let $\langle \mathcal{L}, \mathcal{M}, \models \rangle$ be a logical system, $\mathrm{M} \in \mathcal{M}$ a model and $\mathrm{A} \in \mathcal{L}$ a proposition. Then we say that A is
  - satisfied by M iff $\mathrm{M} \models \mathrm{A}$.
  - satisfiable iff A is satisfied by some model.
  - unsatisfiable iff A is not satisfiable.
  - falsified by M iff $\mathrm{M} \not\models \mathrm{A}$.
  - valid or unfalsifiable (write $\models \mathrm{A}$) iff A is satisfied by every model.
  - invalid or falsifiable (write $\not\models \mathrm{A}$) iff A is not valid.

SOME RIGHTS RESERVED

Derivation Relations and Inference Rules

- **Definition 0.4.** Let $\mathcal{L}$ be a formal language, then we call a relation $\vdash\subseteq\mathcal{P}(\mathcal{L})\times\mathcal{L}$ a derivation relation for $\mathcal{L}$, if
  - $\mathcal{H}\vdash A$, if $A\in\mathcal{H}$ ($\vdash$ is proof reflexive),
  - $\mathcal{H}\vdash A$ and $(\mathcal{H}'\cup\{A\})\vdash B$ imply $(\mathcal{H}\cup\mathcal{H}')\vdash B$ ($\vdash$ is proof transitive),
  - $\mathcal{H}\vdash A$ and $\mathcal{H}\subseteq\mathcal{H}'$ imply $\mathcal{H}'\vdash A$ ($\vdash$ is monotonic or admits weakening).

SOME RIGHTS RESERVED

Derivation Relations and Inference Rules

- **Definition 0.8.** Let $\mathcal{L}$ be a formal language, then we call a relation $\vdash \subseteq \mathcal{P}(\mathcal{L}) \times \mathcal{L}$ a derivation relation for $\mathcal{L}$, if
  - $\mathcal{H} \vdash A$, if $A \in \mathcal{H}$ ($\vdash$ is proof reflexive),
  - $\mathcal{H} \vdash A$ and $(\mathcal{H}' \cup \{A\}) \vdash B$ imply $(\mathcal{H} \cup \mathcal{H}') \vdash B$ ($\vdash$ is proof transitive),
  - $\mathcal{H} \vdash A$ and $\mathcal{H} \subseteq \mathcal{H}'$ imply $\mathcal{H}' \vdash A$ ($\vdash$ is monotonic or admits weakening).
- **Definition 0.9.** Let $\mathcal{L}$ be a formal language, then an inference rule over $\mathcal{L}$ is a decidable $n+1$ ary relation on $\mathcal{L}$. Inference rules are traditionally written as

  $$\frac{A_1 \quad \ldots \quad A_n}{C}\,\mathcal{N}$$

  where $A_1, \ldots, A_n$ and C are schemata for words in $\mathcal{L}$ and $\mathcal{N}$ is a name. The A_i are called assumptions of $\mathcal{N}$, and C is called its conclusion.
  Any $n+1$-tuple

  $$\frac{a_1 \quad \ldots \quad a_n}{c}$$

  in $\mathcal{N}$ is called an application of $\mathcal{N}$ and we say that we apply $\mathcal{N}$ to a set M of words with $a_1, \ldots, a_n \in M$ to obtain c.
- **Definition 0.10.** An inference rule without assumptions is called an axiom.
- **Definition 0.11.** A calculus (or inference system) is a formal language $\mathcal{L}$ equipped with a set $\mathcal{C}$ of inference rules over $\mathcal{L}$.

SOME RIGHTS RESERVED

Derivations

- **Definition 0.12.** Let $\mathcal{L} := \langle \mathcal{L}, \models \rangle$ be a logical system and $\mathcal{C}$ a calculus for $\mathcal{L}$, then a $\mathcal{C}$-derivation of a proposition $\mathrm{C} \in \mathcal{L}$ from a set $\mathcal{H} \subseteq \mathcal{L}$ of hypotheses (write $\mathcal{H} \vdash_{\mathcal{C}} \mathrm{C}$) is a sequence $\mathrm{A}_1, \ldots, \mathrm{A}_m$ of propositions
  - $\mathrm{A}_m = \mathrm{C}$, (derivation culminates in C)
  - for all $1 \leq i \leq m$, either $\mathrm{A}_i \in \mathcal{H}$, or (hypothesis)
  - there is an inference rule $\dfrac{\mathrm{A}_{l_1} \quad \ldots \quad \mathrm{A}_{l_k}}{\mathrm{A}_i}$ in $\mathcal{C}$ with $l_j < i$ for all $j \leq k$. (rule application)

  We can also see a derivation as a derivation tree, where the A_{l_j} are the children of the node A_i.

- **Example 0.13.**

In the propositional Hilbert calculus $\mathcal{H}^0$ we have the derivation $P \vdash_{\mathcal{H}^0} Q \Rightarrow P$: the sequence is $P \Rightarrow Q \Rightarrow P, P, Q \Rightarrow P$ and the corresponding tree on the right.

$$\dfrac{\dfrac{}{P \Rightarrow Q \Rightarrow P}\,K \qquad P}{Q \Rightarrow P}\,MP$$

SOME RIGHTS RESERVED

Formal Systems

- Let $\langle\mathcal{L},\vDash\rangle$ be a logical system and $\mathcal{C}$ a calculus, then $\vdash_{\mathcal{C}}$ is a derivation relation and thus $\langle\mathcal{L},\models,\vDash,\vdash_{\mathcal{C}}\rangle$ a derivation system.
- Therefore we will sometimes also call $\langle\mathcal{L},\mathcal{C},\vDash\rangle$ a formal system, iff $\mathcal{L}:=\langle\mathcal{L},\vDash\rangle$ is a logical system, and $\mathcal{C}$ a calculus for $\mathcal{L}$.
- **Definition 0.14.** Let $\mathcal{C}$ be a calculus, then a $\mathcal{C}$-derivation $\emptyset\vdash_{\mathcal{C}}\mathrm{A}$ is called a proof of A and if one exists (write $\vdash_{\mathcal{C}}\mathrm{A}$) then A is called a $\mathcal{C}$-theorem.
  Definition 0.15. The act of finding a proof for A is called proving A.
- **Definition 0.16.** An inference rule $\mathcal{I}$ is called admissible in a calculus $\mathcal{C}$, if the extension of $\mathcal{C}$ by $\mathcal{I}$ does not yield new theorems.
- **Definition 0.17.** An inference rule

  $$\frac{\mathrm{A}_1 \quad \ldots \quad \mathrm{A}_n}{\mathrm{C}}$$

  is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $\mathrm{A}_1,\ldots,\mathrm{A}_n\vdash_{\mathcal{C}}\mathrm{C}$.
- **Observation 0.18.** *Derivable inference rules are admissible, but not the other way around.*

SOME RIGHTS RESERVED

Chapter 12
Machine-Oriented Calculi for Propositional Logic

SOME RIGHTS RESERVED

12.1 Test Calculi

Automated Deduction as an Agent Inference Procedure

- **Recall:** Our knowledge of the cave entails a definite Wumpus position! (slide 320)
- **Problem:** That was human reasoning, can we build an agent function that does this?
- **Answer:** As for constraint networks, we use inference, here resolution/tableaux.

SOME RIGHTS RESERVED

Unsatisfiability Theorem

- **Theorem 1.1 (Unsatisfiability Theorem).** *$\mathcal{H} \vDash A$ iff $\mathcal{H} \cup \{\neg A\}$ is unsatisfiable.*

- *Proof:* We prove both directions separately

  1. "$\Rightarrow$": Say $\mathcal{H} \vDash A$
     1.1. For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \models A$ and thus $\varphi \not\models (\neg A)$.
  3. "$\Leftarrow$": Say $\mathcal{H} \cup \{\neg A\}$ is unsatisfiable.
     3.1. For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \not\models (\neg A)$ and thus $\varphi \models A$.

☐

- **Observation 1.2.** *Entailment can be tested via satisfiability.*

SOME RIGHTS RESERVED

Test Calculi: A Paradigm for Automating Inference

- **Definition 1.3.** Given a formal system $\langle\mathcal{L},\mathcal{C},\mathcal{M},\models\rangle$, the task of theorem proving consists in determining whether $\mathcal{H}\vdash_{\mathcal{C}}C$ for a conjecture $C\in\mathcal{L}$ and hypotheses $\mathcal{H}\subseteq\mathcal{L}$.
- **Definition 1.4.** Automated theorem proving (ATP) is the automation of theorem proving

SOME RIGHTS RESERVED

Test Calculi: A Paradigm for Automating Inference

- **Definition 1.6.** Given a formal system $\langle\mathcal{L},\mathcal{C},\mathcal{M},\models\rangle$, the task of theorem proving consists in determining whether $\mathcal{H}\vdash_{\mathcal{C}}C$ for a conjecture $C\in\mathcal{L}$ and hypotheses $\mathcal{H}\subseteq\mathcal{L}$.
- **Definition 1.7.** Automated theorem proving (ATP) is the automation of theorem proving
- **Idea:** A set $\mathcal{H}$ of hypotheses and a conjecture A induce a search problem $\Pi_{\mathcal{C}}^{\mathcal{H}\models\mathsf{A}}:=\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, where the states $\mathcal{S}$ are sets of formulae, the actions $\mathcal{A}$ are the inference rules from $\mathcal{C}$, the initial state $\mathcal{I}=\mathcal{H}$, and the goal states are those with $\mathsf{A}\in\mathcal{S}$.
- **Problem:** ATP as a search problem does not admit good heuristics, since these need to take the conjecture $\mathcal{A}$ into account.

SOME RIGHTS RESERVED

Test Calculi: A Paradigm for Automating Inference

- **Definition 1.9.** Given a formal system $\langle\mathcal{L},\mathcal{C},\mathcal{M},\models\rangle$, the task of theorem proving consists in determining whether $\mathcal{H}\vdash_{\mathcal{C}}C$ for a conjecture $C\in\mathcal{L}$ and hypotheses $\mathcal{H}\subseteq\mathcal{L}$.
- **Definition 1.10.** Automated theorem proving (ATP) is the automation of theorem proving
- **Idea:** A set $\mathcal{H}$ of hypotheses and a conjecture A induce a search problem $\Pi_{\mathcal{C}}^{\mathcal{H}\models\mathrm{A}}:=\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, where the states $\mathcal{S}$ are sets of formulae, the actions $\mathcal{A}$ are the inference rules from $\mathcal{C}$, the initial state $\mathcal{I}=\mathcal{H}$, and the goal states are those with $\mathrm{A}\in\mathcal{S}$.
- **Problem:** ATP as a search problem does not admit good heuristics, since these need to take the conjecture $\mathcal{A}$ into account.
- **Idea:** Turn the search around – using the unsatisfiability theorem (1.1).
- **Definition 1.11.** For a given conjecture A and hypotheses $\mathcal{H}$ a test calculus $\mathcal{T}$ tries to derive a refutation $\mathcal{H},\overline{A}\vdash_{\mathcal{T}}\bot$ instead of $\mathcal{H}\vdash A$, where $\overline{A}$ is unsatisfiable iff A is valid and $\bot$, an "obviously" unsatisfiable proposition.

SOME RIGHTS RESERVED

Test Calculi: A Paradigm for Automating Inference

- **Definition 1.12.** Given a formal system $\langle\mathcal{L},\mathcal{C},\mathcal{M},\models\rangle$, the task of theorem proving consists in determining whether $\mathcal{H}\vdash_{\mathcal{C}}C$ for a conjecture $C\in\mathcal{L}$ and hypotheses $\mathcal{H}\subseteq\mathcal{L}$.
- **Definition 1.13.** Automated theorem proving (ATP) is the automation of theorem proving
- **Idea:** A set $\mathcal{H}$ of hypotheses and a conjecture A induce a search problem $\Pi_{\mathcal{C}}^{\mathcal{H}\models\mathsf{A}}:=\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, where the states $\mathcal{S}$ are sets of formulae, the actions $\mathcal{A}$ are the inference rules from $\mathcal{C}$, the initial state $\mathcal{I}=\mathcal{H}$, and the goal states are those with $\mathsf{A}\in\mathcal{S}$.
- **Problem:** ATP as a search problem does not admit good heuristics, since these need to take the conjecture $\mathcal{A}$ into account.
- **Idea:** Turn the search around – using the unsatisfiability theorem (1.1).
- **Definition 1.14.** For a given conjecture A and hypotheses $\mathcal{H}$ a test calculus $\mathcal{T}$ tries to derive a refutation $\mathcal{H},\overline{A}\vdash_{\mathcal{T}}\bot$ instead of $\mathcal{H}\vdash A$, where $\overline{A}$ is unsatisfiable iff A is valid and $\bot$, an "obviously" unsatisfiable proposition.
- **Observation:** A test calculus $\mathcal{C}$ induces a search problem where the initial state is $\mathcal{H}\cup\{\neg\mathsf{A}\}$ and $S\in\mathcal{S}$ is a goal state iff $\bot\in S$. (proximity of $\bot$ easier for heuristics)
- Searching for $\bot$ admits simple heuristics, e.g. size reduction. ($\bot$ minimal)

12.1.1 Normal Forms

SOME RIGHTS RESERVED

Recap: Atoms and Literals

- **Definition 1.15.** A formula is called atomic (or an atom) if it does not contain logical constants, else it is called complex.
- **Definition 1.16.** Let $\langle\mathcal{L},\mathcal{M},\models\rangle$ be a logical system, $\mathrm{A}\in\mathcal{L}$, A a label set, and $\alpha\in A$ a label, then we call a pair a labeled formula and write it as A^α. For a set Φ of propositions we use $\Phi^\alpha{:=}\{\mathrm{A}^\alpha\,|\,\mathrm{A}\in\Phi\}$.

  Definition 1.17. If the label set is $\mathbb{B}$, we call a labeled formula A^{T} positive and A^{F} negative.

  Definition 1.18. Let $\langle\mathcal{L},\mathcal{M},\models\rangle$ be a logical system and A^α a labeled formula. Then we say that $\mathcal{M}\in\mathcal{M}$ satisfies A^α (written $\mathcal{M}|{\models}\mathrm{A}$), iff $\alpha=\mathsf{T}$ and $\mathcal{M}\models\mathrm{A}$ or $\alpha=\mathsf{F}$ and $\mathcal{M}\not\models\mathrm{A}$.
- **Definition 1.19.** Let $\langle\mathcal{L},\mathcal{M},\models\rangle$ be a logical system, $\mathrm{A}\in\mathcal{L}$ atomic, and $\alpha\in\{\mathsf{T},\mathsf{F}\}$, then we call a A^α a literal.
- **Intuition:** To satisfy a formula, we make it "true". To satisfy a labeled formula A^α, it must have the truth value α.
- **Definition 1.20.** For a literal A^α, we call the literal A^β with $\alpha\neq\beta$ the opposite literal (or partner literal).

SOME RIGHTS RESERVED

Alternative Definition: Literals

- **Note:** Literals are often defined without recurring to labeled formulae:
- **Definition 1.21.** A literal is an atom A (positive literal) or negated atom ¬A (negative literal). A and ¬A are opposite literals.
- **Note:** This notion of literal is equivalent to the labeled formulae-notion of literal, but does not generalize as well to logics with more than two truth values.

SOME RIGHTS RESERVED

Normal Forms

- There are two quintessential normal forms for propositional formulae: (there are others as well)
- **Definition 1.22.** A formula is in conjunctive normal form (CNF) if it is T or a conjunction of disjunctions of literals: i.e. if it is of the form $\bigwedge_{i=1}^{n}\bigvee_{j=1}^{m_i} l_{ij}$
- **Definition 1.23.** A formula is in disjunctive normal form (DNF) if it is F or a disjunction of conjunctions of literals: i.e. if it is of the form $\bigvee_{i=1}^{n}\bigwedge_{j=1}^{m_i} l_{ij}$
- **Observation 1.24.** *Every formula has equivalent formulae in CNF and DNF.*

SOME RIGHTS RESERVED

12.2 Analytical Tableaux

12.2.1 Analytical Tableaux

SOME RIGHTS RESERVED

Test Calculi: Tableaux and Model Generation

- **Idea:** A tableau calculus is a test calculus that
  - analyzes a labeled formulae in a tree to determine satisfiability,
  - its branches correspond to valuations ($\leadsto$ models).
- **Example 2.1.** Tableau calculi try to construct models for labeled formulae: E.g. the propositional tableau calculus for PL^0

Tableau refutation (Validity)	Model generation (Satisfiability)
$\models P \wedge Q \Rightarrow Q \wedge P$	$\models P \wedge (Q \vee \neg R) \wedge \neg Q$
$(P \wedge Q \Rightarrow Q \wedge P)^{\mathsf{F}}$ $(P \wedge Q)^{\mathsf{T}}$ $(Q \wedge P)^{\mathsf{F}}$ P^{T} Q^{T} $P^{\mathsf{F}} \mid Q^{\mathsf{F}}$ $\bot \mid \bot$	$(P \wedge (Q \vee \neg R) \wedge \neg Q)^{\mathsf{T}}$ $(P \wedge (Q \vee \neg R))^{\mathsf{T}}$ $\neg Q^{\mathsf{T}}$ Q^{F} P^{T} $(Q \vee \neg R)^{\mathsf{T}}$ $Q^{\mathsf{T}} \mid \neg R^{\mathsf{T}}$ $\bot \mid R^{\mathsf{F}}$
No Model	Herbrand valuation $\{P^{\mathsf{T}}, Q^{\mathsf{F}}, R^{\mathsf{F}}\}$ $\varphi := \{P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}, R \mapsto \mathsf{F}\}$

- **Idea:** Open branches in saturated tableaux yield satisfying assignments.
- **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)
  - Satisfiable, iff there are open branches (correspond to models)

SOME RIGHTS RESERVED

Analytical Tableaux (Formal Treatment of $\mathcal{T}_0$)

- **Idea:** A test calculus where
  - A labeled formula is analyzed in a tree to determine satisfiability,
  - branches correspond to valuations (models)
- **Definition 2.2.** The propositional tableau calculus $\mathcal{T}_0$ has two inference rules per connective (one for each possible label)

$$\frac{(A\wedge B)^{T}}{\begin{array}{c}A^{T}\\ B^{T}\end{array}}\,\mathcal{T}_0\wedge \qquad \frac{(A\wedge B)^{F}}{A^{F}\;\Big|\;B^{F}}\,\mathcal{T}_0\vee \qquad \frac{\neg A^{T}}{A^{F}}\,\mathcal{T}_0\neg^{T} \qquad \frac{\neg A^{F}}{A^{T}}\,\mathcal{T}_0\neg^{F} \qquad \frac{\begin{array}{c}A^{\alpha}\\ A^{\beta}\end{array}\quad \alpha\neq\beta}{\bot}\,\mathcal{T}_0\bot$$

  Use rules exhaustively as long as they contribute new material ($\rightsquigarrow$ termination)

- **Definition 2.3.** We call any tree ($\Big|$ introduces branches) produced by the $\mathcal{T}_0$ inference rules from a set Φ of labeled formulae a tableau for Φ.
- **Definition 2.4.** Call a tableau saturated, iff no rule adds new material and a branch closed, iff it ends in $\bot$, else open. A tableau is closed, iff all of its branches are.

  In analogy to the $\bot$ at the end of closed branches, we sometimes decorate open branches with a $\Box$ symbol.

SOME RIGHTS RESERVED

Analytical Tableaux ($\mathcal{T}_0$ continued)

- **Definition 2.6 ($\mathcal{T}_0$-Theorem/Derivability).** A is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0}$A), iff there is a closed tableau with A^F at the root.
  $\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ derives A in $\mathcal{T}_0$ ($\Phi \vdash_{\mathcal{T}_0} A$), iff there is a closed tableau starting with A^F and Φ^T. The tableau with only a branch of A^F and Φ^T is called initial for $\Phi \vdash_{\mathcal{T}_0} A$.

SOME RIGHTS RESERVED

A Valid Real-World Example

- **Example 2.8.** "*If Mary loves Bill and John loves Mary, then John loves Mary*"

$$
\begin{array}{c}
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}} \\
\neg(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary})) \wedge \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}} \\
(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary})) \wedge \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}} \\
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\neg\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}} \\
\bot
\end{array}
$$

This is a closed tableau, so the $\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary})$ is a $\mathcal{T}_0$-theorem. As we will see, $\mathcal{T}_0$ is sound and complete, so

$$\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary})$$

is valid.

Deriving Entailment in $\mathcal{T}_0$

- **Example 2.9.** "*Mary loves Bill*" and "*John loves Mary*" together entail that "*John loves Mary*"

$$\begin{array}{c} \mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} \\ \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\ \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}} \\ \bot \end{array}$$

This is a closed tableau, so $\{\mathrm{loves}(\mathrm{mary},\mathrm{bill}),\mathrm{loves}(\mathrm{john},\mathrm{mary})\}\vdash_{\mathcal{T}_0}\mathrm{loves}(\mathrm{john},\mathrm{mary})$.
Again, as $\mathcal{T}_0$ is sound and complete we have

$$\{\mathrm{loves}(\mathrm{mary},\mathrm{bill}),\mathrm{loves}(\mathrm{john},\mathrm{mary})\}\models\mathrm{loves}(\mathrm{john},\mathrm{mary})$$

A Falsifiable Real-World Example

- **Example 2.10.** * "*If Mary loves Bill or John loves Mary, then John loves Mary*"
  Try proving the implication (this fails)

$$
\begin{array}{c}
((\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))\Rightarrow\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}\\
\lnot(\lnot\lnot(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))\land\lnot\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}\\
(\lnot\lnot(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))\land\lnot\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}\\
\lnot\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}\\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}\\
\lnot\lnot(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}\\
\lnot(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}\\
(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}\\
\begin{array}{c|c}
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} & \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}\\
 & \bot
\end{array}
\end{array}
$$

Indeed we can make $\mathcal{I}_{\varphi}(\mathrm{loves}(\mathrm{mary},\mathrm{bill}))=\mathsf{T}$ but $\mathcal{I}_{\varphi}(\mathrm{loves}(\mathrm{john},\mathrm{mary}))=\mathsf{F}$.

Testing for Entailment in $\mathcal{T}_0$

- **Example 2.11.** Does "*Mary loves Bill or John loves Mary*" entail that "*John loves Mary*"?

$$\begin{array}{c}
(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\vee\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}\\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}\\
\begin{array}{c|c}
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} & \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}\\
 & \bot
\end{array}
\end{array}$$

This saturated tableau has an open branch that shows that the interpretation with $\mathcal{I}_\varphi(\mathrm{loves}(\mathrm{mary},\mathrm{bill}))=\mathsf{T}$ but $\mathcal{I}_\varphi(\mathrm{loves}(\mathrm{john},\mathrm{mary}))=\mathsf{F}$ falsifies the derivability/entailment conjecture.

SOME RIGHTS RESERVED

12.2.2 Practical Enhancements for Tableaux

Derivable Rules of Inference

- **Definition 2.12.** An inference rule

$$\frac{A_1 \quad \ldots \quad A_n}{C}$$

is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $A_1, \ldots, A_n \vdash_{\mathcal{C}} C$.

- **Definition 2.13.** We have the following derivable inference rules in $\mathcal{T}_0$:

$$\frac{(A \Rightarrow B)^T}{\begin{array}{c|c} A^F & B^T \end{array}} \qquad \frac{(A \Rightarrow B)^F}{\begin{array}{c} A^T \\ B^F \end{array}} \qquad \frac{\begin{array}{c} A^T \\ (A \Rightarrow B)^T \end{array}}{B^T}$$

$$\frac{(A \vee B)^T}{\begin{array}{c|c} A^T & B^T \end{array}} \qquad \frac{(A \vee B)^F}{\begin{array}{c} A^F \\ B^F \end{array}} \qquad \frac{(A \Leftrightarrow B)^T}{\begin{array}{c|c} A^T & A^F \\ B^T & B^F \end{array}} \qquad \frac{(A \Leftrightarrow B)^F}{\begin{array}{c|c} A^T & A^F \\ B^F & B^T \end{array}}$$

$$\begin{array}{c} A^T \\ (A \Rightarrow B)^T \\ (\neg A \vee B)^T \\ \neg(\neg\neg A \wedge \neg B)^T \\ (\neg\neg A \wedge \neg B)^F \\ \begin{array}{c|c} \neg\neg A^F & \neg B^F \\ \neg A^T & B^T \\ A^F & \\ \bot & \end{array} \end{array}$$

SOME RIGHTS RESERVED

Tableaux with Derivable Rules (example)

Example 2.14.

$$
\begin{array}{c}
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}} \\
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}} \\
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\
\bot
\end{array}
$$

SOME RIGHTS RESERVED

12.2.3 Soundness and Termination of Tableaux

SOME RIGHTS RESERVED

Soundness (Tableau)

- **Idea:** A test calculus is refutation sound, iff its inference rules preserve satisfiability and the goal formulae are unsatisfiable.
- **Definition 2.15.** A labeled formula A^α is valid under φ, iff $\mathcal{I}_\varphi(A) = \alpha$.
- **Definition 2.16.** A tableau $\mathcal{T}$ is satisfiable, iff there is a satisfiable branch $\mathcal{P}$ in $\mathcal{T}$, i.e. if the set of formulae on $\mathcal{P}$ is satisfiable.
- **Lemma 2.17.** *$\mathcal{T}_0$ rules transform satisfiable tableaux into satisfiable ones.*
- **Theorem 2.18 (Soundness).** *$\mathcal{T}_0$ is sound, i.e. $\Phi \subseteq wff_0(\mathcal{V}_0)$ valid, if there is a closed tableau $\mathcal{T}$ for Φ^F.*
- *Proof:* by contradiction
  1. Suppose Φ is falsifiable $\hat{=}$ not valid.
  2. Then the initial tableau is satisfiable, (Φ^F satisfiable)
  3. so $\mathcal{T}$ is satisfiable, by 2.17.
  4. Thus there is a satisfiable branch (by definition)
  5. but all branches are closed ($\mathcal{T}$ closed)

  □
- **Theorem 2.19 (Completeness).** *$\mathcal{T}_0$ is complete, i.e. if $\Phi \subseteq wff_0(\mathcal{V}_0)$ is valid, then there is a closed tableau $\mathcal{T}$ for Φ^F.*
- *Proof sketch:* Proof difficult/interesting; see ???

SOME RIGHTS RESERVED

Termination for Tableaux

- **Lemma 2.20.** *$\mathcal{T}_0$ terminates, i.e. every $\mathcal{T}_0$ tableau becomes saturated after finitely many rule applications.*

Termination for Tableaux

- **Lemma 2.22.** *$\mathcal{T}_0$ terminates, i.e. every $\mathcal{T}_0$ tableau becomes saturated after finitely many rule applications.*
- *Proof:* By examining the rules wrt. a measure μ
  1. Let us call a labeled formulae A^{α} worked off in a tableau $\mathcal{T}$, if a $\mathcal{T}_0$ rule has already been applied to it.
  2. It is easy to see that applying rules to worked off formulae will only add formulae that are already present in its branch.
  3. Let $\mu(\mathcal{T})$ be the number of connectives in labeled formulae in $\mathcal{T}$ that are not worked off.
  4. Then each rule application to a labeled formula in $\mathcal{T}$ that is not worked off reduces $\mu(\mathcal{T})$ by at least one. (inspect the rules)
  5. At some point the tableau only contains worked off formulae and literals.
  6. Since there are only finitely many literals in $\mathcal{T}$, so we can only apply $\mathcal{T}_0\bot$ a finite number of times.

☐

SOME RIGHTS RESERVED

Termination for Tableaux

- **Lemma 2.24.** *$\mathcal{T}_0$ terminates, i.e. every $\mathcal{T}_0$ tableau becomes saturated after finitely many rule applications.*
- *Proof:* By examining the rules wrt. a measure μ
  1. Let us call a labeled formulae A^α worked off in a tableau $\mathcal{T}$, if a $\mathcal{T}_0$ rule has already been applied to it.
  2. It is easy to see that applying rules to worked off formulae will only add formulae that are already present in its branch.
  3. Let $\mu(\mathcal{T})$ be the number of connectives in labeled formulae in $\mathcal{T}$ that are not worked off.
  4. Then each rule application to a labeled formula in $\mathcal{T}$ that is not worked off reduces $\mu(\mathcal{T})$ by at least one. (inspect the rules)
  5. At some point the tableau only contains worked off formulae and literals.
  6. Since there are only finitely many literals in $\mathcal{T}$, so we can only apply $\mathcal{T}_0\bot$ a finite number of times.

☐

- **Corollary 2.25.** *$\mathcal{T}_0$ induces a decision procedure for validity in PL^0.*
- *Proof:* We combine the results so far
  1. By 2.20 it is decidable whether $\vdash_{\mathcal{T}_0} A$
  2. By soundness (???) and completeness (???), $\vdash_{\mathcal{T}_0} A$ iff A is valid.

☐

SOME RIGHTS RESERVED

12.3 Resolution for Propositional Logic

SOME RIGHTS RESERVED

12.3.1 Resolution for Propositional Logic

SOME RIGHTS RESERVED

Another Test Calculus: Resolution

- **Definition 3.1.** A clause is a disjunction $l_1^{\alpha_1} \vee \ldots \vee l_n^{\alpha_n}$ of literals. We will use $\Box$ for the "empty" disjunction (no disjuncts) and call it the empty clause. A clause with exactly one literal is called a unit clause.

  Definition 3.2. We will often write a clause set $\{C_1, \ldots, C_n\}$ as $C_1 ; \ldots ; C_n$, use $S ; T$ for the union of the clause sets S and T, and $S ; C$ for the extension by a clause C.

- **Definition 3.3 (Resolution Calculus).** The propositional resolution calculus $\mathcal{R}_0$ operates on clause sets via a single inference rule:

  $$\frac{P^{\mathsf{T}} \vee \mathrm{A} \quad P^{\mathsf{F}} \vee \mathrm{B}}{\mathrm{A} \vee \mathrm{B}} \mathcal{R}$$

  This rule allows to add the resolvent (the clause below the line) to a clause set which contains the two clauses above. The literals P^{T} and P^{F} are called cut literals.

- **Definition 3.4 (Resolution Refutation).** Let S be a clause set, then we call an $\mathcal{R}_0$-derivation of $\Box$ from S $\mathcal{R}_0$-refutation and write $\mathcal{D} \colon S \vdash_{\mathcal{R}_0} \Box$.

SOME RIGHTS RESERVED

Clause Normal Form Transformation (A calculus)

- **Definition 3.5.** We will often write a clause set $\{C_1, \ldots, C_n\}$ as $C_1 ; \ldots ; C_n$, use $S ; T$ for the union of the clause sets S and T, and $S ; C$ for the extension by a clause C.
- **Definition 3.6 (Transformation into Clause Normal Form).** The propositional CNF calculus CNF_0 consists of the following four inference rules on sets of labeled formulae.

$$\frac{\mathrm{C} \vee (\mathrm{A} \vee \mathrm{B})^{\mathsf{T}}}{\mathrm{C} \vee \mathrm{A}^{\mathsf{T}} \vee \mathrm{B}^{\mathsf{T}}}\ CNF{\vee}^{T} \qquad \frac{\mathrm{C} \vee (\mathrm{A} \vee \mathrm{B})^{\mathsf{F}}}{\mathrm{C} \vee \mathrm{A}^{\mathsf{F}} ; \mathrm{C} \vee \mathrm{B}^{\mathsf{F}}}\ CNF{\vee}^{F}$$

$$\frac{\mathrm{C} \vee \neg \mathrm{A}^{\mathsf{T}}}{\mathrm{C} \vee \mathrm{A}^{\mathsf{F}}}\ CNF{\neg}^{T} \qquad \frac{\mathrm{C} \vee \neg \mathrm{A}^{\mathsf{F}}}{\mathrm{C} \vee \mathrm{A}^{\mathsf{T}}}\ CNF{\neg}^{F}$$

- **Definition 3.7.** We write $CNF_0(\mathrm{A}^\alpha)$ for the set of all clauses derivable from A^α via the rules above.

SOME RIGHTS RESERVED

Derived Rules of Inference

- **Definition 3.8.** An inference rule

$$\frac{A_1 \quad \ldots \quad A_n}{C}$$

is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $A_1, \ldots, A_n \vdash_{\mathcal{C}} C$.

- **Idea:** Derived rules make derivations shorter.

- **Example 3.9.**

$$\cfrac{\cfrac{\cfrac{C \vee (A \Rightarrow B)^T}{C \vee (\neg A \vee B)^T}}{C \vee \neg A^T \vee B^T}}{C \vee A^F \vee B^T} \quad \leadsto \quad \frac{C \vee (A \Rightarrow B)^T}{C \vee A^F \vee B^T}$$

- **Other Derived CNF Rules:**

$$\frac{C \vee (A \Rightarrow B)^T}{C \vee A^F \vee B^T} \quad \frac{C \vee (A \Rightarrow B)^F}{C \vee A^T \,;\, C \vee B^F} \qquad \frac{C \vee (A \wedge B)^T}{C \vee A^T \,;\, C \vee B^T} \quad \frac{C \vee (A \wedge B)^F}{C \vee A^F \vee B^F}$$

SOME RIGHTS RESERVED

Example: Proving Axiom S with Resolution

- **Example 3.10.** Clause Normal Form transformation

$$\begin{array}{c} ((P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R)^{\mathsf{F}} \\ \hline (P \Rightarrow Q \Rightarrow R)^{\mathsf{T}} ; ((P \Rightarrow Q) \Rightarrow P \Rightarrow R)^{\mathsf{F}} \\ \hline P^{\mathsf{F}} \vee (Q \Rightarrow R)^{\mathsf{T}} ; (P \Rightarrow Q)^{\mathsf{T}} ; (P \Rightarrow R)^{\mathsf{F}} \\ \hline P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} ; P^{\mathsf{T}} ; R^{\mathsf{F}} \end{array}$$

Result $\{P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}}, P^{\mathsf{F}} \vee Q^{\mathsf{T}}, P^{\mathsf{T}}, R^{\mathsf{F}}\}$

- **Example 3.11.** Resolution Proof

1	$P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}}$	initial
2	$P^{\mathsf{F}} \vee Q^{\mathsf{T}}$	initial
3	P^{T}	initial
4	R^{F}	initial
5	$P^{\mathsf{F}} \vee Q^{\mathsf{F}}$	resolve 1.3 with 4.1
6	Q^{F}	resolve 5.1 with 3.1
7	P^{F}	resolve 2.2 with 6.1
8	$\square$	resolve 7.1 with 3.1

Clause Set Simplification

- **Observation:** Let Δ be a clause set, l a literal with $l \in \Delta$ (unit clause), and Δ' be Δ where
  - all clauses $l \vee C$ have been removed and
  - and all clauses $\bar{l} \vee C$ have been shortened to C.

  Then Δ is satisfiable, iff Δ' is. We call Δ' the clause set simplification of Δ wrt. l.
- **Corollary 3.12.** *Adding clause set simplification wrt. unit clauses to $\mathcal{R}_0$ does not affect soundness and completeness.*
- This is almost always a good idea! (clause set simplification is cheap)

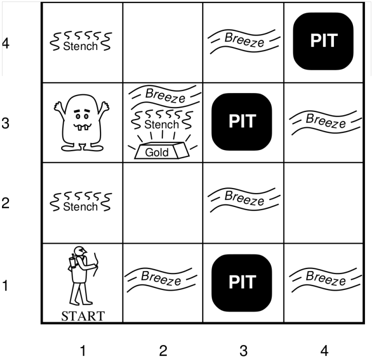
SOME RIGHTS RESERVED

12.3.2 Killing a Wumpus with Propositional Inference

SOME RIGHTS RESERVED

Applying Propositional Inference: Where is the Wumpus?

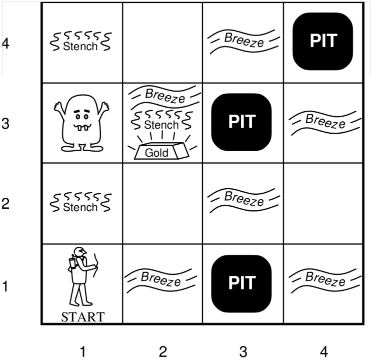
- **Example 3.13 (Finding the Wumpus).** The situation

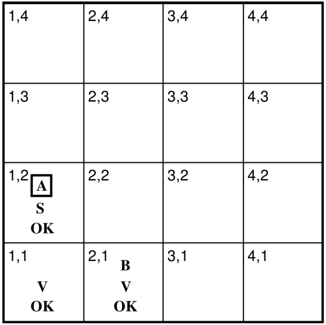


SOME RIGHTS RESERVED

Applying Propositional Inference: Where is the Wumpus?

- **Example 3.14 (Finding the Wumpus).** The situation and what the agent knows

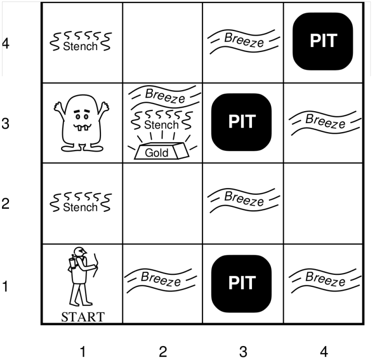




1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 **A** **S** **OK**	2,2	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1	4,1

Applying Propositional Inference: Where is the Wumpus?

- **Example 3.15 (Finding the Wumpus).** The situation and what the agent knows

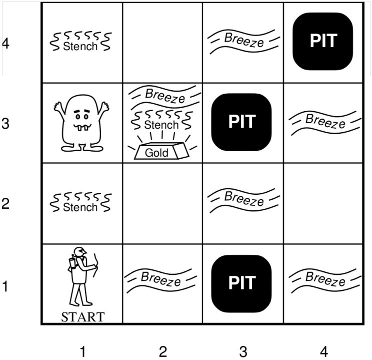


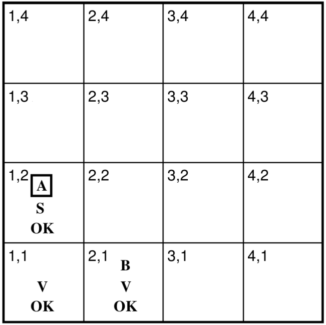
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 **A** **S** **OK**	2,2	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1	4,1

- What should the agent do next and why?

Applying Propositional Inference: Where is the Wumpus?

- **Example 3.16 (Finding the Wumpus).** The situation and what the agent knows



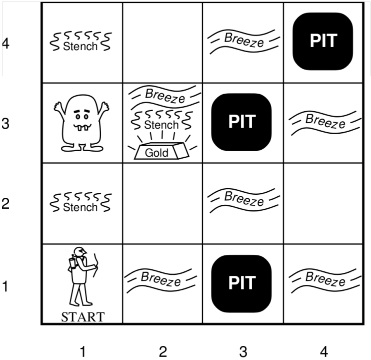


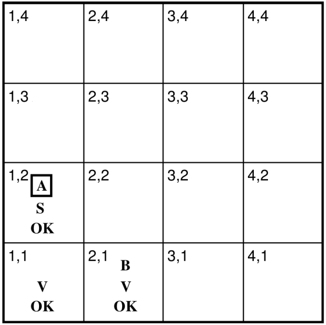
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 $\boxed{\text{A}}$ S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

  - What should the agent do next and why?
  - **One possibility**: Convince yourself that the Wumpus is in $[1, 3]$ and shoot it.

Applying Propositional Inference: Where is the Wumpus?

- **Example 3.17 (Finding the Wumpus).** The situation and what the agent knows





1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

  - What should the agent do next and why?
  - **One possibility**: Convince yourself that the Wumpus is in $[1,3]$ and shoot it.
  - What is the general mechanism here? (for the agent function)

SOME RIGHTS RESERVED

Where is the Wumpus? Our Knowledge

- **Idea:** We formalize the knowledge about the Wumpus world in PL^0 and use a test calculus to check for entailment.
- **Simplification:** We worry only about the Wumpus and stench: $S_{i,j} \mathrel{\widehat{=}}$ "*stench in* $[i,j]$", $W_{i,j} \mathrel{\widehat{=}}$ Wumpus in $[i,j]$.
- **Propositions whose value we know:** $\neg S_{1,1}$, $\neg W_{1,1}$, $\neg S_{2,1}$, $\neg W_{2,1}$, $S_{1,2}$, $\neg W_{1,2}$.
- **Knowledge about the Wumpus and smell:**
  From "*Cell adjacent to Wumpus: Stench (else: None)*", we get
  $R_1 := \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$
  $R_2 := \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}$
  $R_3 := \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}$
  $R_4 := S_{1,2} \Rightarrow (W_{1,3} \vee W_{2,2} \vee W_{1,1})$
  $\vdots$
- **To show:**
  $R_1, R_2, R_3, R_4 \models W_{1,3}$ (we will use resolution)

SOME RIGHTS RESERVED

And Now Using Resolution Conventions

- We obtain the clause set Δ composed of the following clauses:
  - **Propositions whose value we know**: $S_{1,1}^{F}$, $W_{1,1}^{F}$, $S_{2,1}^{F}$, $W_{2,1}^{F}$, $S_{1,2}^{T}$, $W_{1,2}^{F}$
  - **Knowledge about the Wumpus and smell**:

from	clauses
R_1	$S_{1,1}^{T} \vee W_{1,1}^{F}$, $S_{1,1}^{T} \vee W_{1,2}^{F}$, $S_{1,1}^{T} \vee W_{2,1}^{F}$
R_2	$S_{2,1}^{T} \vee W_{1,1}^{F}$, $S_{2,1}^{T} \vee W_{2,1}^{F}$, $S_{2,1}^{T} \vee W_{2,2}^{F}$, $S_{2,1}^{T} \vee W_{3,1}^{F}$
R_3	$S_{1,2}^{T} \vee W_{1,1}^{F}$, $S_{1,2}^{T} \vee W_{1,2}^{F}$, $S_{1,2}^{T} \vee W_{2,2}^{F}$, $S_{1,2}^{T} \vee W_{1,3}^{F}$
R_4	$S_{1,2}^{F} \vee W_{1,3}^{T} \vee W_{2,2}^{T} \vee W_{1,1}^{T}$

  - **Negated goal formula**: $W_{1,3}^{F}$

SOME RIGHTS RESERVED

Resolution Proof Killing the Wumpus!

- **Example 3.18 (Where is the Wumpus).** We show a derivation that proves that he is in $(1,3)$.
  - "*Assume the Wumpus is not in $(1,3)$. Then either there's no stench in $(1,2)$, or the Wumpus is in some other neigbor cell of $(1,2)$.*"
    - Parents: ${W_{1,3}}^{\mathsf{F}}$ and ${S_{1,2}}^{\mathsf{F}} \vee {W_{1,3}}^{\mathsf{T}} \vee {W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.
    - Resolvent: ${S_{1,2}}^{\mathsf{F}} \vee {W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.
  - "*There's a stench in $(1,2)$, so it must be another neighbor.*"
    - Parents: ${S_{1,2}}^{\mathsf{T}}$ and ${S_{1,2}}^{\mathsf{F}} \vee {W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.
    - Resolvent: ${W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.
  - "*We've been to $(1,1)$, and there's no Wumpus there, so it can't be $(1,1)$.*"
    - Parents: ${W_{1,1}}^{\mathsf{F}}$ and ${W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.
    - Resolvent: ${W_{2,2}}^{\mathsf{T}}$.
  - "*There is no stench in $(2,1)$ so it can't be $(2,2)$ either, in contradiction.*"
    - Parents: ${S_{2,1}}^{\mathsf{F}}$ and ${S_{2,1}}^{\mathsf{T}} \vee {W_{2,2}}^{\mathsf{F}}$.
    - Resolvent: ${W_{2,2}}^{\mathsf{F}}$.
    - Parents: ${W_{2,2}}^{\mathsf{F}}$ and ${W_{2,2}}^{\mathsf{T}}$.
    - Resolvent: $\square$.

  As resolution is sound, we have shown that indeed $R_1, R_2, R_3, R_4 \models W_{1,3}$.

SOME RIGHTS RESERVED

Where does the Conjecture ${W_{1,3}}^{\mathsf{F}}$ come from?

- **Question:** Where did the ${W_{1,3}}^{\mathsf{F}}$ come from?
- **Observation 3.19.** *We need a general mechanism for making conjectures.*
- **Idea:** Interpret the Wumpus world as a search problem $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ where
  - the states $\mathcal{S}$ are given by the cells (and agent orientation) and
  - the actions $\mathcal{A}$ by the possible actions of the agent.

  Use tree search as the main agent program and a test calculus for testing all dangers (pits), opportunities (gold) and the Wumpus.
- **Example 3.20 (Back to the Wumpus).** In 3.13, the agent is in $[1,2]$, it has perceived stench, and the possible actions include shoot, and $\mathrm{goForward}$. Evaluating either of these leads to the conjecture $W_{1,3}$. And since $W_{1,3}$ is entailed, the action shoot probably comes out best, heuristically.
- **Remark:** Analogous to the backtracking with inference algorithm from CSP.

12.4 Conclusion

SOME RIGHTS RESERVED

Summary

- Every propositional formula can be brought into conjunctive normal form (CNF), which can be identified with a set of clauses.
- The tableau and resolution calculi are deduction procedures based on trying to derive a contradiction from the negated theorem (a closed tableau or the empty clause). They are refutation complete, and can be used to prove $\mathrm{KB} \models \mathsf{A}$ by showing that $\mathrm{KB} \cup \{\neg\mathsf{A}\}$ is unsatisfiable.

SOME RIGHTS RESERVED

Chapter 13
Propositional Reasoning: SAT Solvers

SOME RIGHTS RESERVED

13.1 Introduction

Reminder: Our Agenda for Propositional Logic

- : Basic definitions and concepts; machine-oriented calculi
  - Sets up the framework. Tableaux and resolution are the quintessential reasoning procedures underlying most successful SAT solvers.
- **This chapter**: The Davis Putnam procedure and clause learning.
  - State-of-the-art algorithms for reasoning about propositional logic, and an important observation about how they behave.

SOME RIGHTS RESERVED

SAT: The Propositional Satisfiability Problem

- **Definition 1.1.** The SAT problem (SAT): Given a propositional formula A, decide whether or not A is satisfiable. We denote the class of all SAT problems with SAT
- The SAT problem was the first problem proved to be NP-complete!
- A is commonly assumed to be in CNF. This is without loss of generality, because any A can be transformed into a satisfiability-equivalent CNF formula (cf.) in polynomial time.
- Active research area, annual SAT conference, lots of tools etc. available: `http://www.satlive.org/`
- **Definition 1.2.** Tools addressing SAT are commonly referred to as SAT solvers.
- **Recall:** To decide whether $\mathrm{KB} \vDash \mathsf{A}$, decide satisfiability of $\theta := \mathrm{KB} \cup \{\neg \mathsf{A}\}$: θ is unsatisfiable iff $\mathrm{KB} \vDash \mathsf{A}$.
- **Consequence:** Deduction can be performed using SAT solvers.

SOME RIGHTS RESERVED

SAT vs. CSP

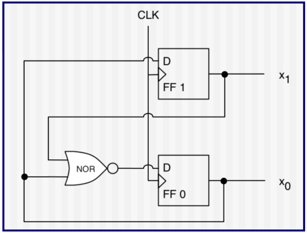
- **Recall:** Constraint network $\langle V, D, C, C, C, V, E\rangle$ has variables $v \in V$ with finite domains $D_v \in D$, and binary constraints $C_{uv} \in C$ which are relations over u, and v specifying the permissible combined assignments to u and v. One extension is to allow constraints of higher arity.
- **Observation 1.3 (SAT: A kind of CSP).** *SAT can be viewed as a CSP problem in which all variable domains are Boolean, and the constraints have unbounded order.*
- **Theorem 1.4 (Encoding CSP as SAT).** *Given any constraint network $\mathcal{C}$, we can in low order polynomial time construct a CNF formula* $A(\mathcal{C})$ *that is satisfiable iff $\mathcal{C}$ is solvable.*
- *Proof:* We design a formula, relying on known transformation to CNF
  1. encode multi-XOR for each variable
  2. encode each constraint by DNF over relation
  3. Running time: $\mathcal{O}(nd^2 + md^2)$ where n is the number of variables, d the domain size, and m the number of constraints.

  ☐
- **Upshot:** Anything we can do with CSP, we can (in principle) do with SAT.

SOME RIGHTS RESERVED

Example Application: Hardware Verification

- **Example 1.5 (Hardware Verification).**



- Counter, repeatedly from $c = 0$ to $c = 2$.
- 2 bits x_1 and x_0; $c = 2 * x_1 + x_0$.
- (FF$\widehat{=}$ Flip-Flop, D $\widehat{=}$ Data IN, CLK $\widehat{=}$ Clock)
- **To Verify**: If $c < 3$ in current clock cycle, then $c < 3$ in next clock cycle.

- **Step 1:** Encode into propositional logic.
  - **Propositions**: x_1, x_0; and y_1, y_0 (value in next cycle).
  - **Transition relation**: $y_1 \Leftrightarrow y_0$; $y_0 \Leftrightarrow \neg(x_1 \vee x_0)$.
  - **Initial state**: $\neg(x_1 \wedge x_0)$.
  - **Error property**: $x_1 \wedge y_0$.
- **Step 2:** Transform to CNF, encode as a clause set Δ.
  - **Clauses**: ${y_1}^F \vee {x_0}^T$, ${y_1}^T \vee {x_0}^F$, ${y_0}^T \vee {x_1}^T \vee {x_0}^T$, ${y_0}^F \vee {x_1}^F$, ${y_0}^F \vee {x_0}^F$, ${x_1}^F \vee {x_0}^F$, ${y_1}^T$, ${y_0}^T$.
- **Step 3:** Call a SAT solver (up next).

SOME RIGHTS RESERVED

Our Agenda for This Chapter

- **The Davis-Putnam (Logemann-Loveland) Procedure:** How to systematically test satisfiability?
  - The quintessential SAT solving procedure, DPLL.
- **DPLL is (A Restricted Form of) Resolution:** How does this relate to what we did in the last chapter?
  - mathematical understanding of DPLL.
- **Why Did Unit Propagation Yield a Conflict?:** How can we analyze which mistakes were made in "dead" search branches?
  - Knowledge is power, see next.
- **Clause Learning:** How can we learn from our mistakes?
  - One of the key concepts, perhaps *the* key concept, underlying the success of SAT.
- **Phase Transitions – Where the Really Hard Problems Are:** Are *all* formulas "hard" to solve?
  - The answer is "no". And in some cases we can figure out exactly when they are/aren't hard to solve.

SOME RIGHTS RESERVED

13.2 The Davis-Putnam (Logemann-Loveland) Procedure

SOME RIGHTS RESERVED

The DPLL Procedure

- **Definition 2.1.** The Davis Putnam procedure (DPLL) is a SAT solver called on a clause set Δ and the empty assignment ϵ. It interleaves unit propagation (UP) and splitting:

```
function DPLL($\Delta$,$I$) returns a partial assignment $I$, or ‘‘unsatisfiable’’
  /* Unit Propagation (UP) Rule: */
  $\Delta'$ := a copy of $\Delta$; $I'$ := $I$
  while $\Delta'$ contains a unit clause $C = P^\alpha$ do
    extend $I'$ with $[\alpha/P]$, clause−set simplify $\Delta'$
   /* Termination Test: */
   if $\square \in \Delta'$ then return ‘‘unsatisfiable’’
   if $\Delta' = \{\}$ then return $I'$
   /* Splitting Rule: */
  select some proposition $P$ for which $I'$ is not defined
  $I''$ := $I'$ extended with one truth value for $P$; $\Delta''$ := a copy of $\Delta'$; simplify $\Delta''$
  if $I'''$ := DPLL($\Delta''$,$I''$) $\neq$ ‘‘unsatisfiable’’ then return $I'''$
  $I''$ := $I'$ extended with the other truth value for $P$; $\Delta''$ := $\Delta'$; simplify $\Delta''$
  return DPLL($\Delta''$,$I''$)
```

- In practice, of course one uses flags etc. instead of “copy”.

SOME RIGHTS RESERVED

DPLL: Example (Vanilla1)

- **Example 2.2 (UP and Splitting).** Let
  $\Delta := P^{\mathsf{T}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}} \,;\, P^{\mathsf{F}} \vee Q^{\mathsf{F}} \,;\, R^{\mathsf{T}} \,;\, P^{\mathsf{T}} \vee Q^{\mathsf{F}}$
  1. UP Rule: $R \mapsto \mathsf{T}$
     $P^{\mathsf{T}} \vee Q^{\mathsf{T}} \,;\, P^{\mathsf{F}} \vee Q^{\mathsf{F}} \,;\, P^{\mathsf{T}} \vee Q^{\mathsf{F}}$
  2. Splitting Rule:

2a. $P \mapsto \mathsf{F}$
$Q^{\mathsf{T}} \,;\, Q^{\mathsf{F}}$

3a. UP Rule: $Q \mapsto \mathsf{T}$
$\Box$
returning “unsatisfiable”

2b. $P \mapsto \mathsf{T}$
Q^{F}

3b. UP Rule: $Q \mapsto \mathsf{F}$
clause set empty
returning “$R \mapsto \mathsf{T}, P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}$

SOME RIGHTS RESERVED

DPLL: Example (Vanilla2)

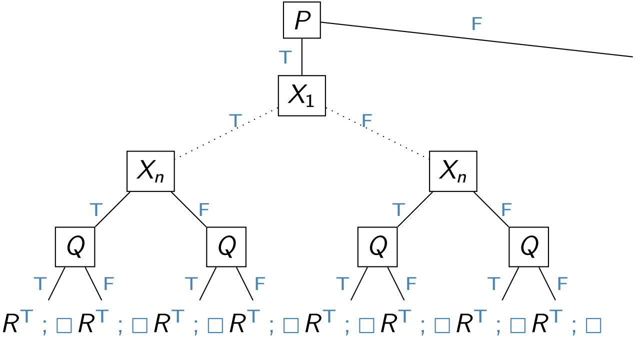
- **Observation:** Sometimes UP is all we need.
- **Example 2.3.** Let $\Delta := Q^{F} \vee P^{F} \mathbin{;} P^{T} \vee Q^{F} \vee R^{F} \vee S^{F} \mathbin{;} Q^{T} \vee S^{F} \mathbin{;} R^{T} \vee S^{F} \mathbin{;} S^{T}$

  1. UP Rule: $S \mapsto \mathsf{T}$
     $Q^{F} \vee P^{F} \mathbin{;} P^{T} \vee Q^{F} \vee R^{F} \mathbin{;} Q^{T} \mathbin{;} R^{T}$
  2. UP Rule: $Q \mapsto \mathsf{T}$
     $P^{F} \mathbin{;} P^{T} \vee R^{F} \mathbin{;} R^{T}$
  3. UP Rule: $R \mapsto \mathsf{T}$
     $P^{F} \mathbin{;} P^{T}$
  4. UP Rule: $P \mapsto \mathsf{T}$
     $\Box$

SOME RIGHTS RESERVED

DPLL: Example (Redundance1)

- **Example 2.4.** We introduce some nasty redundance to make DPLL slow.
  $\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
  DPLL on $\Delta ; \Theta$ with $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}}$



Properties of DPLL

- **Unsatisfiable case:** What can we say if "unsatisfiable" is returned?
  - In this case, we know that Δ is unsatisfiable: Unit propagation is *sound*, in the sense that it does not reduce the set of solutions.

Properties of DPLL

- **Unsatisfiable case:** What can we say if “unsatisfiable” is returned?
  - In this case, we know that Δ is unsatisfiable: Unit propagation is *sound*, in the sense that it does not reduce the set of solutions.
- **Satisfiable case:** What can we say when a partial interpretation I is returned?
  - Any extension of I to a complete interpretation satisfies Δ. (By construction, I suffices to satisfy all clauses.)

SOME RIGHTS RESERVED

Properties of DPLL

- **Unsatisfiable case:** What can we say if “unsatisfiable” is returned?
  - In this case, we know that Δ is unsatisfiable: Unit propagation is *sound*, in the sense that it does not reduce the set of solutions.
- **Satisfiable case:** What can we say when a partial interpretation I is returned?
  - Any extension of I to a complete interpretation satisfies Δ. (By construction, I suffices to satisfy all clauses.)
- Déjà Vu, Anybody?

SOME RIGHTS RESERVED

Properties of DPLL

- **Unsatisfiable case:** What can we say if "unsatisfiable" is returned?
  - In this case, we know that Δ is unsatisfiable: Unit propagation is *sound*, in the sense that it does not reduce the set of solutions.
- **Satisfiable case:** What can we say when a partial interpretation I is returned?
  - Any extension of I to a complete interpretation satisfies Δ. (By construction, I suffices to satisfy all clauses.)
- Déjà Vu, Anybody?
- DPLL $\widehat{=}$ backtracking with inference, where inference $\widehat{=}$ unit propagation.
  - Unit propagation is sound: It does not reduce the set of solutions.
  - Running time is exponential in worst case, good variable/value selection strategies required.

SOME RIGHTS RESERVED

13.3 DPLL $\widehat{=}$ (A Restricted Form of) Resolution

SOME RIGHTS RESERVED

UP $\widehat{=}$ Unit Resolution

- **Observation:** The unit propagation (UP) rule corresponds to a calculus:

```
while $\Delta'$ contains a unit clause $\{l\}$ do
  extend $I'$ with the respective truth value for the proposition underlying $l$
  simplify $\Delta'$ /* remove false literals */
```

- **Definition 3.1 (Unit Resolution).** Unit resolution (UR) is the test calculus consisting of the following inference rule:

$$\frac{C \vee P^{\alpha} \quad P^{\beta} \quad \alpha \neq \beta}{C}\,\mathrm{UR}$$

- Unit propagation $\widehat{=}$ resolution restricted to cases where one parent is unit clause.
- **Observation 3.2 (Soundness).** UR *is refutation sound.* *(since resolution is)*
- **Observation 3.3 (Completeness).** UR *is not refutation complete (alone).*
- **Example 3.4.** $P^{\mathsf{T}} \vee Q^{\mathsf{T}}\,;\, P^{\mathsf{T}} \vee Q^{\mathsf{F}}\,;\, P^{\mathsf{F}} \vee Q^{\mathsf{T}}\,;\, P^{\mathsf{F}} \vee Q^{\mathsf{F}}$ is unsatisfiable but UR cannot derive the empty clause $\square$.
- UR makes only limited inferences, as long as there are unit clauses. It does not guarantee to infer everything that can be inferred.

SOME RIGHTS RESERVED

DPLL vs. Resolution

- **Definition 3.5.** We define the number of decisions of a DPLL run as the total number of times a truth value was set by either unit propagation or splitting.
- **Theorem 3.6.** *If DPLL returns "unsatisfiable" on Δ, then $\Delta \vdash_{\mathcal{R}_0} \Box$ with a resolution proof whose length is at most the number of decisions.*
- *Proof:* Consider first DPLL without UP
  1. Consider any leaf node N, for proposition X, both of whose truth values directly result in a clause C that has become empty.
  2. Then for $X = \mathsf{F}$ the respective clause C must contain X^{T}; and for $X = \mathsf{T}$ the respective clause C must contain X^{F}. Thus we can resolve these two clauses to a clause $C(N)$ that does not contain X.
  3. $C(N)$ can contain only the negations of the decision literals $l_1, \ldots, l_k$ above N. Remove N from the tree, then iterate the argument. Once the tree is empty, we have derived the empty clause.
  4. Unit propagation can be simulated via applications of the splitting rule, choosing a proposition that is constrained by a unit clause: One of the two truth values then immediately yields an empty clause.

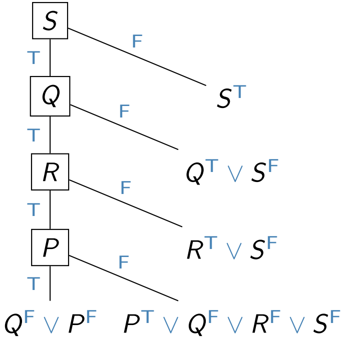
$\Box$

SOME RIGHTS RESERVED

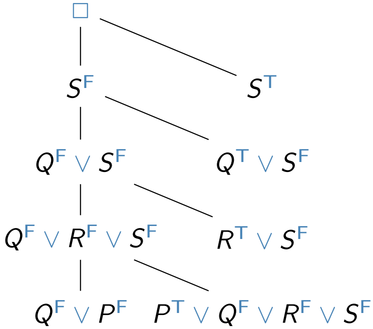
DPLL vs. Resolution: Example (Vanilla2)

- **Observation:** The proof of 3.6 is constructive, so we can use it as a method to read of a resolution proof from a DPLL trace.
- **Example 3.7.** We follow the steps in the proof of 3.6 for $\Delta := Q^{F} \vee P^{F} ; P^{T} \vee Q^{F} \vee R^{F} \vee S^{F} ; Q^{T} \vee S^{F} ; R^{T} \vee S^{F} ; S^{T}$

DPLL: (Without UP; leaves annotated with clauses that became empty)



Resolution proof from that DPLL tree:



- **Intuition:** From a (top-down) DPLL tree, we generate a (bottom-up) resolution proof.

SOME RIGHTS RESERVED

DPLL vs. Resolution: Discussion

- **So What?:** The theorem we just proved helps to *understand* DPLL: DPLL is an efficient practical method for conducting resolution proofs.
- **In fact:** DPLL $\widehat{=}$ tree resolution.
- **Definition 3.8.** In a tree resolution, each derived clause C is used only once (at its parent).
- **Problem:** The same C must be derived anew every time it is used!
- **This is a fundamental weakness:** There are inputs Δ whose shortest tree resolution proof is exponentially longer than their shortest (general) resolution proof.
- **Intuitively:** DPLL makes the same mistakes over and over again.
- **Idea:** DPLL should learn from its mistakes on one search branch, and apply the learned knowledge to other branches.
- **To the rescue:** clause learning (up next)

SOME RIGHTS RESERVED

13.4 Conclusion

SOME RIGHTS RESERVED

Summary

- SAT solvers decide satisfiability of CNF formulas. This can be used for deduction, and is highly successful as a general problem solving technique (e.g., in verification).
- DPLL $\widehat{=}$ backtracking with inference performed by unit propagation (UP), which iteratively instantiates unit clauses and simplifies the formula.
- DPLL proofs of unsatisfiability correspond to a restricted form of resolution. The restriction forces DPLL to "makes the same mistakes over again".
- Implication graphs capture how UP derives conflicts. Their analysis enables us to do clause learning. DPLL with clause learning is called CDCL. It corresponds to full resolution, not "making the same mistakes over again".
- CDCL is state of the art in applications, routinely solving formulas with millions of propositions.
- In particular random formula distributions, typical problem hardness is characterized by phase transitions.

SOME RIGHTS RESERVED

State of the Art in SAT

- **SAT competitions:**
  - Since beginning of the 90s `http://www.satcompetition.org/`
  - *random* vs. *industrial* vs. *handcrafted* benchmarks.
  - Largest industrial instances: $>$ 1.000.000 propositions.
- **State of the art is CDCL:**
  - Vastly superior on handcrafted and industrial benchmarks.
  - Key techniques: clause learning! Also: Efficient implementation (UP!), good branching heuristics, random restarts, portfolios.
- **What about local search?:**
  - Better on random instances.
  - No "dramatic" progress in last decade.
  - Parameters are difficult to adjust.

SOME RIGHTS RESERVED

But – What About Local Search for SAT?

- **There's a wealth of research on local search for SAT, e.g.:**
- **Definition 4.1.** The GSAT algorithm **OUTPUT**: a satisfying truth assignment of Δ, if found

```
function GSAT ($\Delta$, $MaxFlips$ $MaxTries$
  for $i$ :=1 to $MaxTries$
    $I$ := a randomly−generated truth assignment
    for $j$ :=1 to $MaxFlips$
    if $I$ satisfies $\Delta$ then return $I$
          $X$:= a proposition reversing whose truth assignment gives
          the largest increase in the number of satisfied clauses
        $I$ := $I$ with the truth assignment of $X$ reversed
    end for
  end for
  return ‘‘no satisfying assignment found’’
```

- local search is not as successful in SAT applications, and the underlying ideas are very similar to those presented in (Not covered here)

Topics We Didn't Cover Here

- **Variable/value selection heuristics**: A whole zoo is out there.
- **Implementation techniques**: One of the most intensely researched subjects. Famous "watched literals" technique for UP had huge practical impact.
- **Local search**: In space of all truth value assignments. GSAT (slide 407) had huge impact at the time (1992), caused huge amount of follow-up work. Less intensely researched since clause learning hit the scene in the late 90s.
- **Portfolios**: How to combine several SAT solvers efficiently?
- **Random restarts**: Tackling heavy-tailed runtime distributions.
- **Tractable SAT**: Polynomial-time sub-classes (most prominent: 2-SAT, Horn formulas).
- **MaxSAT**: Assign weight to each clause, maximize weight of satisfied clauses (= optimization version of SAT).
- **Resolution special cases**: There's a universe in between unit resolution and full resolution: trade off inference vs. search.
- **Proof complexity**: Can one resolution special case X simulate another one Y polynomially? Or is there an exponential separation (example families where X is exponentially less efficient than Y)?

SOME RIGHTS RESERVED

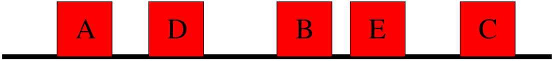
Chapter 14
First-Order Predicate Logic

14.1 Motivation: A more Expressive Language

SOME RIGHTS RESERVED

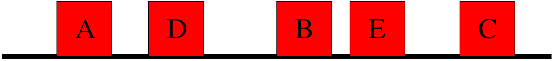
Let's Talk About Blocks, Baby . . .

- **Question:** What do you see here?



Let's Talk About Blocks, Baby ...

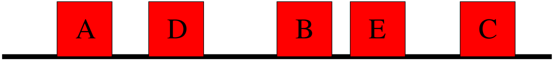
- **Question:** What do you see here?



- **You say:** "All blocks are red"; "All blocks are on the table"; "A is a block".
- **And now:** Say it in propositional logic!

Let's Talk About Blocks, Baby . . .

- **Question:** What do you see here?

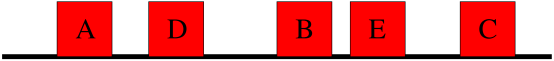


- **You say:** “All blocks are red”; “All blocks are on the table”; “A is a block”.
- **And now:** Say it in propositional logic!
- **Answer:** “isRedA”,“isRedB”, . . . , “onTableA”, “onTableB”, . . . , “isBlockA”, . . .
- **Wait a sec!:** Why don't we just say, e.g., “AllBlocksAreRed” and “isBlockA”?
- **Problem:** Could we conclude that A is red? (No)
  These statements are atomic (just strings); their inner structure (“all blocks”, “is a block”) is not captured.

SOME RIGHTS RESERVED

Let's Talk About Blocks, Baby …

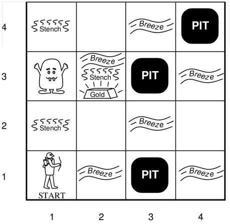
- **Question:** What do you see here?



- **You say:** "All blocks are red"; "All blocks are on the table"; "A is a block".
- **And now:** Say it in propositional logic!
- **Answer:** "isRedA","isRedB", …, "onTableA", "onTableB", …, "isBlockA", …
- **Wait a sec!:** Why don't we just say, e.g., "AllBlocksAreRed" and "isBlockA"?
- **Problem:** Could we conclude that A is red? (No)
  These statements are atomic (just strings); their inner structure ("all blocks", "is a block") is not captured.
- **Idea:** Predicate Logic (PL^1) extends propositional logic with the ability to explicitly speak about objects and their properties.
- **How?:** Variables ranging over objects, predicates describing object properties, …
- **Example 1.4.** "$\forall x.\mathrm{block}(x) \Rightarrow \mathrm{red}(x)$"; "$\mathrm{block}(A)$"

SOME RIGHTS RESERVED

Let's Talk About the Wumpus Instead?



Percepts: [*Stench*, *Breeze*, *Glitter*, *Bump*, *Scream*]

- - Cell adjacent to Wumpus: *Stench* (else: *None*).
  - Cell adjacent to Pit: *Breeze* (else: *None*).
  - Cell that contains gold: *Glitter* (else: *None*).
  - You walk into a wall: *Bump* (else: *None*).
  - Wumpus shot by arrow: *Scream* (else: *None*).
- Say, in propositional logic: "Cell adjacent to Wumpus: *Stench*."
  - $W_{1,1} \Rightarrow S_{1,2} \wedge S_{2,1}$
  - $W_{1,2} \Rightarrow S_{2,2} \wedge S_{1,1} \wedge S_{1,3}$
  - $W_{1,3} \Rightarrow S_{2,3} \wedge S_{1,2} \wedge S_{1,4}$
  - ...
- **Note:** Even when we *can* describe the problem suitably, for the desired reasoning, the propositional formulation typically is way too large to write (by hand).
- **PL1 solution:** "$\forall x.\mathrm{Wumpus}(x) \Rightarrow (\forall y.\mathrm{adj}(x,y) \Rightarrow \mathrm{stench}(y))$"

SOME RIGHTS RESERVED

Blocks/Wumpus, Who Cares? Let's Talk About Numbers!

- Even worse!
- **Example 1.5 (Integers).** A limited vocabulary to talk about these
  - The objects: $\{1, 2, 3, \ldots\}$.
  - Predicate 1: "$\mathrm{even}(x)$" should be true iff x is even.
  - Predicate 2: "$\mathrm{eq}(x, y)$" should be true iff $x = y$.
  - Function: $\mathrm{succ}(x)$ maps x to $x + 1$.
- **Old problem:** Say, in propositional logic, that "$1 + 1 = 2$".
  - Inner structure of vocabulary is ignored (cf. "AllBlocksAreRed").
  - PL1 solution: "$\mathrm{eq}(\mathrm{succ}(1), 2)$".
- **New Problem:** Say, in propositional logic, "if x is even, so is $x + 2$".
  - It is impossible to speak about infinite sets of objects!
  - PL1 solution: "$\forall x.\mathrm{even}(x) \Rightarrow \mathrm{even}(\mathrm{succ}(\mathrm{succ}(x)))$".

SOME RIGHTS RESERVED

Now We're Talking

- **Example 1.6.**

  $$\forall n.\mathrm{gt}(n,2) \Rightarrow \neg(\exists a, b, c.\mathrm{eq}(\mathrm{plus}(\mathrm{pow}(a,n),\mathrm{pow}(b,n)),\mathrm{pow}(c,n)))$$

  Read: "*Forall $n > 2$, there are no a, b, c, such that $a^n + b^n = c^n$*"(Fermat's last theorem)

- **Theorem proving in PL1:** Arbitrary theorems, in principle.
  - Fermat's last theorem is of course infeasible, but interesting theorems can and have been proved automatically.
  - See `http://en.wikipedia.org/wiki/Automated_theorem_proving`.
  - **Note**: Need to axiomatize "Plus", "PowerOf", "Equals". See `http://en.wikipedia.org/wiki/Peano_axioms`

SOME RIGHTS RESERVED

What Are the Practical Relevance/Applications?

- **. . . even asking this question is a sacrilege:**

What Are the Practical Relevance/Applications?

- **. . . even asking this question is a sacrilege:**
- (Quotes from Wikipedia)
  - *“In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics.”*

What Are the Practical Relevance/Applications?

- **. . . even asking this question is a sacrilege:**
- (Quotes from Wikipedia)
  - *“In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics.”*
  - *“The development of logic since Frege, Russell, and Wittgenstein had a profound influence on the practice of philosophy and the perceived nature of philosophical problems, and Philosophy of mathematics.”*

SOME RIGHTS RESERVED

What Are the Practical Relevance/Applications?

- **. . . even asking this question is a sacrilege:**
- (Quotes from Wikipedia)
  - *"In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics."*
  - *"The development of logic since Frege, Russell, and Wittgenstein had a profound influence on the practice of philosophy and the perceived nature of philosophical problems, and Philosophy of mathematics."*
  - *"During the later medieval period, major efforts were made to show that Aristotle's ideas were compatible with Christian faith."*
  - (In other words: the church issued for a long time that Aristotle's ideas were *in*compatible with Christian faith.)

SOME RIGHTS RESERVED

What Are the Practical Relevance/Applications?

- **You're asking it anyhow:**
  - **Logic programming**. Prolog et al.
  - **Databases**. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
  - Semantic technology. Mega-trend since > a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.

SOME RIGHTS RESERVED

What Are the Practical Relevance/Applications?

- **You're asking it anyhow:**
  - **Logic programming**. Prolog et al.
  - **Databases**. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
  - Semantic technology. Mega-trend since > a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.
- Prominent PL1 fragment: Web Ontology Language OWL.

SOME RIGHTS RESERVED

What Are the Practical Relevance/Applications?

- **You're asking it anyhow:**
  - **Logic programming**. Prolog et al.
  - **Databases**. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
  - Semantic technology. Mega-trend since $>$ a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.
- Prominent PL1 fragment: Web Ontology Language OWL.
- Prominent data set: The WWW. (semantic web)

SOME RIGHTS RESERVED

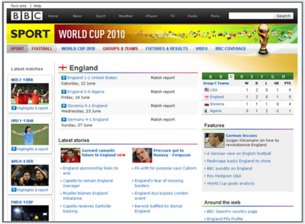
What Are the Practical Relevance/Applications?

- **You're asking it anyhow:**
  - **Logic programming**. Prolog et al.
  - **Databases**. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
  - Semantic technology. Mega-trend since > a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.
- Prominent PL1 fragment: Web Ontology Language OWL.
- Prominent data set: The WWW. (semantic web)
- **Assorted quotes on Semantic Web and OWL:**
  - "*The brain of humanity.*"
  - "*The Semantic Web will never work.*"
  - "*A TRULY meaningful way of interacting with the Web may finally be here: the Semantic Web. The idea was proposed 10 years ago. A triumvirate of internet heavyweights – Google, Twitter, and Facebook – are making it real.*"

SOME RIGHTS RESERVED

(A Few) Semantic Technology Applications

Web Queries



Text only | Help

Latest matches

NED 2-1 BRA
Highlights & report

URU 1-1 GHA
Highlights & report

ARG 0-4 GER
Highlights & report

PAR 0-1 ESP

England

England 1-1 United States — Match report
Saturday, 12 June

England 0-0 Algeria — Match report
Friday, 18 June

Slovenia 0-1 England — Match report
Wednesday, 23 June

Germany 4-1 England — Match report
Sunday, 27 June

A B C D E F G H					
Group C Teams	W	D	L	GD	PTS
USA	1	2	0	1	5
England	1	2	0	1	5
Slovenia	1	1	1	0	4
Algeria	0	1	2	-2	1

Latest stories

Gerrard commits future to England NEW

Pressure got to Rooney - Ferguson

- England sponsorship likely to end
- Capello to remain England manager
- Mueller blames England imbalance
- Capello receives Gartside backing
- FA unfit for purpose says Caborn
- England's fear of crossing borders
- England duo bypass London event
- Barwick baffled by dismal England

Features

German lessons
Jurgen Klinsmann on how to revolutionise England

- A German view on English football
- Redknapp backs England to shine
- BBC pundits on England
- Roy Hodgson Q&A
- World Cup goals analysis

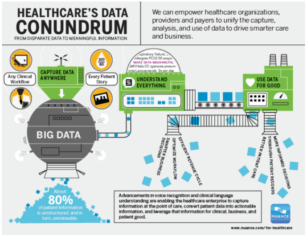
Around the web

- BBC Search+ country page
- England Fifa Profile

Jeopardy (IBM Watson)

Context-Aware Apps

Healthcare



HEALTHCARE'S DATA CONUNDRUM

FROM DISPARATE DATA TO MEANINGFUL INFORMATION

We can empower healthcare organizations, providers and payers to unify the capture, analysis, and use of data to drive smarter care and business.

About 80% of patient information is unstructured, and in turn, unmineable.

Advancements in voice recognition and clinical language understanding are enabling the healthcare enterprise to capture information at the point of care, convert patient data into actionable information, and leverage that information for clinical, business, and patient good.

www.nuance.com/for-healthcare

SOME RIGHTS RESERVED

Our Agenda for This Topic

- **This Chapter**: Basic definitions and concepts; normal forms.
  - Sets up the framework and basic operations.
  - **Syntax**: How to write PL1 formulas? (Obviously required)
  - **Semantics**: What is the meaning of PL1 formulas? (Obviously required.)
  - **Normal Forms**: What are the basic normal forms, and how to obtain them? (Needed for algorithms, which are defined on these normal forms.)
- **Next Chapter**: Compilation to propositional reasoning; unification; lifted resolution/tableau.
  - Algorithmic principles for reasoning about predicate logic.

SOME RIGHTS RESERVED

14.2 First-Order Logic

SOME RIGHTS RESERVED

First-Order Predicate Logic (PL^1)

- **Coverage:** We can talk about (“*All humans are mortal*”)
  - individual things and denote them by variables or constants
  - properties of individuals, (e.g. being human or mortal)
  - relations of individuals, (e.g. *sibling_of* relationship)
  - functions on individuals, (e.g. the *father_of* function)

  We can also state the existence of an individual with a certain property, or the universality of a property.
- But we cannot state assertions like
  - “*There is a surjective function from the natural numbers into the reals*”.
- First-Order Predicate Logic has many good properties (complete calculi, compactness, unitary, linear unification,…)
- But too weak for formalizing: (at least directly)
  - natural numbers, torsion groups, calculus, …
  - generalized quantifiers (“*most, few,…*”)

SOME RIGHTS RESERVED

14.2.1 First-Order Logic: Syntax and Semantics

PL^1 Syntax (Signature and Variables)

- **Definition 2.1.** First-order logic (PL^1), is a formal system extensively used in mathematics, philosophy, linguistics, and CS. It combines propositional logic with the ability to quantify over individuals.
- PL^1 talks about two kinds of objects: (so we have two kinds of symbols)
  - truth values by reusing PL^0
  - individuals, e.g. numbers, foxes, Pokémon,. . .
- **Definition 2.2.** A first-order signature consists of (all disjoint; $k \in \mathbb{N}$)
  - connectives: $\Sigma_0 = \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots\}$ (functions on truth values)
  - function constants: $\Sigma^f_k = \{f, g, h, \ldots\}$ (k-ary functions on individuals)
  - predicate constants: ${\Sigma^p}_k = \{p, q, r, \ldots\}$ (k-ary relations among individuals.)
  - (Skolem constants: $\Sigma^{sk}_k = \{f^1_k, f^2_k, \ldots\}$) (witness constructors; countably ∞)
  - We take Σ_1 to be all of these together: $\Sigma_1 := \Sigma^f \cup \Sigma^p \cup \Sigma^{sk}$ and define $\Sigma := \Sigma_1 \cup \Sigma_0$.
- **Definition 2.3.** We assume a set of individual variables: $\mathcal{V}_\iota := \{X, Y, Z, \ldots\}$. (countably ∞)

SOME RIGHTS RESERVED

PL^1 Syntax (Formulae)

- **Definition 2.4.** Terms: $\mathrm{A} \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ (denote individuals)
  - $\mathcal{V}_\iota \subseteq wff_\iota(\Sigma_1, \mathcal{V}_\iota)$,
  - if $f \in \Sigma^f_k$ and $\mathrm{A}^i \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $f(\mathrm{A}^1, \ldots, \mathrm{A}^k) \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$.
- **Definition 2.5.** First-order propositions: $\mathrm{A} \in wff_o(\Sigma_1, \mathcal{V}_\iota)$:(denote truth values)
  - if $p \in \Sigma^p{}_k$ and $\mathrm{A}^i \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $p(\mathrm{A}^1, \ldots, \mathrm{A}^k) \in wff_o(\Sigma_1, \mathcal{V}_\iota)$,
  - if $\mathrm{A}, \mathrm{B} \in wff_o(\Sigma_1, \mathcal{V}_\iota)$ and $X \in \mathcal{V}_\iota$, then $T, \mathrm{A} \wedge \mathrm{B}, \neg \mathrm{A}, \forall X.\mathrm{A} \in wff_o(\Sigma_1, \mathcal{V}_\iota)$.

  $\forall$ is a binding operator called the universal quantifier.
- **Definition 2.6.** We define the connectives $F, \vee, \Rightarrow, \Leftrightarrow$ via the abbreviations $\mathrm{A} \vee \mathrm{B} := \neg(\neg \mathrm{A} \wedge \neg \mathrm{B})$, $\mathrm{A} \Rightarrow \mathrm{B} := \neg \mathrm{A} \vee \mathrm{B}$, $\mathrm{A} \Leftrightarrow \mathrm{B} := (\mathrm{A} \Rightarrow \mathrm{B}) \wedge (\mathrm{B} \Rightarrow \mathrm{A})$, and $F := \neg T$. We will use them like the primary connectives $\wedge$ and $\neg$
- **Definition 2.7.** We use $\exists X.\mathrm{A}$ as an abbreviation for $\neg(\forall X.\neg \mathrm{A})$. $\exists$ is a binding operator called the existential quantifier.
- **Definition 2.8.** Call formulae without connectives or quantifiers atomic else complex.

SOME RIGHTS RESERVED

Alternative Notations for Quantifiers

Here	Elsewhere	
$\forall x.A$	$\bigwedge x.A$	$(x)A$
$\exists x.A$	$\bigvee x.A$	

Free and Bound Variables

- **Definition 2.9.** We call an occurrence of a variable X bound in a formula A (otherwise free), iff it occurs in a sub-formula $\forall X.\mathrm{B}$ of A.
  For a formula A, we will use $\mathrm{BVar}(\mathrm{A})$ (and $\mathrm{free}(\mathrm{A})$) for the set of bound (free) variables of A, i.e. variables that have a free/bound occurrence in A.
- **Definition 2.10.** We define the set $\mathrm{free}(\mathrm{A})$ of free variables of a formula A:

$$
\begin{aligned}
&\mathrm{free}(X) := \{X\}\\
&\mathrm{free}(f(\mathrm{A}_1,\ldots,\mathrm{A}_n)) := \bigcup_{1\leq i\leq n}\mathrm{free}(\mathrm{A}_i)\\
&\mathrm{free}(p(\mathrm{A}_1,\ldots,\mathrm{A}_n)) := \bigcup_{1\leq i\leq n}\mathrm{free}(\mathrm{A}_i)\\
&\mathrm{free}(\neg\mathrm{A}) := \mathrm{free}(\mathrm{A})\\
&\mathrm{free}(\mathrm{A}\wedge\mathrm{B}) := \mathrm{free}(\mathrm{A})\cup\mathrm{free}(\mathrm{B})\\
&\mathrm{free}(\forall X.\mathrm{A}) := \mathrm{free}(\mathrm{A})\backslash\{X\}
\end{aligned}
$$

- **Definition 2.11.** We call a formula A closed or ground, iff $\mathrm{free}(\mathrm{A}) = \emptyset$. We call a closed proposition a sentence, and denote the set of all ground term with $cwff_\iota(\Sigma_\iota)$ and the set of sentences with $cwff_o(\Sigma_\iota)$.
- **Axiom 2.12.** *Bound variables can be renamed, i.e. any subterm $\forall X.\mathrm{B}$ of a formula A can be replaced by $\mathrm{A}' := (\forall Y.\mathrm{B}')$, where B' arises from B by replacing all $X \in \mathrm{free}(\mathrm{B})$ with a new variable Y that does not occur in A. We call A' an alphabetical variant of A – and the other way around too.*

SOME RIGHTS RESERVED

Semantics of PL^1 (Models)

- **Definition 2.13.** We inherit the domain $\mathcal{D}_0 = \{\mathsf{T}, \mathsf{F}\}$ of truth values from PL^0 and assume an arbitrary domain $\mathcal{D}_\iota \neq \emptyset$ of individuals.(this choice is a parameter to the semantics)
- **Definition 2.14.** An interpretation $\mathcal{I}$ assigns values to constants, e.g.
  - $\mathcal{I}(\neg)\colon \mathcal{D}_0 \to \mathcal{D}_0$ with $\mathsf{T} \mapsto \mathsf{F}$, $\mathsf{F} \mapsto \mathsf{T}$, and $\mathcal{I}(\wedge) = \ldots$ (as in PL^0)
  - $\mathcal{I}\colon \Sigma^f_k \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (interpret function symbols as arbitrary functions)
  - $\mathcal{I}\colon {\Sigma^p}_k \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicates as arbitrary relations)
- **Definition 2.15.** A variable assignment $\varphi\colon \mathcal{V}_\iota \to \mathcal{D}_\iota$ maps variables into the domain.
- **Definition 2.16.** A model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$ of PL^1 consists of a domain $\mathcal{D}_\iota$ and an interpretation $\mathcal{I}$.

Semantics of PL^1 (Evaluation)

- **Definition 2.17.** Given a model $\langle\mathcal{D},\mathcal{I}\rangle$, the value function $\mathcal{I}_\varphi$ is recursively defined: (two parts: terms & propositions)
  - $\mathcal{I}_\varphi\colon wf\!f_\iota(\Sigma_1,\mathcal{V}_\iota)\to\mathcal{D}_\iota$ assigns values to terms.
    - $\mathcal{I}_\varphi(X):=\varphi(X)$ and
    - $\mathcal{I}_\varphi(f(\mathrm{A}_1,\ldots,\mathrm{A}_k)):=\mathcal{I}(f)(\mathcal{I}_\varphi(\mathrm{A}_1),\ldots,\mathcal{I}_\varphi(\mathrm{A}_k))$
  - $\mathcal{I}_\varphi\colon wf\!f_o(\Sigma_1,\mathcal{V}_\iota)\to\mathcal{D}_0$ assigns values to formulae:
    - $\mathcal{I}_\varphi(T)=\mathcal{I}(T)=\mathsf{T}$,
    - $\mathcal{I}_\varphi(\neg\mathrm{A})=\mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathrm{A}))$
    - $\mathcal{I}_\varphi(\mathrm{A}\wedge\mathrm{B})=\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathrm{A}),\mathcal{I}_\varphi(\mathrm{B}))$ (just as in PL^0)
    - $\mathcal{I}_\varphi(p(\mathrm{A}_1,\ldots,\mathrm{A}_k)):=\mathsf{T}$, iff $\langle\mathcal{I}_\varphi(\mathrm{A}_1),\ldots,\mathcal{I}_\varphi(\mathrm{A}_k)\rangle\in\mathcal{I}(p)$
    - $\mathcal{I}_\varphi(\forall X.\mathrm{A}):=\mathsf{T}$, iff $\mathcal{I}_{\varphi,[\mathsf{a}/X]}(\mathrm{A})=\mathsf{T}$ for all $\mathsf{a}\in\mathcal{D}_\iota$.
- **Definition 2.18 (Assignment Extension).** Let φ be a variable assignment into D and $a\in D$, then $\varphi,[a/X]$ is called the extension of φ with $[a/X]$ and is defined as $\{(Y,a)\in\varphi\,|\,Y\neq X\}\cup\{(X,a)\}$: $\varphi,[a/X]$ coincides with φ off X, and gives the result a there.

SOME RIGHTS RESERVED

Semantics Computation: Example

- **Example 2.19.** We define an instance of first-order logic:
  - Signature: Let $\Sigma_0^f := \{j, m\}$, $\Sigma_1^f := \{f\}$, and ${\Sigma^p}_2 := \{o\}$
  - Universe: $\mathcal{D}_\iota := \{J, M\}$
  - Interpretation: $\mathcal{I}(j) := J$, $\mathcal{I}(m) := M$, $\mathcal{I}(f)(J) := M$, $\mathcal{I}(f)(M) := M$, and $\mathcal{I}(o) := \{(M,J)\}$.

  Then $\forall X.o(f(X), X)$ is a sentence and with $\psi := \varphi,[\mathsf{a}/X]$ for $\mathsf{a} \in \mathcal{D}_\iota$ we have

$$\begin{aligned}
\mathcal{I}_\varphi(\forall X.o(f(X), X)) = \mathsf{T} \quad & \text{iff} \quad \mathcal{I}_\psi(o(f(X), X)) = \mathsf{T} \text{ for all } \mathsf{a} \in \mathcal{D}_\iota \\
& \text{iff} \quad (\mathcal{I}_\psi(f(X)),\mathcal{I}_\psi(X)) \in \mathcal{I}(o) \text{ for all } \mathsf{a} \in \{J, M\} \\
& \text{iff} \quad (\mathcal{I}(f)(\mathcal{I}_\psi(X)),\psi(X)) \in \{(M,J)\} \text{ for all } \mathsf{a} \in \{J, M\} \\
& \text{iff} \quad (\mathcal{I}(f)(\psi(X)),a) = (M,J) \text{ for all } \mathsf{a} \in \{J, M\} \\
& \text{iff} \quad \mathcal{I}(f)(a) = M \text{ and } a = J \text{ for all } \mathsf{a} \in \{J, M\}
\end{aligned}$$

But $\mathsf{a} \neq J$ for $\mathsf{a} = M$, so $\mathcal{I}_\varphi(\forall X.o(f(X), X)) = \mathsf{F}$ in the model $\langle \mathcal{D}_\iota, \mathcal{I} \rangle$.

SOME RIGHTS RESERVED

14.2.2 First-Order Substitutions

Substitutions on Terms

- **Intuition:** If B is a term and X is a variable, then we denote the result of systematically replacing all occurrences of X in a term A by B with $[B/X](A)$.
- **Problem:** What about $[Z/Y],[Y/X](X)$, is that Y or Z?
- **Folklore:** $[Z/Y],[Y/X](X) = Y$, but $[Z/Y]([Y/X](X)) = Z$ of course. (Parallel application)
- $[\frac{t}{x}]$

  $[t/s]$

  Definition 2.20. Let $wfe(\Sigma,\mathcal{V})$ be an expression language, then we call $\sigma\colon \mathcal{V} \to wfe(\Sigma,\mathcal{V})$ a substitution, iff the support $\mathrm{supp}(\sigma) := \{X \mid (X,A) \in \sigma, X \neq A\}$ of σ is finite. We denote the empty substitution with ϵ.
- **Definition 2.21 (Substitution Application).** We define substitution application by
  - $\sigma(c) = c$ for $c \in \Sigma$
  - $\sigma(X) = A$, iff $X \in \mathcal{V}$ and $(X,A) \in \sigma$.
  - $\sigma(f(A_1,\ldots,A_n)) = f(\sigma(A_1),\ldots,\sigma(A_n))$,
  - $\sigma(\forall X.A) = \forall X.\sigma_{-X}(A)$. ($\exists$ analogous)
- **Example 2.22.** $[a/x],[f(b)/y],[a/z]$ instantiates $g(x,y,h(z))$ to $g(a,f(b),h(a))$.

Substitution Extension

- **Definition 2.23 (Substitution Extension).** Let σ be a substitution, then we denote the extension of σ with $[\mathrm{A}/X]$ by $\sigma,[\mathrm{A}/X]$ and define it as $\{(Y,\mathrm{B}) \in \sigma \mid Y \neq X\} \cup \{(X,\mathrm{A})\}$: $\sigma,[\mathrm{A}/X]$ coincides with σ off X, and gives the result A there.
- **Note:** If σ is a substitution, then $\sigma,[\mathrm{A}/X]$ is also a substitution.
- We also need the dual operation: removing a variable from the support:
- **Definition 2.24.** We can discharge a variable X from a substitution σ by setting $\sigma_{-X}{:=}\sigma,[X/X]$.

SOME RIGHTS RESERVED

Substitutions on Propositions

- **Problem:** We want to extend substitutions to propositions, in particular to quantified formulae: What is $\sigma(\forall X.A)$?
- **Idea:** σ should not instantiate bound variables. ($[A/X](\forall X.B) = \forall A.B'$ ill-formed)
- **Definition 2.25.** $\sigma(\forall X.A) := (\forall X.\sigma_{-X}(A))$.
- **Problem:** This can lead to variable capture: $[f(X)/Y](\forall X.p(X,Y))$ would evaluate to $\forall X.p(X,f(X))$, where the second occurrence of X is bound after instantiation, whereas it was free before. **Solution:** Rename away the bound variable X in $\forall X.p(X,Y)$ before applying the substitution.
- **Definition 2.26 (Capture-Avoiding Substitution Application).** Let σ be a substitution, A a formula, and A′ an alphabetic variant of A, such that $\mathrm{intro}(\sigma) \cap \mathrm{BVar}(A) = \emptyset$. Then we define capture-avoiding substitution application via $\sigma(A) := \sigma(A')$.

SOME RIGHTS RESERVED

Substitution Value Lemma for Terms

- **Lemma 2.27.** *Let* A *and* B *be terms, then* $\mathcal{I}_\varphi([\mathrm{B}/X]\mathrm{A}) = \mathcal{I}_\psi(\mathrm{A})$, *where* $\psi = \varphi, [\mathcal{I}_\varphi(\mathrm{B})/X]$.
- *Proof:* by induction on the depth of A:

  1. depth=0

     Then A *is a variable (say* Y*), or constant, so we have three cases*

     1.1. $\mathrm{A} = Y = X$

        1.1.1. then

        $\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A})) = \mathcal{I}_\varphi([\mathrm{B}/X](X)) = \mathcal{I}_\varphi(\mathrm{B}) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(\mathrm{A})$.

     1.3. $\mathrm{A} = Y \neq X$

        1.3.1. then $\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A})) = \mathcal{I}_\varphi([\mathrm{B}/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(\mathrm{A})$.

     1.5. A is a constant

        1.5.1. Analogous to the preceding case ($Y \neq X$).

     1.7. This completes the base case (depth = 0).

  3. depth>0

     3.1. then $\mathrm{A} = f(\mathrm{A}_1, \ldots, \mathrm{A}_n)$ and we have

$$
\begin{aligned}
\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A})) &= \mathcal{I}(f)(\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A}_1)), \ldots, \mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A}_n))) \\
&= \mathcal{I}(f)(\mathcal{I}_\psi(\mathrm{A}_1), \ldots, \mathcal{I}_\psi(\mathrm{A}_n)) \\
&= \mathcal{I}_\psi(\mathrm{A}).
\end{aligned}
$$

Substitution Value Lemma for Propositions

- **Lemma 2.28.** $\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A})) = \mathcal{I}_\psi(\mathrm{A})$, *where* $\psi = \varphi,[\mathcal{I}_\varphi(\mathrm{B})/X]$.
- *Proof:* by induction on the number n of connectives and quantifiers in A:
  1. $n = 0$
     - 1.1. then A is an atomic proposition, and we can argue like in the induction step of the substitution value lemma for terms.
  3. $n > 0$ and $\mathrm{A} = \neg\mathrm{B}$ or $\mathrm{A} = \mathrm{C} \circ \mathrm{D}$
     - 3.1. Here we argue like in the induction step of the term lemma as well.
  5. $n > 0$ and $\mathrm{A} = \forall Y.\mathrm{C}$ where (WLOG) $X \neq Y$ *(otherwise rename)*
     - 5.1. then $\mathcal{I}_\psi(\mathrm{A}) = \mathcal{I}_\psi(\forall Y.\mathrm{C}) = \mathsf{T}$, iff $\mathcal{I}_{\psi,[a/Y]}(\mathrm{C}) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$.
     - 5.2. But $\mathcal{I}_{\psi,[a/Y]}(\mathrm{C}) = \mathcal{I}_{\varphi,[a/Y]}([\mathrm{B}/X](\mathrm{C})) = \mathsf{T}$, by induction hypothesis.
     - 5.3. So $\mathcal{I}_\psi(\mathrm{A}) = \mathcal{I}_\varphi(\forall Y.[\mathrm{B}/X](\mathrm{C})) = \mathcal{I}_\varphi([\mathrm{B}/X](\forall Y.\mathrm{C})) = \mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A}))$

☐

SOME RIGHTS RESERVED

14.3 First-Order Natural Deduction

SOME RIGHTS RESERVED

First-Order Natural Deduction ($\mathcal{ND}^1$; Gentzen [**Gentzen:uudlsi35**])

- Rules for connectives just as always
- **Definition 3.1 (New Quantifier Rules).** The first-order natural deduction calculus $\mathcal{ND}^1$ extends $\mathcal{ND}_0$ by the following four rules:

$$\frac{\mathrm{A}}{\forall X.\mathrm{A}}\,\forall I^{*} \qquad \frac{\forall X.\mathrm{A}}{[\mathrm{B}/X](\mathrm{A})}\,\forall E$$

$$\frac{[\mathrm{B}/X](\mathrm{A})}{\exists X.\mathrm{A}}\,\exists I \qquad \frac{\exists X.\mathrm{A} \quad \begin{array}{c}[[c/X](\mathrm{A})]^{1}\\ \vdots\\ \mathrm{C}\end{array} \quad c \in \Sigma_0^{sk} \text{ new}}{\mathrm{C}}\,\exists E^{1}$$

* means that A does not depend on any hypothesis in which X is free.

First-Order Natural Deduction in Sequent Formulation

- Rules for connectives from $\mathcal{ND}^0_\vdash$
- **Definition 3.2 (New Quantifier Rules).** The inference rules of the first-order sequent style ND calculus $\mathcal{ND}^1_\vdash$ consist of those from $\mathcal{ND}^0_\vdash$ plus the following quantifier rules:

$$\frac{\Gamma \vdash \mathrm{A} \quad X \notin \mathrm{free}(\Gamma)}{\Gamma \vdash \forall X.\mathrm{A}}\ \forall I \qquad \frac{\Gamma \vdash \forall X.\mathrm{A}}{\Gamma \vdash [\mathrm{B}/X](\mathrm{A})}\ \forall E$$

$$\frac{\Gamma \vdash [\mathrm{B}/X](\mathrm{A})}{\Gamma \vdash \exists X.\mathrm{A}}\ \exists I \qquad \frac{\Gamma \vdash \exists X.\mathrm{A} \quad \Gamma, [c/X](\mathrm{A}) \vdash \mathrm{C} \quad c \in \Sigma^{sk}_0 \text{ new}}{\Gamma \vdash \mathrm{C}}\ \exists E$$

Natural Deduction with Equality

- **Definition 3.3 (First-Order Logic with Equality).** We extend PL^1 with a new logical constant for equality $= \in {\Sigma^p}_2$ and fix its interpretation to $\mathcal{I}(=) := \{(x,x) \mid x \in \mathcal{D}_\iota\}$. We call the extended logic first-order logic with equality ($\mathrm{PL}^1_=$)
- We now extend natural deduction as well.
- **Definition 3.4.** For the calculus of natural deduction with equality ($\mathcal{ND}^1_=$) we add the following two rules to $\mathcal{ND}^1$ to deal with equality:

$$\frac{}{A = A} = I \qquad \frac{A = B \quad C\,[A]_p}{[B/p]C} = E$$

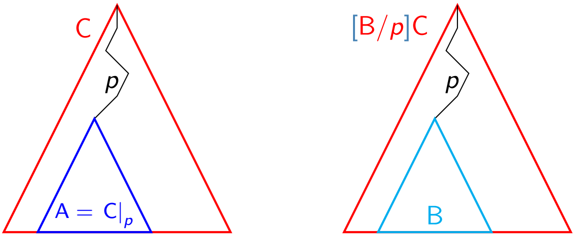
  where $C\,[A]_p$ if the formula C has a subterm A at position p and $[B/p]C$ is the result of replacing that subterm with B.
- In many ways equivalence behaves like equality, we will use the following rules in $\mathcal{ND}^1$
- **Definition 3.5.** $\Leftrightarrow I$ is derivable and $\Leftrightarrow E$ is admissible in $\mathcal{ND}^1$:

$$\frac{}{A \Leftrightarrow A} \Leftrightarrow I \qquad \frac{A \Leftrightarrow B \quad C\,[A]_p}{[B/p]C} \Leftrightarrow E$$

SOME RIGHTS RESERVED

Positions in Formulae

- **Idea:** Formulae are (naturally) trees, so we can use tree positions to talk about subformulae
- **Definition 3.7.** A position p is a tuple of natural numbers that in each node of an expression (tree) specifies into which child to descend. For an expression A we denote the subexpression at p with $\mathrm{A}|_p$.
  We will sometimes write an expression C as $\mathrm{C}\,[\mathrm{A}]_p$ to indicate that C the subexpression A at position p.
  If $\mathrm{C}\,[\mathrm{A}]_p$ and A is atomic, then we speak of an occurrence of A in C.
- **Definition 3.8.** Let p be a position, then $[\mathrm{A}/p]\mathrm{C}$ is the expression obtained from C by replacing the subexpression at p by A.
- **Example 3.9 (Schematically).**



$\mathcal{ND}^1_=$ Example: $\sqrt{2}$ is Irrational

- We can do real mathematics with $\mathcal{ND}^1_=$:
- **Theorem 3.10.** $\sqrt{2}$ *is irrational*

  Proof: We prove the assertion by contradiction

  1. Assume that $\sqrt{2}$ is rational.
  2. Then there are numbers p and q such that $\sqrt{2} = p/q$.
  3. So we know $2q^2 = p^2$.
  4. But $2q^2$ has an odd number of prime factors while p^2 an even number.
  5. This is a contradiction (since they are equal), so we have proven the assertion

  □

SOME RIGHTS RESERVED

$\mathcal{ND}^1_=$ Example: $\sqrt{2}$ is Irrational (the Proof)

#	hyp	formula	NDjust
1		$\forall n, m.\neg(2n+1) = (2m)$	lemma
2		$\forall n, m.\#(n^m) = m\#(n)$	lemma
3		$\forall n, p.\mathrm{prime}(p) \Rightarrow \#(pn) = (\#(n)+1)$	lemma
4		$\forall x.\mathrm{irr}(x) \Leftrightarrow \neg(\exists p, q.x = p/q)$	definition
5		$\mathrm{irr}(\sqrt{2}) \Leftrightarrow \neg(\exists p, q.\sqrt{2} = p/q)$	$\forall E(4)$
6	6	$\neg\mathrm{irr}(\sqrt{2})$	Ax
7	6	$\neg\neg(\exists p, q.\sqrt{2} = p/q)$	$\Leftrightarrow E(6,5)$
8	6	$\exists p, q.\sqrt{2} = p/q$	$\neg E(7)$
9	6,9	$\sqrt{2} = p/q$	Ax
10	6,9	$2q^2 = p^2$	arith(9)
11	6,9	$\#(p^2) = 2\#(p)$	$\forall E^2(2)$
12	6,9	$\mathrm{prime}(2) \Rightarrow \#(2q^2) = (\#(q^2)+1)$	$\forall E^2(1)$

SOME RIGHTS RESERVED

$\mathcal{ND}^1_=$ Example: $\sqrt{2}$ is Irrational (the Proof continued)

13		$\mathrm{prime}(2)$	lemma
14	6,9	$\#(2q^2) = \#(q^2) + 1$	$\Rightarrow E(13, 12)$
15	6,9	$\#(q^2) = 2\#(q)$	$\forall E^2(2)$
16	6,9	$\#(2q^2) = 2\#(q) + 1$	$= E(14, 15)$
17		$\#(p^2) = \#(p^2)$	$= I$
18	6,9	$\#(2q^2) = \#(q^2)$	$= E(17, 10)$
19	6.9	$2\#(q) + 1 = \#(p^2)$	$= E(18, 16)$
20	6.9	$2\#(q) + 1 = 2\#(p)$	$= E(19, 11)$
21	6.9	$\neg(2\#(q) + 1) = (2\#(p))$	$\forall E^2(1)$
22	6,9	F	$FI(20, 21)$
23	6	F	$\exists E^6(22)$
24		$\neg\neg\mathrm{irr}(\sqrt{2})$	$\neg I^6(23)$
25		$\mathrm{irr}(\sqrt{2})$	$\neg E^2(23)$

SOME RIGHTS RESERVED

14.4 Conclusion

SOME RIGHTS RESERVED

Summary (Predicate Logic)

- First-order logic allows to explicitly speak about objects and their properties. It is thus a more natural and compact representation language than propositional logic; it also enables us to speak about infinite sets of objects.
- Logic has thousands of years of history. A major current application in AI is *semantic technology*. (up soon)
- First-order logic (PL^1) allows universal and existential quantifier quantification over individuals.
- A PL^1 model consists of a universe $\mathcal{D}_\iota$ and a function $\mathcal{I}$ mapping individual constants/predicate constants/function constants to elements/relations/functions on $\mathcal{D}_\iota$.
- First-order natural deduction is a sound and complete calculus for PL^1 intended and optimized for human understanding.

SOME RIGHTS RESERVED

Applications for $\mathcal{ND}^1$ (and extensions)

- **Recap:** We can express mathematical theorems in PL^1 and prove them in $\mathcal{ND}^1$.
- **Problem:** These proofs can be huge (giga-steps), how can we trust them?

SOME RIGHTS RESERVED

Applications for $\mathcal{ND}^1$ (and extensions)

- **Recap:** We can express mathematical theorems in PL^1 and prove them in $\mathcal{ND}^1$.
- **Problem:** These proofs can be huge (giga-steps), how can we trust them?
- **Definition 4.3.** A proof checker for a calculus $\mathcal{C}$ is a program that reads (a formal representation) of a $\mathcal{C}$-proof $\mathcal{P}$ and performs proof-checking, i.e. it checks whether all rule applications in $\mathcal{P}$ are (syntactically) correct.
- **Remark:** Proof-checking goes step-by-step $\rightsquigarrow$ proof checkers run in linear time.

SOME RIGHTS RESERVED

Applications for $\mathcal{ND}^1$ (and extensions)

- **Recap:** We can express mathematical theorems in PL^1 and prove them in $\mathcal{ND}^1$.
- **Problem:** These proofs can be huge (giga-steps), how can we trust them?
- **Definition 4.5.** A proof checker for a calculus $\mathcal{C}$ is a program that reads (a formal representation) of a $\mathcal{C}$-proof $\mathcal{P}$ and performs proof-checking, i.e. it checks whether all rule applications in $\mathcal{P}$ are (syntactically) correct.
- **Remark:** Proof-checking goes step-by-step $\leadsto$ proof checkers run in linear time.
- **Idea:** If the logic can express (safety)-properties of programs, we can use proof checkers for formal program verification. (there are extensions of PL^1 that can)
- **Problem:** These proofs can be humongous, how can humans write them?

SOME RIGHTS RESERVED

Applications for $\mathcal{ND}^1$ (and extensions)

- **Recap:** We can express mathematical theorems in PL^1 and prove them in $\mathcal{ND}^1$.
- **Problem:** These proofs can be huge (giga-steps), how can we trust them?
- **Definition 4.7.** A proof checker for a calculus $\mathcal{C}$ is a program that reads (a formal representation) of a $\mathcal{C}$-proof $\mathcal{P}$ and performs proof-checking, i.e. it checks whether all rule applications in $\mathcal{P}$ are (syntactically) correct.
- **Remark:** Proof-checking goes step-by-step $\leadsto$ proof checkers run in linear time.
- **Idea:** If the logic can express (safety)-properties of programs, we can use proof checkers for formal program verification. (there are extensions of PL^1 that can)
- **Problem:** These proofs can be humongous, how can humans write them?
- **Idea:** Automate proof construction via
  - lemma/theorem libraries that collect useful intermediate results
  - tactics $\widehat{=}$ subroutines that construct recurring sub-proofs
  - calls to automated theorem prover (ATP) (next chapter)

  Proof checkers that do any/all of these are called proof assistants.

SOME RIGHTS RESERVED

Applications for $\mathcal{ND}^1$ (and extensions)

- **Recap:** We can express mathematical theorems in PL^1 and prove them in $\mathcal{ND}^1$.
- **Problem:** These proofs can be huge (giga-steps), how can we trust them?
- **Definition 4.9.** A proof checker for a calculus $\mathcal{C}$ is a program that reads (a formal representation) of a $\mathcal{C}$-proof $\mathcal{P}$ and performs proof-checking, i.e. it checks whether all rule applications in $\mathcal{P}$ are (syntactically) correct.
- **Remark:** Proof-checking goes step-by-step $\rightsquigarrow$ proof checkers run in linear time.
- **Idea:** If the logic can express (safety)-properties of programs, we can use proof checkers for formal program verification. (there are extensions of PL^1 that can)
- **Problem:** These proofs can be humongous, how can humans write them?
- **Idea:** Automate proof construction via
  - lemma/theorem libraries that collect useful intermediate results
  - tactics $\widehat{=}$ subroutines that construct recurring sub-proofs
  - calls to automated theorem prover (ATP) (next chapter)

  Proof checkers that do any/all of these are called proof assistants.
- **Definition 4.10.** Formal methods are logic-based techniques for the specification, development, analysis, and verification of software and hardware.
- Formal methods is a major (industrial) application of AI/logic technology.

SOME RIGHTS RESERVED

Chapter 15
Automated Theorem Proving in First-Order Logic

SOME RIGHTS RESERVED

15.1 First-Order Inference with Tableaux

15.1.1 First-Order Tableau Calculi

SOME RIGHTS RESERVED

Test Calculi: Tableaux and Model Generation

- **Idea:** A tableau calculus is a test calculus that
  - analyzes a labeled formulae in a tree to determine satisfiability,
  - its branches correspond to valuations ($\leadsto$ models).
- **Example 1.1.** Tableau calculi try to construct models for labeled formulae: E.g. the propositional tableau calculus for PL^0

Tableau refutation (Validity)	Model generation (Satisfiability)
$\models P \wedge Q \Rightarrow Q \wedge P$	$\models P \wedge (Q \vee \neg R) \wedge \neg Q$
$(P \wedge Q \Rightarrow Q \wedge P)^{\mathsf{F}}$ $(P \wedge Q)^{\mathsf{T}}$ $(Q \wedge P)^{\mathsf{F}}$ P^{T} Q^{T} $P^{\mathsf{F}} \mid Q^{\mathsf{F}}$ $\bot \mid \bot$	$(P \wedge (Q \vee \neg R) \wedge \neg Q)^{\mathsf{T}}$ $(P \wedge (Q \vee \neg R))^{\mathsf{T}}$ $\neg Q^{\mathsf{T}}$ Q^{F} P^{T} $(Q \vee \neg R)^{\mathsf{T}}$ $Q^{\mathsf{T}} \mid \neg R^{\mathsf{T}}$ $\bot \mid R^{\mathsf{F}}$
No Model	Herbrand valuation $\{P^{\mathsf{T}}, Q^{\mathsf{F}}, R^{\mathsf{F}}\}$ $\varphi := \{P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}, R \mapsto \mathsf{F}\}$

- **Idea:** Open branches in saturated tableaux yield satisfying assignments.
- **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)
  - Satisfiable, iff there are open branches (correspond to models)

Analytical Tableaux (Formal Treatment of $\mathcal{T}_0$)

- **Idea:** A test calculus where
  - A labeled formula is analyzed in a tree to determine satisfiability,
  - branches correspond to valuations (models)
- **Definition 1.2.** The propositional tableau calculus $\mathcal{T}_0$ has two inference rules per connective (one for each possible label)

$$\frac{(A \wedge B)^{\mathsf{T}}}{\begin{array}{c}A^{\mathsf{T}}\\ B^{\mathsf{T}}\end{array}}\,\mathcal{T}_0\wedge \qquad \frac{(A \wedge B)^{\mathsf{F}}}{A^{\mathsf{F}} \;\Big|\; B^{\mathsf{F}}}\,\mathcal{T}_0\vee \qquad \frac{\neg A^{\mathsf{T}}}{A^{\mathsf{F}}}\,\mathcal{T}_0\neg^{\mathsf{T}} \qquad \frac{\neg A^{\mathsf{F}}}{A^{\mathsf{T}}}\,\mathcal{T}_0\neg^{\mathsf{F}} \qquad \frac{\begin{array}{c}A^{\alpha}\\ A^{\beta}\end{array} \quad \alpha \neq \beta}{\perp}\,\mathcal{T}_0\perp$$

  Use rules exhaustively as long as they contribute new material (⇝ termination)
- **Definition 1.3.** We call any tree ($\Big|$ introduces branches) produced by the $\mathcal{T}_0$ inference rules from a set Φ of labeled formulae a tableau for Φ.
- **Definition 1.4.** Call a tableau saturated, iff no rule adds new material and a branch closed, iff it ends in $\perp$, else open. A tableau is closed, iff all of its branches are.

  In analogy to the $\perp$ at the end of closed branches, we sometimes decorate open branches with a $\Box$ symbol.

Analytical Tableaux ($\mathcal{T}_0$ continued)

- **Definition 1.6 ($\mathcal{T}_0$-Theorem/Derivability).** A is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0}$A), iff there is a closed tableau with A^F at the root.
  $\Phi \subseteq wff_0(\mathcal{V}_0)$ derives A in $\mathcal{T}_0$ ($\Phi \vdash_{\mathcal{T}_0} A$), iff there is a closed tableau starting with A^F and Φ^T. The tableau with only a branch of A^F and Φ^T is called initial for $\Phi \vdash_{\mathcal{T}_0} A$.

SOME RIGHTS RESERVED

First-Order Standard Tableaux ($\mathcal{T}_1$)

- **Definition 1.8.** The standard tableau calculus ($\mathcal{T}_1$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the following quantifier rules:

$$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \quad \mathbf{C} \in cwff_\iota(\Sigma_\iota)}{([\mathbf{C}/X](\mathbf{A}))^{\mathsf{T}}} \, \mathcal{T}_1\forall \qquad \frac{(\forall X.\mathbf{A})^{\mathsf{F}} \quad c \in \Sigma_0^{sk} \text{ new}}{([c/X](\mathbf{A}))^{\mathsf{F}}} \, \mathcal{T}_1\exists$$

- **Problem:** The rule $\mathcal{T}_1\forall$ displays a case of "don't know indeterminism": to find a refutation we have to guess a formula C from the (usually infinite) set $cwff_\iota(\Sigma_\iota)$.
  For proof search, this means that we have to systematically try all, so $\mathcal{T}_1\forall$ is infinitely branching in general.

Free variable Tableaux ($\mathcal{T}_1^f$)

- **Definition 1.9.** The free variable tableau calculus ($\mathcal{T}_1^f$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the quantifier rules:

  $$\frac{(\forall X.\mathrm{A})^{\mathsf{T}} \quad Y \text{ new}}{([Y/X](\mathrm{A}))^{\mathsf{T}}}\,\mathcal{T}_1^f\forall \qquad \frac{(\forall X.\mathrm{A})^{\mathsf{F}} \quad \mathrm{free}(\forall X.\mathrm{A}) = \{X^1,\ldots,X^k\} \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X^1,\ldots,X^k)/X](\mathrm{A}))^{\mathsf{F}}}\,\mathcal{T}_1^f\exists$$

  and generalizes its cut rule $\mathcal{T}_0\bot$ to:

  $$\frac{\begin{matrix}\mathrm{A}^{\alpha}\\ \mathrm{B}^{\beta}\end{matrix} \quad \alpha \neq \beta \quad \sigma(\mathrm{A}) = \sigma(\mathrm{B})}{\bot : \sigma}\,\mathcal{T}_1^f\bot$$

  $\mathcal{T}_1^f\bot$ instantiates the whole tableau by σ.
- **Advantage:** No guessing necessary in $\mathcal{T}_1^f\forall$-rule!
- **New Problem:** find suitable substitution (most general unifier) (later)

SOME RIGHTS RESERVED

Free variable Tableaux ($\mathcal{T}_1^f$): Derivable Rules

- **Definition 1.10.** Derivable quantifier rules in $\mathcal{T}_1^f$:

$$\frac{(\exists X.A)^{\mathsf{T}} \quad \mathrm{free}(\forall X.A) = \{X^1, \ldots, X^k\} \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X^1, \ldots, X^k)/X](A))^{\mathsf{T}}}$$

$$\frac{(\exists X.A)^{\mathsf{F}} \quad Y \text{ new}}{([Y/X](A))^{\mathsf{F}}}$$

SOME RIGHTS RESERVED

Tableau Reasons about Blocks

- **Example 1.11 (Reasoning about Blocks).** Returing to slide 409

Can we prove $\mathrm{red}(\mathsf{A})$ from $\forall x.\mathrm{block}(x) \Rightarrow \mathrm{red}(x)$ and $\mathrm{block}(\mathsf{A})$?

$$\begin{array}{c}
(\forall X.\mathrm{block}(X) \Rightarrow \mathrm{red}(X))^{\mathsf{T}}\\
\mathrm{block}(\mathsf{A})^{\mathsf{T}}\\
\mathrm{red}(\mathsf{A})^{\mathsf{F}}\\
(\mathrm{block}(Y) \Rightarrow \mathrm{red}(Y))^{\mathsf{T}}\\
\begin{array}{c|c}
\mathrm{block}(Y)^{\mathsf{F}} & \mathrm{red}(\mathsf{A})^{\mathsf{T}}\\
\bot : [\mathsf{A}/Y] & \bot
\end{array}
\end{array}$$

SOME RIGHTS RESERVED

15.1.2 First-Order Unification

SOME RIGHTS RESERVED

Unification (Definitions)

- **Definition 1.12.** For given terms $\mathbb{A}_1, \ldots, \mathbb{A}_n$, unification is the problem of finding a substitution σ (called unifier), such that $\sigma(\mathbb{A}_1) = \ldots = \sigma(\mathbb{A}_n)$.
- **Notation:** We write pairs as $\mathbb{A}_1 =^? \ldots =^? \mathbb{A}_n$ e.g. $f(X) =^? f(g(Y))$.
- **Definition 1.13.** Solutions (e.g. $[g(a)/X], [a/Y]$, $[g(g(a))/X], [g(a)/Y]$, or $[g(Z)/X], [Z/Y]$) are called unifiers, $\mathrm{U}(\mathbb{A}_1, \ldots, \mathbb{A}_n) := \{\sigma \mid \sigma(\mathbb{A}_1) = \ldots = \sigma(\mathbb{A}_n)\}$.
- **Idea:** Find representatives in $\mathrm{U}(\mathbb{A}_1, \ldots, \mathbb{A}_n)$, that generate the set of solutions.
- **Definition 1.14.** Let σ and θ be substitutions and $W \subseteq \mathcal{V}_\iota$, we say that a substitution σ is more general than θ (on W; write $\sigma \leq \theta[W]$), iff there is a substitution ρ, such that $\theta = \rho \circ \sigma[W]$, where $\sigma = \rho[W]$, iff $\sigma(X) = \rho(X)$ for all $X \in W$.
- **Definition 1.15.** σ is called a most general unifier (mgu) of $\mathbb{A}_1, \ldots, \mathbb{A}_n$, iff it is minimal in $\mathrm{U}(\mathbb{A}_1, \ldots, \mathbb{A}_n)$ wrt. $\leq[\mathrm{free}(\mathbb{A}_1) \cup \ldots \cup \mathrm{free}(\mathbb{A}_n)]$.

SOME RIGHTS RESERVED

Unification Problems ($\widehat{=}$ Equational Systems)

- **Idea:** Unification is equation solving.
- **Definition 1.16.** We call a formula $A^1 =^? B^1 \wedge \ldots \wedge A^n =^? B^n$ an unification problem iff $A^i, B^i \in \mathit{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$.
- **Note:** We consider unification problems as sets of equations ($\wedge$ is ACI), and equations as two-element multisets ($=^?$ is C).
- **Definition 1.17.** A substitution is called a unifier for a unification problem $\mathcal{E}$ (and thus $\mathcal{E}$ unifiable), iff it is a (simultaneous) unifier for all pairs in $\mathcal{E}$.

Solved forms and Most General Unifiers

- **Definition 1.18.** We call a pair $\mathsf{A} =^? \mathsf{B}$ solved in a unification problem $\mathcal{E}$, iff $\mathsf{A} = X$, $\mathcal{E} = X =^? \mathsf{A} \wedge \mathcal{E}'$, and $X \notin (\mathrm{free}(\mathsf{A}) \cup \mathrm{free}(\mathcal{E}'))$. We call an unification problem $\mathcal{E}$ a solved form, iff all its pairs are solved.
- **Lemma 1.19.** *Solved forms are of the form* $X^1 =^? \mathsf{B}^1 \wedge \ldots \wedge X^n =^? \mathsf{B}^n$ *where the* X^i *are distinct and* $X^i \notin \mathrm{free}(\mathsf{B}^j)$.
- **Definition 1.20.** Any substitution $\sigma = [\mathsf{B}^1/X^1],\ldots,[\mathsf{B}^n/X^n]$ induces a solved unification problem $\mathcal{E}_\sigma := (X^1 =^? \mathsf{B}^1 \wedge \ldots \wedge X^n =^? \mathsf{B}^n)$.
- **Lemma 1.21.** *If* $\mathcal{E} = X^1 =^? \mathsf{B}^1 \wedge \ldots \wedge X^n =^? \mathsf{B}^n$ *is a solved form, then* $\mathcal{E}$ *has the unique most general unifier* $\sigma_{\mathcal{E}} := [\mathsf{B}^1/X^1],\ldots,[\mathsf{B}^n/X^n]$.
- *Proof:* Let $\theta \in \mathrm{U}(\mathcal{E})$
  1. then $\theta(X^i) = \theta(\mathsf{B}^i) = \theta \circ \sigma_{\mathcal{E}}(X^i)$
  2. and thus $\theta = \theta \circ \sigma_{\mathcal{E}}[\mathrm{supp}(\sigma)]$.

☐

- **Note:** We can rename the introduced variables in most general unifiers!

SOME RIGHTS RESERVED

Unification Algorithm

- **Definition 1.22.** The inference system $\mathcal{U}$ consists of the following rules:

$$\frac{\mathcal{E} \wedge f(\mathrm{A}^1,\ldots,\mathrm{A}^n) =^? f(\mathrm{B}^1,\ldots,\mathrm{B}^n)}{\mathcal{E} \wedge \mathrm{A}^1 =^? \mathrm{B}^1 \wedge \ldots \wedge \mathrm{A}^n =^? \mathrm{B}^n}\,\mathcal{U}\mathrm{dec} \qquad \frac{\mathcal{E} \wedge \mathrm{A} =^? \mathrm{A}}{\mathcal{E}}\,\mathcal{U}\mathrm{triv}$$

$$\frac{\mathcal{E} \wedge X =^? bA \wedge X \notin \mathrm{free}(\mathrm{A}) \wedge X \in \mathrm{free}(\mathcal{E})}{[\mathrm{A}/X](\mathcal{E}) \wedge X =^? \mathrm{A}}\,\mathcal{U}\mathrm{elim}$$

- **Lemma 1.23.** $\mathcal{U}$ *is correct: $\mathcal{E}\vdash_{\mathcal{U}}\mathcal{F}$ implies $\mathrm{U}(\mathcal{F}) \subseteq \mathrm{U}(\mathcal{E})$.*
- **Lemma 1.24.** $\mathcal{U}$ *is complete: $\mathcal{E}\vdash_{\mathcal{U}}\mathcal{F}$ implies $\mathrm{U}(\mathcal{E}) \subseteq \mathrm{U}(\mathcal{F})$.*
- **Lemma 1.25.** $\mathcal{U}$ *is confluent: the order of derivations does not matter.*
- **Corollary 1.26.** *First-order unification is unitary: i.e. most general unifiers are unique up to renaming of introduced variables.*
- *Proof sketch:* $\mathcal{U}$ is trivially branching.

SOME RIGHTS RESERVED

Unification Examples

- **Example 1.27.** Two similar unification problems:

$\begin{array}{cl} f(g(X,X),h(a)) =^? f(g(a,Z),h(Z)) & \\ \hline g(X,X) =^? g(a,Z) \wedge h(a) =^? h(Z) & \mathcal{U}\mathrm{dec} \\ \hline X =^? a \wedge X =^? Z \wedge h(a) =^? h(Z) & \mathcal{U}\mathrm{dec} \\ \hline X =^? a \wedge X =^? Z \wedge a =^? Z & \mathcal{U}\mathrm{dec} \\ \hline X =^? a \wedge a =^? Z \wedge a =^? Z & \mathcal{U}\mathrm{elim} \\ \hline X =^? a \wedge Z =^? a \wedge a =^? a & \mathcal{U}\mathrm{elim} \\ \hline X =^? a \wedge Z =^? a & \mathcal{U}\mathrm{triv} \end{array}$	$\begin{array}{cl} f(g(X,X),h(a)) =^? f(g(b,Z),h(Z)) & \\ \hline g(X,X) =^? g(b,Z) \wedge h(a) =^? h(Z) & \mathcal{U}\mathrm{de} \\ \hline X =^? b \wedge X =^? Z \wedge h(a) =^? h(Z) & \mathcal{U}\mathrm{dec} \\ \hline X =^? b \wedge X =^? Z \wedge a =^? Z & \mathcal{U}\mathrm{dec} \\ \hline X =^? b \wedge b =^? Z \wedge a =^? Z & \mathcal{U}\mathrm{elim} \\ \hline X =^? b \wedge Z =^? b \wedge a =^? b & \mathcal{U}\mathrm{elim} \end{array}$
MGU: $[a/X], [a/Z]$	$a =^? b$ not unifiable

SOME RIGHTS RESERVED

Unification (Termination)

- **Definition 1.28.** Let S and T be multisets and $\leq$ a partial ordering on $S \cup T$. Then we define $S \prec^m S$, iff $S = C \uplus T'$ and $T = C \uplus \{t\}$, where $s \leq t$ for all $s \in S'$. We call $\leq^m$ the multiset ordering induced by $\leq$.
- **Definition 1.29.** We call a variable X solved in an unification problem $\mathcal{E}$, iff $\mathcal{E}$ contains a solved pair $X =^? A$.
- **Lemma 1.30.** *If $\prec$ is linear/terminating On S, then $\prec^m$ is linear/terminating on $\mathcal{P}(S)$.*
- **Lemma 1.31.** *$\mathcal{U}$ is terminating.* (*any $\mathcal{U}$-derivation is finite*)
- *Proof:* We prove termination by mapping $\mathcal{U}$ transformation into a Noetherian space.
  1. Let $\mu(\mathcal{E}) := \langle n, \mathcal{N} \rangle$, where
     - n is the number of unsolved variables in $\mathcal{E}$
     - $\mathcal{N}$ is the multiset of term depths in $\mathcal{E}$
  2. The lexicographic order $\prec$ on pairs $\mu(\mathcal{E})$ is decreased by all inference rules.
     - 2.1. $\mathcal{U}\mathrm{dec}$ and $\mathcal{U}\mathrm{triv}$ decrease the multiset of term depths without increasing the unsolved variables.
     - 2.2. $\mathcal{U}\mathrm{elim}$ decreases the number of unsolved variables (by one), but may increase term depths.

☐

SOME RIGHTS RESERVED

First-Order Unification is Decidable

- **Definition 1.32.** We call an equational problem $\mathcal{E}$ $\mathcal{U}$-reducible, iff there is a $\mathcal{U}$-step $\mathcal{E}\vdash_{\mathcal{U}}\mathcal{F}$ from $\mathcal{E}$.
- **Lemma 1.33.** *If $\mathcal{E}$ is unifiable but not solved, then it is $\mathcal{U}$-reducible.*
- *Proof:* We assume that $\mathcal{E}$ is unifiable but unsolved and show the $\mathcal{U}$ rule that applies.

  1. There is an unsolved pair $\mathrm{A} =^{?} \mathrm{B}$ in $\mathcal{E} = \mathcal{E} \wedge \mathrm{A} =^{?} \mathrm{B}'$.

  we have two cases

  2. $\mathrm{A}, \mathrm{B} \notin \mathcal{V}_{\iota}$
     - 2.1. then $\mathrm{A} = f(\mathrm{A}^1 \ldots \mathrm{A}^n)$ and $\mathrm{B} = f(\mathrm{B}^1 \ldots \mathrm{B}^n)$, and thus $\mathcal{U}\mathrm{dec}$ is applicable
  4. $\mathrm{A} = X \in \mathrm{free}(\mathcal{E})$
     - 4.1. then $\mathcal{U}\mathrm{elim}$ (if $\mathrm{B} \neq X$) or $\mathcal{U}\mathrm{triv}$ (if $\mathrm{B} = X$) is applicable.

  □

- **Corollary 1.34.** *First-order unification is decidable in PL^1.*
- *Proof:*

  1. $\mathcal{U}$-irreducible unification problems can be reached in finite time by 1.31.
  2. They are either solved or unsolvable by 1.33, so they provide the answer.

  □

SOME RIGHTS RESERVED

15.1.3 Efficient Unification

SOME RIGHTS RESERVED

Complexity of Unification

- **Observation:** Naive implementations of unification are exponential in time and space.
- **Example 1.35.** Consider the terms

$$\begin{aligned} s_n &= f(f(x_0,x_0),f(f(x_1,x_1),f(\ldots,f(x_{n-1},x_{n-1}))\ldots)) \\ t_n &= f(x_1,f(x_2,f(x_3,f(\cdots,x_n)\cdots))) \end{aligned}$$

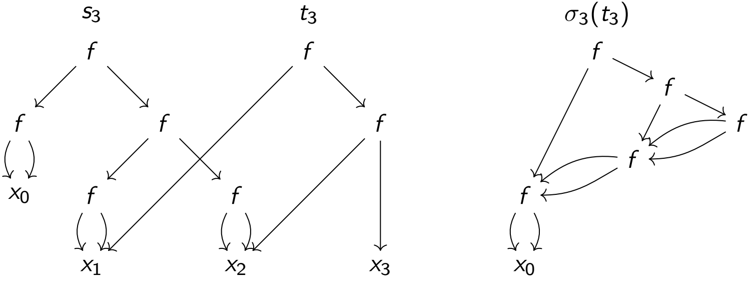
  - The most general unifier of s_n and t_n is

    $\sigma_n := [f(x_0,x_0)/x_1], [f(f(x_0,x_0),f(x_0,x_0))/x_2], [f(f(f(x_0,x_0),f(x_0,x_0)),f(f(x_0,x_0),f(x_0,$

  - It contains $\sum_{i=1}^{n} 2^i = 2^{n+1} - 2$ occurrences of the variable x_0. (exponential)
- **Problem:** The variable x_0 has been copied too often.
- **Idea:** Find a term representation that re-uses subterms.

Directed Acyclic Graphs (DAGs) for Terms

- **Recall:** Terms in first-order logic are essentially trees.
- **Concrete Idea:** Use directed acyclic graphs for representing terms:
  - variables my only occur once in the DAG.
  - subterms can be referenced multiply. (subterm sharing)
  - we can even represent multiple terms in a common DAG
- **Observation 1.36.** *Terms can be transformed into DAGs in linear time.*
- **Example 1.37.** Continuing from ??? …s_3, t_3, and $\sigma_3(s_3)$ as DAGs:



In general: s_n, t_n, and $\sigma_n(s_n)$ only need space in $\mathcal{O}(n)$. (just count)

DAG Unification Algorithm

- **Observation:** In $\mathcal{U}$, the $\mathcal{U}\mathrm{elim}$ rule applies solved pairs ⇝ subterm duplication.
- **Idea:** Replace $\mathcal{U}\mathrm{elim}$ the notion of solved forms by something better.
- **Definition 1.38.** We say that $X^1 =^? \mathsf{B}^1 \wedge \ldots \wedge X^n =^? \mathsf{B}^n$ is a DAG solved form, iff the X^i are distinct and $X^i \notin \mathrm{free}(\mathsf{B}^j)$ for $i \leq j$.
- **Definition 1.39.** The inference system $\mathcal{DU}$ contains rules $\mathcal{U}\mathrm{dec}$ and $\mathcal{U}\mathrm{triv}$ from $\mathcal{U}$ plus the following:

$$\frac{\mathcal{E} \wedge X =^? \mathsf{A} \wedge X =^? \mathsf{B} \quad \mathsf{A}, \mathsf{B} \notin \mathcal{V}_\iota \quad |\mathsf{A}| \leq |\mathsf{B}|}{\mathcal{E} \wedge X =^? \mathsf{A} \wedge \mathsf{A} =^? \mathsf{B}}\ \mathcal{DU}\mathrm{merge}$$

$$\frac{\mathcal{E} \wedge X =^? Y \quad X \neq Y \quad X, Y \in \mathrm{free}(\mathcal{E})}{[Y/X](\mathcal{E}) \wedge X =^? Y}\ \mathcal{DU}\mathrm{evar}$$

  where $|\mathsf{A}|$ is the number of symbols in A.
- The analysis for $\mathcal{U}$ applies mutatis mutandis.

SOME RIGHTS RESERVED

Unification by DAG-chase

- **Idea:** Extend the Input-DAGs by edges that represent unifiers.
- **Definition 1.40.** Write $n.a$, if a is the symbol of node n.
- (standard) auxiliary procedures: (all constant or linear time in DAGs)
  - find(n) follows the path from n and returns the end node.
  - union(n, m) adds an edge between n and m.
  - occur(n, m) determines whether $n.x$ occurs in the DAG with root m.

SOME RIGHTS RESERVED

Algorithm dag−unify

- Input: symmetric pairs of nodes in DAGs

```
fun dag−unify($n$,$n$) = true
  | dag−unify($n.x$,$m$) = if occur($n$,$m$) then true else union($n$,$m$)
  | dag−unify($n.f$,$m.g$) =
      if $g$!=$f$ then false
      else
        forall ($i$,$j$) => dag−unify(find($i$),find($j$)) (chld $m$,chld $n$)
      end
```

- **Observation 1.41.** dag−unify *uses linear space, since no new nodes are created, and at most one link per variable.*
- **Problem:** dag−unify still uses exponential time.
- **Example 1.42.** Consider terms $f(s_n, f(t'_n, x_n)), f(t_n, f(s'_n, y_n)))$, where $s'_n = [y_i/x_i](s_n)$ und $t'_n = [y_i/x_i](t_n)$.
  dag−unify needs exponentially many recursive calls to unify the nodes x_n and y_n. (they are unified after n calls, but checking needs the time)

SOME RIGHTS RESERVED

Algorithm uf−unify

- **Recall:** dag−unify still uses exponential time.
- **Idea:** Also bind the function nodes, if the arguments are unified.

```
uf−unify($n.f$,$m.g$) =
  if $g$!=$f$ then false
  else union($n$,$m$);
    forall ($i$,$j$) => uf−unify(find($i$),find($j$)) (chld $m$,chld $n$)
  end
```

- This only needs linearly many recursive calls as it directly returns with true or makes a node inaccessible for find.
- Linearly many calls to linear procedures give quadratic running time.
- **Remark:** There are versions of uf−unify that are linear in time and space, but for most purposes, our algorithm suffices.

SOME RIGHTS RESERVED

15.1.4 Implementing First-Order Tableaux

SOME RIGHTS RESERVED

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\rightsquigarrow$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.43.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*

SOME RIGHTS RESERVED

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\leadsto$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.48.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.49.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists x.p(x))$.

Start, close left branch	use $\mathcal{T}_1^f\forall$ again (right branch)
$((p(a) \vee p(b)) \Rightarrow (\exists x.p(x)))^F$ $(p(a) \vee p(b))^T$ $(\exists x.p(x))^F$ $(\forall x.\neg p(x))^T$ $\neg p(y)^T$ $p(y)^F$ $\begin{array}{c\|c} p(a)^T & p(b)^T \\ \bot : [a/y] & \end{array}$	$((p(a) \vee p(b)) \Rightarrow (\exists x.p(x)))^F$ $(p(a) \vee p(b))^T$ $(\exists x.p(x))^F$ $(\forall x.\neg p(x))^T$ $\neg p(a)^T$ $p(a)^F$ $\begin{array}{c\|c} p(a)^T & p(b)^T \\ \bot : [a/y] & \neg p(z)^T \\ & p(z)^F \\ & \bot : [b/z] \end{array}$

After we have used up $p(y)^F$ by applying $[a/y]$ in $\mathcal{T}_1^f\bot$, we have to get a new instance $p(z)^F$ via $\mathcal{T}_1^f\forall$.

SOME RIGHTS RESERVED

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once. $\rightsquigarrow$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.53.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.54.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists x.p(x))$.
- **Definition 1.55.** Let $\mathcal{T}$ be a tableau for A, and a positive occurrence of $\forall x.$B in A, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.$B its multiplicity.
- **Observation 1.56.** *Given a prescribed multiplicity for each positive $\forall$, saturation with $\mathcal{T}_1^f$ terminates.*
- *Proof sketch:* All $\mathcal{T}_1^f$ rules reduce the number of connectives and negative $\forall$ or the multiplicity of positive $\forall$.

SOME RIGHTS RESERVED

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\leadsto$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.58.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.59.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists x.p(x))$.
- **Definition 1.60.** Let $\mathcal{T}$ be a tableau for A, and a positive occurrence of $\forall x.$B in A, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.$B its multiplicity.
- **Observation 1.61.** *Given a prescribed multiplicity for each positive $\forall$, saturation with $\mathcal{T}_1^f$ terminates.*
- *Proof sketch:* All $\mathcal{T}_1^f$ rules reduce the number of connectives and negative $\forall$ or the multiplicity of positive $\forall$.
- **Theorem 1.62.** *$\mathcal{T}_1^f$ is only complete with unbounded multiplicities.*
- *Proof sketch:* Replace $p(a) \vee p(b)$ with $p(a_1) \vee \ldots \vee p(a_n)$ in 1.44.

SOME RIGHTS RESERVED

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\leadsto$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.63.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.64.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists x.p(x))$.
- **Definition 1.65.** Let $\mathcal{T}$ be a tableau for A, and a positive occurrence of $\forall x.\mathrm{B}$ in A, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.\mathrm{B}$ its multiplicity.
- **Observation 1.66.** *Given a prescribed multiplicity for each positive $\forall$, saturation with $\mathcal{T}_1^f$ terminates.*
- *Proof sketch:* All $\mathcal{T}_1^f$ rules reduce the number of connectives and negative $\forall$ or the multiplicity of positive $\forall$.
- **Theorem 1.67.** *$\mathcal{T}_1^f$ is only complete with unbounded multiplicities.*
- *Proof sketch:* Replace $p(a) \vee p(b)$ with $p(a_1) \vee \ldots \vee p(a_n)$ in 1.44.
- **Remark:** Otherwise validity in PL^1 would be decidable.
- **Implementation:** We need an iterative multiplicity deepening process.

SOME RIGHTS RESERVED

Treating $\mathcal{T}_1^f\bot$

- **Recall:** The $\mathcal{T}_1^f\bot$ rule instantiates the whole tableau.
- **Problem:** There may be more than one $\mathcal{T}_1^f\bot$ opportunity on a branch.
- **Example 1.68.** Choosing which matters – this tableau does not close!

$$\begin{array}{c|c}
\multicolumn{2}{c}{(\exists x.(p(a) \wedge p(b) \Rightarrow p(x)) \wedge (q(b) \Rightarrow q(x)))^{\mathsf{F}}} \\
\multicolumn{2}{c}{((p(a) \wedge p(b) \Rightarrow p(y)) \wedge (q(b) \Rightarrow q(y)))^{\mathsf{F}}} \\
(p(a) \wedge p(b) \Rightarrow p(y))^{\mathsf{F}} & (q(b) \Rightarrow q(y))^{\mathsf{F}} \\
p(a)^{\mathsf{T}} & q(b)^{\mathsf{T}} \\
p(b)^{\mathsf{T}} & q(y)^{\mathsf{F}} \\
p(y)^{\mathsf{F}} & \\
\bot : [a/y] &
\end{array}$$

choosing the other $\mathcal{T}_1^f\bot$ in the left branch allows closure.

- **Idea:** Two ways of systematic proof search in $\mathcal{T}_1^f$:
  - backtracking search over $\mathcal{T}_1^f\bot$ opportunities
  - saturate without $\mathcal{T}_1^f\bot$ and find spanning matings (next slide)

Spanning Matings for $\mathcal{T}_1^f\bot$

- **Observation 1.69.** *$\mathcal{T}_1^f$ without $\mathcal{T}_1^f\bot$ is terminating and confluent for given multiplicities.*
- **Idea:** Saturate without $\mathcal{T}_1^f\bot$ and treat all cuts at the same time (later).
- **Definition 1.70.**
  Let $\mathcal{T}$ be a $\mathcal{T}_1^f$ tableau, then we call a unification problem $\mathcal{E} := \mathrm{A}_1 =^? \mathrm{B}_1 \wedge \ldots \wedge \mathrm{A}_n =^? \mathrm{B}_n$ a mating for $\mathcal{T}$, iff ${\mathrm{A}_i}^{\mathsf{T}}$ and ${\mathrm{B}_i}^{\mathsf{F}}$ occur in the same branch in $\mathcal{T}$.
  We say that $\mathcal{E}$ is a spanning mating, if $\mathcal{E}$ is unifiable and every branch $\mathcal{B}$ of $\mathcal{T}$ contains ${\mathrm{A}_i}^{\mathsf{T}}$ and ${\mathrm{B}_i}^{\mathsf{F}}$ for some i.
- **Theorem 1.71.** *A $\mathcal{T}_1^f$-tableau with a spanning mating induces a closed $\mathcal{T}_1$ tableau.*
- *Proof sketch:* Just apply the unifier of the spanning mating.
- **Idea:** Existence is sufficient, we do not need to compute the unifier.
- **Implementation:** Saturate without $\mathcal{T}_1^f\bot$, backtracking search for spanning matings with $\mathcal{DU}$, adding pairs incrementally.

SOME RIGHTS RESERVED

15.2 First-Order Resolution

SOME RIGHTS RESERVED

First-Order Resolution (and CNF)

- **Definition 2.1.** The first-order CNF calculus $\mathcal{CNF}_1$ is given by the inference rules of $\mathcal{CNF}_0$ extended by the following quantifier rules:

$$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \vee \mathbf{C} \quad Z \notin (\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathbf{C}))}{([Z/X](\mathbf{A}))^{\mathsf{T}} \vee \mathbf{C}}$$

$$\frac{(\forall X.\mathbf{A})^{\mathsf{F}} \vee \mathbf{C} \quad \{X_1, \ldots, X_k\} = \mathrm{free}(\forall X.\mathbf{A}) \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X_1, \ldots, X_k)/X](\mathbf{A}))^{\mathsf{F}} \vee \mathbf{C}}$$

  the first-order CNF $\mathcal{CNF}_1(\Phi)$ of Φ is the set of all clauses that can be derived from Φ.

- **Definition 2.2 (First-Order Resolution Calculus).** The First-order resolution calculus ($\mathcal{R}_1$) is a test calculus that manipulates formulae in conjunctive normal form. $\mathcal{R}_1$ has two inference rules:

$$\frac{\mathbf{A}^{\mathsf{T}} \vee \mathbf{C} \quad \mathbf{B}^{\mathsf{F}} \vee \mathbf{D} \quad \sigma = \mathrm{mgu}(\mathbf{A}, \mathbf{B})}{(\sigma(\mathbf{C})) \vee (\sigma(\mathbf{D}))} \qquad \frac{\mathbf{A}^{\alpha} \vee \mathbf{B}^{\alpha} \vee \mathbf{C} \quad \sigma = \mathrm{mgu}(\mathbf{A}, \mathbf{B})}{(\sigma(\mathbf{A})) \vee (\sigma(\mathbf{C}))}$$

SOME RIGHTS RESERVED

First-Order CNF – Derived Rules

- **Definition 2.3.** The following inference rules are derivable from the ones above via $(\exists X.A) = \neg(\forall X.\neg A)$:

$$\frac{(\exists X.A)^{T} \vee C \quad \{X_1, \ldots, X_k\} = \mathrm{free}(\forall X.A) \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X_1, \ldots, X_k)/X](A))^{T} \vee C}$$

$$\frac{(\exists X.A)^{F} \vee C \quad Z \notin (\mathrm{free}(A) \cup \mathrm{free}(C))}{([Z/X](A))^{F} \vee C}$$

SOME RIGHTS RESERVED

15.2.1 Resolution Examples

SOME RIGHTS RESERVED

Col. West, *a Criminal?*

- **Example 2.4.** From [**RusNor:AIMA09**]

  The law says it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

  Prove that Col. West is a criminal.
- **Remark:** Modern resolution theorem provers prove this in less than 50ms.
- **Problem:** That is only true, if we only give the theorem prover exactly the right laws and background knowledge. If we give it all of them, it drowns in the combinatorial explosion.
- Let us build a resolution proof for the claim above.
- **But first** we must translate the situation into first-order logic clauses.
- **Convention:** In what follows, for better readability we will sometimes write implications $P \wedge Q \wedge R \Rightarrow S$ instead of clauses $P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{T}}$.

SOME RIGHTS RESERVED

Col. West, *a Criminal?*

- "*It is a crime for an American to sell weapons to hostile nations*":
  Clause: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$

Col. West, *a Criminal?*

- "*It is a crime for an American to sell weapons to hostile nations*":
  Clause: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$
- "*Nono has some missiles*": $\exists X.\mathrm{own}(\mathrm{NN}, X) \wedge \mathrm{mle}(X)$
  Clauses: $\mathrm{own}(\mathrm{NN}, c)^{\mathsf{T}}$ and $\mathrm{mle}(c)$ (c is Skolem constant)

SOME RIGHTS RESERVED

Col. West, *a Criminal?*

- "*It is a crime for an American to sell weapons to hostile nations*":
  Clause: $\mathrm{ami}(X_1) \land \mathrm{weap}(Y_1) \land \mathrm{sell}(X_1, Y_1, Z_1) \land \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$
- "*Nono has some missiles*": $\exists X.\mathrm{own}(\mathrm{NN}, X) \land \mathrm{mle}(X)$
  Clauses: $\mathrm{own}(\mathrm{NN}, c)^{\mathsf{T}}$ and $\mathrm{mle}(c)$ (c is Skolem constant)
- "*All of Nono's missiles were sold to it by Colonel West.*"
  Clause: $\mathrm{mle}(X_2) \land \mathrm{own}(\mathrm{NN}, X_2) \Rightarrow \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NN})$

SOME RIGHTS RESERVED

Col. West, *a Criminal?*

- "*It is a crime for an American to sell weapons to hostile nations*":
  Clause: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$
- "*Nono has some missiles*": $\exists X.\mathrm{own}(\mathrm{NN}, X) \wedge \mathrm{mle}(X)$
  Clauses: $\mathrm{own}(\mathrm{NN}, c)^T$ and $\mathrm{mle}(c)$ (c is Skolem constant)
- "*All of Nono's missiles were sold to it by Colonel West.*"
  Clause: $\mathrm{mle}(X_2) \wedge \mathrm{own}(\mathrm{NN}, X_2) \Rightarrow \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NN})$
- "*Missiles are weapons*":
  Clause: $\mathrm{mle}(X_3) \Rightarrow \mathrm{weap}(X_3)$

SOME RIGHTS RESERVED

Col. West, *a Criminal?*

- "*It is a crime for an American to sell weapons to hostile nations*":
  Clause: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$
- "*Nono has some missiles*": $\exists X.\mathrm{own}(\mathrm{NN}, X) \wedge \mathrm{mle}(X)$
  Clauses: $\mathrm{own}(\mathrm{NN}, c)^T$ and $\mathrm{mle}(c)$ (c is Skolem constant)
- "*All of Nono's missiles were sold to it by Colonel West.*"
  Clause: $\mathrm{mle}(X_2) \wedge \mathrm{own}(\mathrm{NN}, X_2) \Rightarrow \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NN})$
- "*Missiles are weapons*":
  Clause: $\mathrm{mle}(X_3) \Rightarrow \mathrm{weap}(X_3)$
- "*An enemy of America counts as "hostile"*":
  Clause: $\mathrm{enmy}(X_4, \mathrm{USA}) \Rightarrow \mathrm{host}(X_4)$

SOME RIGHTS RESERVED

Col. West, *a Criminal?*

- "*It is a crime for an American to sell weapons to hostile nations*":
  Clause: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$
- "*Nono has some missiles*": $\exists X.\mathrm{own}(\mathrm{NN}, X) \wedge \mathrm{mle}(X)$
  Clauses: $\mathrm{own}(\mathrm{NN}, c)^{\mathsf{T}}$ and $\mathrm{mle}(c)$ (c is Skolem constant)
- "*All of Nono's missiles were sold to it by Colonel West.*"
  Clause: $\mathrm{mle}(X_2) \wedge \mathrm{own}(\mathrm{NN}, X_2) \Rightarrow \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NN})$
- "*Missiles are weapons*":
  Clause: $\mathrm{mle}(X_3) \Rightarrow \mathrm{weap}(X_3)$
- "*An enemy of America counts as "hostile"*":
  Clause: $\mathrm{enmy}(X_4, \mathrm{USA}) \Rightarrow \mathrm{host}(X_4)$
- "*West is an American.*"
  Clause: $\mathrm{ami}(\mathrm{West})$

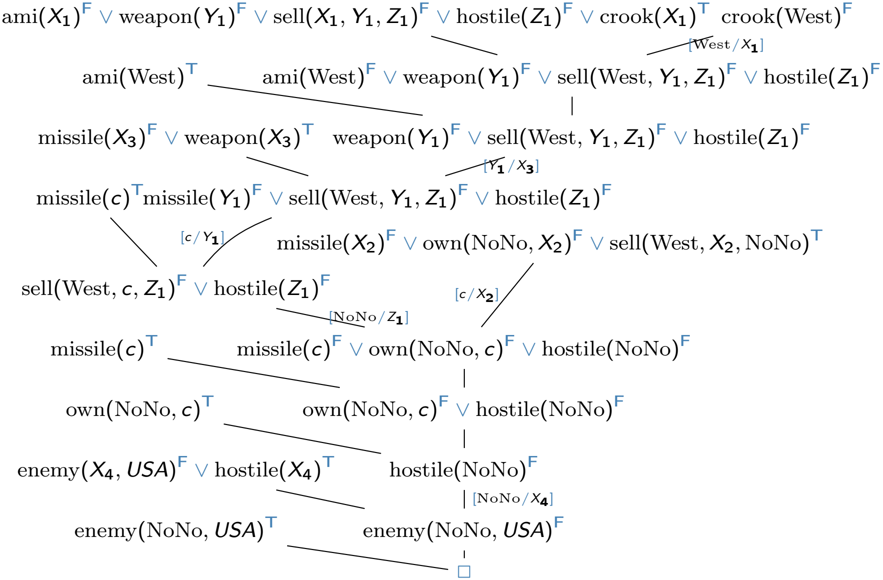
SOME RIGHTS RESERVED

Col. West, *a Criminal?*

- "*It is a crime for an American to sell weapons to hostile nations*":
  Clause: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$
- "*Nono has some missiles*": $\exists X.\mathrm{own}(\mathrm{NN}, X) \wedge \mathrm{mle}(X)$
  Clauses: $\mathrm{own}(\mathrm{NN}, c)^{\mathsf{T}}$ and $\mathrm{mle}(c)$ (c is Skolem constant)
- "*All of Nono's missiles were sold to it by Colonel West.*"
  Clause: $\mathrm{mle}(X_2) \wedge \mathrm{own}(\mathrm{NN}, X_2) \Rightarrow \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NN})$
- "*Missiles are weapons*":
  Clause: $\mathrm{mle}(X_3) \Rightarrow \mathrm{weap}(X_3)$
- "*An enemy of America counts as "hostile"*":
  Clause: $\mathrm{enmy}(X_4, \mathrm{USA}) \Rightarrow \mathrm{host}(X_4)$
- "*West is an American.*"
  Clause: $\mathrm{ami}(\mathrm{West})$
- "*The country Nono is an enemy of America*":
  $\mathrm{enmy}(\mathrm{NN}, \mathrm{USA})$

SOME RIGHTS RESERVED

Col. West, *a Criminal!* PL1 Resolution Proof



$\mathrm{ami}(X_1)^F \vee \mathrm{weapon}(Y_1)^F \vee \mathrm{sell}(X_1, Y_1, Z_1)^F \vee \mathrm{hostile}(Z_1)^F \vee \mathrm{crook}(X_1)^T \quad \mathrm{crook}(\mathrm{West})^F$

$[\mathrm{West}/X_1]$

$\mathrm{ami}(\mathrm{West})^T \quad \mathrm{ami}(\mathrm{West})^F \vee \mathrm{weapon}(Y_1)^F \vee \mathrm{sell}(\mathrm{West}, Y_1, Z_1)^F \vee \mathrm{hostile}(Z_1)^F$

$\mathrm{missile}(X_3)^F \vee \mathrm{weapon}(X_3)^T \quad \mathrm{weapon}(Y_1)^F \vee \mathrm{sell}(\mathrm{West}, Y_1, Z_1)^F \vee \mathrm{hostile}(Z_1)^F$

$[Y_1/X_3]$

$\mathrm{missile}(c)^T \quad \mathrm{missile}(Y_1)^F \vee \mathrm{sell}(\mathrm{West}, Y_1, Z_1)^F \vee \mathrm{hostile}(Z_1)^F$

$[c/Y_1]$

$\mathrm{missile}(X_2)^F \vee \mathrm{own}(\mathrm{NoNo}, X_2)^F \vee \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NoNo})^T$

$\mathrm{sell}(\mathrm{West}, c, Z_1)^F \vee \mathrm{hostile}(Z_1)^F$

$[c/X_2]$

$[\mathrm{NoNo}/Z_1]$

$\mathrm{missile}(c)^T \quad \mathrm{missile}(c)^F \vee \mathrm{own}(\mathrm{NoNo}, c)^F \vee \mathrm{hostile}(\mathrm{NoNo})^F$

$\mathrm{own}(\mathrm{NoNo}, c)^T \quad \mathrm{own}(\mathrm{NoNo}, c)^F \vee \mathrm{hostile}(\mathrm{NoNo})^F$

$\mathrm{enemy}(X_4, USA)^F \vee \mathrm{hostile}(X_4)^T \quad \mathrm{hostile}(\mathrm{NoNo})^F$

$[\mathrm{NoNo}/X_4]$

$\mathrm{enemy}(\mathrm{NoNo}, USA)^T \quad \mathrm{enemy}(\mathrm{NoNo}, USA)^F$

$\square$

SOME RIGHTS RESERVED

Curiosity Killed the Cat?

- **Example 2.5.** From [**RusNor:AIMA09**]

  Everyone who loves all animals is loved by someone.
  Anyone who kills an animal is loved by noone.
  Jack loves all animals.
  Cats are animals.
  Either Jack or curiosity killed the cat (whose name is “Garfield”).

  Prove that curiosity killed the cat.

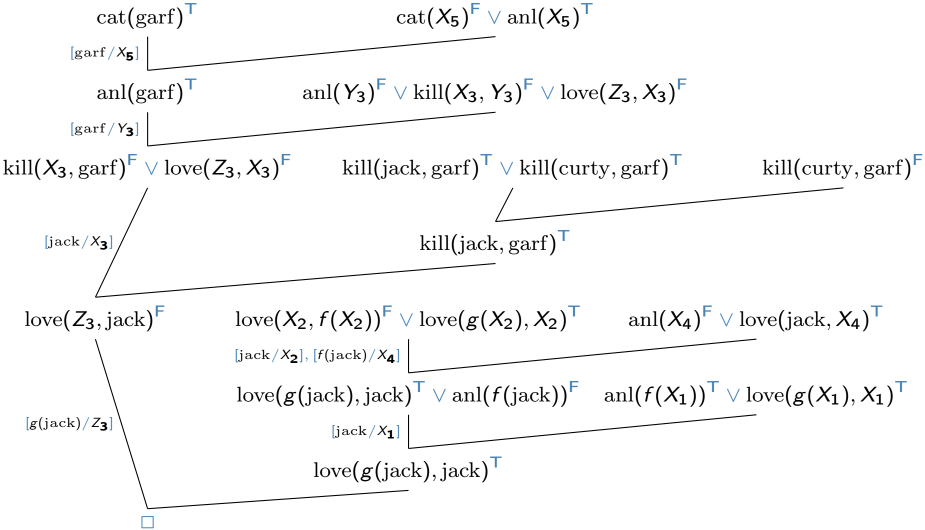
SOME RIGHTS RESERVED

Curiosity Killed the Cat? Clauses

- "*Everyone who loves all animals is loved by someone*":
  $\forall X.(\forall Y.\mathrm{animal}(Y) \Rightarrow \mathrm{love}(X,Y)) \Rightarrow (\exists Z.\mathrm{love}(Z,X))$
  Clauses: $\mathrm{animal}(g(X_1))^{T} \vee \mathrm{love}(g(X_1),X_1)^{T}$ and $\mathrm{love}(X_2,f(X_2))^{F} \vee \mathrm{love}(g(X_2),X_2)^{T}$
- "*Anyone who kills an animal is loved by noone*":
  $\forall X.\exists Y.\mathrm{animal}(Y) \wedge \mathrm{kill}(X,Y) \Rightarrow (\forall Z.\neg\mathrm{love}(Z,X))$
  Clause: $\mathrm{animal}(Y_3)^{F} \vee \mathrm{kill}(X_3,Y_3)^{F} \vee \mathrm{love}(Z_3,X_3)^{F}$
- "*Jack loves all animals*":
  Clause: $\mathrm{animal}(X_4)^{F} \vee \mathrm{love}(\mathrm{jack},X_4)^{T}$
- "*Cats are animals*":
  Clause: $\mathrm{cat}(X_5)^{F} \vee \mathrm{animal}(X_5)^{T}$
- "*Either Jack or curiosity killed the cat (whose name is "Garfield")*":
  Clauses: $\mathrm{kill}(\mathrm{jack},\mathrm{garf})^{T} \vee \mathrm{kill}(\mathrm{curiosity},\mathrm{garf})^{T}$ and $\mathrm{cat}(\mathrm{garf})^{T}$

SOME RIGHTS RESERVED

Curiosity Killed the Cat! PL1 Resolution Proof



$\text{cat}(\text{garf})^{T}$

$\text{cat}(X_5)^{F} \vee \text{anl}(X_5)^{T}$

$[\text{garf}/X_5]$

$\text{anl}(\text{garf})^{T}$

$\text{anl}(Y_3)^{F} \vee \text{kill}(X_3, Y_3)^{F} \vee \text{love}(Z_3, X_3)^{F}$

$[\text{garf}/Y_3]$

$\text{kill}(X_3, \text{garf})^{F} \vee \text{love}(Z_3, X_3)^{F}$

$\text{kill}(\text{jack}, \text{garf})^{T} \vee \text{kill}(\text{curty}, \text{garf})^{T}$

$\text{kill}(\text{curty}, \text{garf})^{F}$

$[\text{jack}/X_3]$

$\text{kill}(\text{jack}, \text{garf})^{T}$

$\text{love}(Z_3, \text{jack})^{F}$

$\text{love}(X_2, f(X_2))^{F} \vee \text{love}(g(X_2), X_2)^{T}$

$\text{anl}(X_4)^{F} \vee \text{love}(\text{jack}, X_4)^{T}$

$[\text{jack}/X_2], [f(\text{jack})/X_4]$

$\text{love}(g(\text{jack}), \text{jack})^{T} \vee \text{anl}(f(\text{jack}))^{F}$

$\text{anl}(f(X_1))^{T} \vee \text{love}(g(X_1), X_1)^{T}$

$[g(\text{jack})/Z_3]$

$[\text{jack}/X_1]$

$\text{love}(g(\text{jack}), \text{jack})^{T}$

$\square$

SOME RIGHTS RESERVED

15.3 Logic Programming as Resolution Theorem Proving

SOME RIGHTS RESERVED

We know all this already

- Goals, goal sets, rules, and facts are just clauses. (called Horn clauses)
- **Observation 3.1 (Rule).** *H:$-B_1$,…,B_n. corresponds to $H^T \vee B_1{}^F \vee \ldots \vee B_n{}^F$ (head the only positive literal)*
- **Observation 3.2 (Goal set).** ?$-$ G_1,…,G_n. *corresponds to $G_1{}^F \vee \ldots \vee G_n{}^F$*
- **Observation 3.3 (Fact).** *F. corresponds to the unit clause F^T.*
- **Definition 3.4.** A Horn clause is a clause with at most one positive literal.
- **Recall:** Backchaining as search:
  - state = tuple of goals; goal state = empty list (of goals).
  - $next(\langle G, R_1, \ldots, R_l\rangle) := \langle\sigma(B_1), \ldots, \sigma(B_m), \sigma(R_1), \ldots, \sigma(R_l)\rangle$ if there is a rule H:$-B_1$,…, B_m. and a substitution σ with $\sigma(H) = \sigma(G)$.
- **Note:** Backchaining becomes resolution

$$\frac{P^T \vee A \quad P^F \vee B}{A \vee B}$$

positive, unit-resulting hyperresolution (PURR)

SOME RIGHTS RESERVED

PROLOG (Horn Logic)

- **Definition 3.5.** A clause is called a Horn clause, iff contains at most one positive literal, i.e. if it is of the form $B_1{}^{\mathsf{F}} \vee \ldots \vee B_n{}^{\mathsf{F}} \vee A^{\mathsf{T}}$ – i.e. A:−B_1,. . .,B_n. in Prolog notation.
  - Rule clause: general case, e.g. fallible(X) : human(X).
  - Fact clause: no negative literals, e.g. human(sokrates).
  - Program: set of rule and fact clauses.
  - Query: no positive literals: e.g. ?− fallible(X),greek(X).
- **Definition 3.6.** Horn logic is the formal system whose language is the set of Horn clauses together with the calculus $\mathcal{H}$ given by MP, $\wedge I$, and Subst.
- **Definition 3.7.** A logic program P entails a query Q with answer substitution σ, iff there is a $\mathcal{H}$ derivation D of Q from P and σ is the combined substitution of the Subst instances in D.

SOME RIGHTS RESERVED

PROLOG: Our Example

- **Program:**

```
human(leibniz).
human(sokrates).
greek(sokrates).
fallible(X):−human(X).
```

- **Example 3.8 (Query).** ?− fallible(X),greek(X).
- **Answer substitution:** $[\mathrm{sokrates}/X]$

SOME RIGHTS RESERVED

Knowledge Base (Example)

- **Example 3.9.** car(c). is in the knowlege base generated by

```
has_motor(c).
has_wheels(c,4).
car(X):— has_motor(X),has_wheels(X,4).
```

$$\dfrac{\dfrac{m(c) \quad w(c,4)}{m(c) \wedge w(c,4)}\wedge I \qquad \dfrac{m(x) \wedge w(x,4) \Rightarrow car(x)}{m(c) \wedge w(c,4) \Rightarrow car(c)}\text{Subst}}{car(c)}\text{MP}$$

SOME RIGHTS RESERVED

Why Only Horn Clauses?

- General clauses of the form A1,. . .,An : B1,. . .,Bn.
- e.g. greek(sokrates),greek(perikles)
  - **Question**: Are there fallible greeks?
  - **Indefinite answer**: Yes, Perikles or Sokrates
  - Warning: how about Sokrates and Perikles?
- e.g. greek(sokrates),roman(sokrates):−.
  - **Query**: Are there fallible greeks?
  - **Answer**: Yes, Sokrates, if he is not a roman
  - Is this abduction?????

SOME RIGHTS RESERVED

Three Principal Modes of Inference

- **Definition 3.10.** Deduction $\widehat{=}$ knowledge extension
- **Example 3.11.** $\frac{\mathit{rains} \Rightarrow \mathit{wet\_street} \quad \mathit{rains}}{\mathit{wet\_street}} D$

Three Principal Modes of Inference

- **Definition 3.16.** Deduction $\widehat{=}$ knowledge extension
- **Example 3.17.** $\frac{rains \Rightarrow wet\_street \quad rains}{wet\_street}\,D$
- **Definition 3.18.** Abduction $\widehat{=}$ explanation
- **Example 3.19.** $\frac{rains \Rightarrow wet\_street \quad wet\_street}{rains}\,A$

SOME RIGHTS RESERVED

Three Principal Modes of Inference

- **Definition 3.22.** Deduction $\widehat{=}$ knowledge extension
- **Example 3.23.** $\frac{rains \Rightarrow wet\_street \quad rains}{wet\_street} D$
- **Definition 3.24.** Abduction $\widehat{=}$ explanation
- **Example 3.25.** $\frac{rains \Rightarrow wet\_street \quad wet\_street}{rains} A$
- **Definition 3.26.** Induction $\widehat{=}$ learning general rules from examples
- **Example 3.27.** $\frac{wet\_street \quad rains}{rains \Rightarrow wet\_street} I$

SOME RIGHTS RESERVED

15.4 Summary: ATP in First-Order Logic

SOME RIGHTS RESERVED

Summary: ATP in First-Order Logic

- The propositional calculi for ATP can be extended to first-order logic by adding quantifier rules.
  - The rule for the universal quantifier can be made efficient by introducing metavariables that postpone the decision for instances.
  - We have to extend the witness constants in the rules for existential quantifiers to Skolem functions.
  - The cut rules can used to instantiate the metavariables by unification.

  These ideas are enough to build a tableau calculus for first-order logic.
- Unification is an efficient decision procdure for finding substitutions that make first-order terms (syntactically) equal.
- In prenex normal form, all quantifiers are up front. In Skolem normal form, additionally there are no existential quantifiers. In claus normal form, additionally the formula is in CNF.
- Any PL^1 formula can efficiently be brought into a satisfiability-equivalent clause normal form.
- This allows first-order resolution.

SOME RIGHTS RESERVED

Chapter 16
Knowledge Representation and the Semantic Web

SOME RIGHTS RESERVED

16.1 Introduction to Knowledge Representation

SOME RIGHTS RESERVED

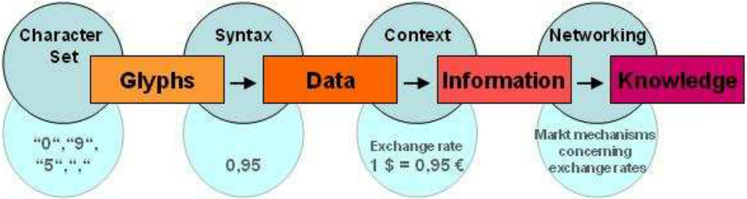
the introduction to knowledge representation]27279

16.1.1 Knowledge & Representation

SOME RIGHTS RESERVED

What is knowledge? Why Representation?

- Lots/all of (academic) disciplines deal with knowledge!
- According to Probst/Raub/Romhardt [**ProbstRaubRomhardt**]



- **For the purposes of this course:** Knowledge is the information necessary to support intelligent reasoning!

representation	can be used to determine
set of words	whether a word is admissible
list of words	the rank of a word
a lexicon	translation and/or grammatical function
structure	function

SOME RIGHTS RESERVED

Knowledge Representation vs. Data Structures

- **Idea:** Representation as structure and function.
  - the representation determines the content theory (what is the data?)
  - the function determines the process model (what do we do with the data?)
- **Question:** Why do we use the term "knowledge representation" rather than
  - data structures? (sets, lists, ... above)
  - information representation? (it is information)
- **Answer:** No good reason other than AI practice, with the intuition that
  - data is simple and general (supports many algorithms)
  - knowledge is complex (has distinguished process model)

SOME RIGHTS RESERVED

Some Paradigms for Knowledge Representation in AI/NLP

- GOFAI (good old-fashioned AI)
  - symbolic knowledge representation, process model based on heuristic search
- Statistical, corpus-based approaches.
  - symbolic representation, process model based on machine learning
  - knowledge is divided into symbolic- and statistical (search) knowledge
- The connectionist approach
  - sub-symbolic representation, process model based on primitive processing elements (nodes) and weighted links
  - knowledge is only present in activation patters, etc.

SOME RIGHTS RESERVED

KR Approaches/Evaluation Criteria

- **Definition 1.1.** The evaluation criteria for knowledge representation approaches are:
  - Expressive adequacy: What can be represented, what distinctions are supported.
  - Reasoning efficiency: Can the representation support processing that generates results in acceptable speed?
  - Primitives: What are the primitive elements of representation, are they intuitive, cognitively adequate?
  - Meta representation: Knowledge about knowledge
  - Completeness: The problems of reasoning with knowledge that is known to be incomplete.

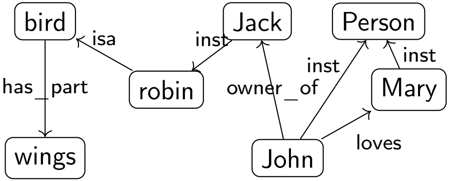
SOME RIGHTS RESERVED

16.1.2 Semantic Networks

SOME RIGHTS RESERVED

Semantic Networks [ColQui:rtsm69]

- **Definition 1.2.** A semantic network is a directed graph for representing knowledge:
  - nodes represent objects and concepts (classes of objects) (e.g. John (object) and bird (concept))
  - edges (called links) represent relations between these (isa, father_of, belongs_to)
- **Example 1.3.** A semantic network for birds and persons:

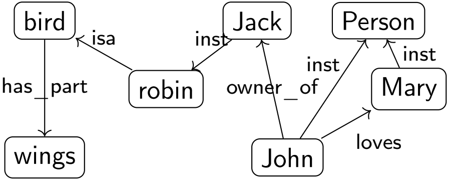


- **Problem:** How do we derive new information from such a network?
- **Idea:** Encode taxonomic information about objects and concepts in special links (“isa” and “inst”) and specify property inheritance along them in the process model.

SOME RIGHTS RESERVED

Deriving Knowledge Implicit in Semantic Networks

- **Observation 1.4.** *There is more knowledge in a semantic network than is explicitly written down.*
- **Example 1.5.** In the network below, we "know" that "*robins have wings*" and in particular, "*Jack has wings*".



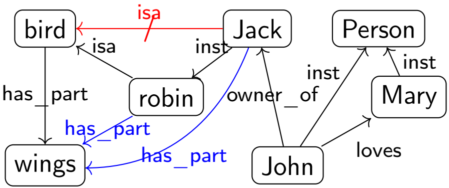
- **Idea:** Links labeled with "isa" and "inst" are special: they propagate properties encoded by other links.
- **Definition 1.6.** We call links labeled by
  - "isa" an inclusion or isa link (inclusion of concepts)
  - "inst" instance or inst link (concept membership)

SOME RIGHTS RESERVED

Deriving Knowledge Semantic Networks

- **Definition 1.7 (Inference in Semantic Networks).** We call all link labels except "inst" and "isa" in a semantic network relations.

  Let N be a semantic network and R a relation in N such that $A \xrightarrow{\text{isa}} B \xrightarrow{R} C$ or $A \xrightarrow{\text{inst}} B \xrightarrow{R} C$, then we can derive a relation $A \xrightarrow{R} C$ in N.
  The process of deriving new concepts and relations from existing ones is called inference and concepts/relations that are only available via inference implicit (in a semantic network).
- **Intuition:** Derived relations represent knowledge that is implicit in the network; they could be added, but usually are not to avoid clutter.
- **Example 1.8.** Derived relations in 1.5

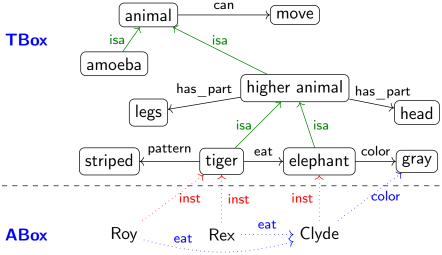


- **Slogan:** Get out more knowledge from a semantic networks than you put in.

SOME RIGHTS RESERVED

Terminologies and Assertions

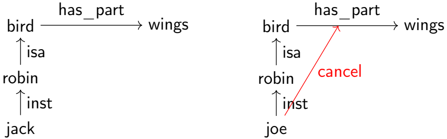
- *Remark 1.9.* We should distinguish concepts from objects.
- **Definition 1.10.** We call the subgraph of a semantic network N spanned by the isa links and relations between concepts the terminology (or TBox, or the famous Isa Hierarchy) and the subgraph spanned by the inst links and relations between objects, the assertions (together the ABox) of N.
- **Example 1.11.** In this semantic network we keep objects concept apart notationally:



In particular we have objects “Rex”, “Roy”, and “Clyde”, which have (derived) relations (e.g. “*Clyde*” is “*gray*”).

Limitations of Semantic Networks

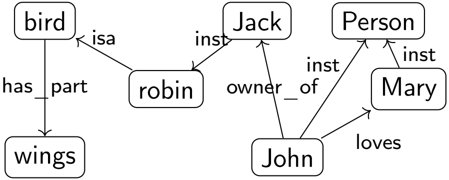
- What is the meaning of a link?
  - link labels are very suggestive (misleading for humans)
  - meaning of link types defined in the process model (no denotational semantics)
- **Problem:** No distinction of optional and defining traits!
- **Example 1.12.** Consider a robin that has lost its wings in an accident:



"Cancel-links" have been proposed, but their status and process model are debatable.

Another Notation for Semantic Networks

- **Definition 1.13.** Function/argument notation for semantic networks
  - interprets nodes as arguments (reification to individuals)
  - interprets links as functions (predicates actually)
- **Example 1.14.**



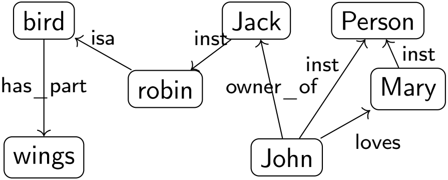
isa(robin,bird)
haspart(bird,wings)
inst(Jack,robin)
owner_of(John, robin)
loves(John,Mary)

- **Evaluation:**

+ linear notation (equivalent, but better to implement on a computer)
+ easy to give process model by deduction (e.g. in Prolog)
− worse locality properties (networks are associative)

A Denotational Semantics for Semantic Networks

- **Observation:** If we handle isa and inst links specially in function/argument notation



robin $\subseteq$ bird
haspart(bird,wings)
Jack $\in$ robin
owner_of(John, Jack)
loves(John,Mary)

it looks like first-order logic, if we take

  - $a \in S$ to mean $S(a)$ for an object a and a concept S.
  - $A \subseteq B$ to mean $\forall X.A(X) \Rightarrow B(X)$ and concepts A and B
  - $R(A, B)$ to mean $\forall X.A(X) \Rightarrow (\exists Y.B(Y) \wedge R(X, Y))$ for a relation R.
- **Idea:** Take first-order deduction as process model (gives inheritance for free)

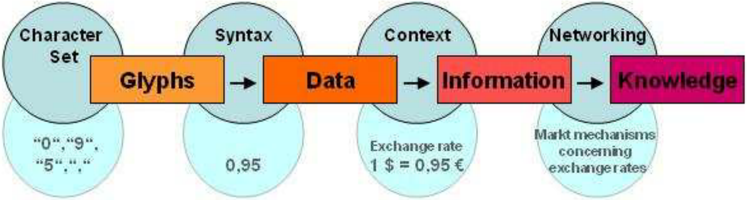
SOME RIGHTS RESERVED

16.1.3 The Semantic Web

SOME RIGHTS RESERVED

The Semantic Web

- **Definition 1.15.** The semantic web is the result including of semantic content in web pages with the aim of converting the WWW into a machine-understandable "web of data", where inference based services can add value to the ecosystem.
- **Idea:** Move web content up the ladder, use inference to make connections.



- **Example 1.16.** Information not explicitly represented (in one place)

  Query: "*Who was US president when Barak Obama was born?*"
  Google: "*...BIRTH DATE: August 04, 1961...*"
  Query: "*Who was US president in 1961?*"
  Google: "*President: Dwight D. Eisenhower [...] John F. Kennedy (starting Jan. 20.)*"

  Humans understand the text and combine the information to get the answer. Machines need more than just text ⇝ semantic web technology.

SOME RIGHTS RESERVED

What is the Information a User sees?

- **Example 1.17.** Take the following web-site with a conference announcement

  WWW2002
  The eleventh International World Wide Web Conference
  Sheraton Waikiki Hotel
  Honolulu, Hawaii, USA
  7-11 May 2002
  Registered participants coming from
  Australia, Canada, Chile Denmark, France, Germany, Ghana, Hong Kong, India, Ireland, Italy, Japan, Malta, New Zealand, The Netherlands, Norway, Singapore, Switzerland, the United Kingdom, the United States, Vietnam, Zaire

  On the 7th May Honolulu will provide the backdrop of the eleventh International World Wide Web Conference.

  Speakers confirmed
  Tim Berners-Lee: Tim is the well known inventor of the Web,
  Ian Foster: Ian is the pioneer of the Grid, the next generation internet.

SOME RIGHTS RESERVED

What the machine sees

- **Example 1.18.** Here is what the machine “sees” from the conference announcement:

  𝒲𝒲𝒲∈′′∈

  𝒯⟨⌉⌉↕⌉⊑⌉\⊔⟨ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉

  𝒮⟨⌉∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔⌉↕

  ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜

  ↖∞∞ℳ⊣†∈′′∈

  ℛ⌉}⟩∫⊔⌉∇⌉⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫⌋≀⇕⟩\}{∇≀⇕

  𝒜⊓∫⊔∇⊣↕⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩↕⌉𝒟⌉\⇕⊣∇∥⇔ℱ∇⊣\⌋⌉⇔𝒢⌉∇⇕⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔

  ℐ∇⌉↕⊣\⌈⇔ℐ⊔⊣↕†⇔𝒥⊣√⊣\⇔ℳ⊣↕⊔⊣⇔𝒩⌉⊒𝒵⌉⊣↕⊣\⌈⇔𝒯⟨⌉𝒩⌉⊔⟨⌉∇↕⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔

  𝒮⟩\}⊣√≀∇⌉⇔𝒮⊒⟩⊔‡⌉∇↕⊣\⌈⇔⊔⟨⌉𝒰\⟩⊔⌉⌈𝒦⟩\}⌈≀⇕⇔⊔⟨⌉𝒰\⟩⊔⌉⌈𝒮⊔⊣⊔⌉∫⇔𝒱⟩⌉⊔\⊣⇕⇔𝒵⊣⟩∇⌉

  𝒪\⊔⟨⌉⊔⟨ℳ⊣†ℋ≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈⌉⊔⟨⌉⌊⊣⌋∥⌈∇≀√≀{⊔⟨⌉⌉↕⌉⊑⌉\⊔⟨

  ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉↙

  𝒮√⌉⊣∥⌉∇∫⌋≀\{⟩∇⇕⌉⌈

  𝒯⟩⇕ℬ⌉∇\⌉∇∫↖ℒ⌉⌉¬𝒯⟩⇕⟩∫⊔⟨⌉⊒⌉↕↕∥\≀⊒\⟩\⊑⌉\⊔≀∇≀{⊔⟨⌉𝒲⌉⌊⇔

  ℐ⊣\ℱ≀∫⊔⌉∇¬ℐ⊣\⟩∫⊔⟨⌉√⟩≀\⌉⌉∇≀{⊔⟨⌉𝒢∇⟩⌈⇔⊔⟨⌉\⌉§⊔}⌉\⌉∇⊣⊔⟩≀\⟩\⊔⌉∇\⌉⊔↙

Solution: XML markup with “meaningful” Tags

- **Example 1.19.** Let’s annotate (parts of) the meaning via XML markup

<title>𝒲𝒲𝒲∈′′∈
𝒯⟨]]↕]⊑]\⊔⟨ℐ\⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋]</title>
<place>𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜</place>
<date>↖∞∞ℳ⊣†∈′′∈</date>
<participants>ℛ]}⟩∫⊔]∇]⌈√⊣∇⊔⟩]⟩√⊣\⊔∫]≀↕⟩\}{∇≀↕
𝒜⊓∫⊔∇⊣↕⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩↕]𝒟]\↕⊣∇∥⇔ℱ∇⊣\⌋]⇔𝒢]∇↕⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔
ℐ∇]↕⊣\⌈⇔ℐ⊔⊣↕†⇔𝒥⊣√⊣\⇔ℳ⊣↕⊔⊣⇔𝒩]⊒𝒵]⊣↕⊣\⌈⇔𝒯⟨]𝒩]⊔⟨]∇↕⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔
𝒮⟩\}⊣√≀∇]⇔𝒮⊒⟩⊔‡]∇↕⊣\⌈⇔⊔⟨]𝒰\⟩⊔]⌈𝒦⟩\}⌈≀↕⇔⊔⟨]𝒰\⟩⊔]⌈𝒮⊔⊣⊔]∫⇔𝒱⟩]⊔\⊣↕⇔𝒵⊣⟩∇]
</participants>
<introduction>𝒪\⊔⟨]⊔⟨ℳ⊣†ℋ≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈]⊔⟨]⌊⊣]∥⌈∇≀√≀{⊔⟨]]↕]⊑]\⊔⟨ℐ\↖
⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋]↙</introduction>
<program>𝒮√]⊣∥]∇∫]≀\{⟩∇↕]⌈
<speaker>𝒯⟩↕ℬ]∇\]∇∫↖ℒ]]¬𝒯⟩↕⟩∫⊔⟨]⊒]↕↕∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]𝒲]⌊</speaker>
<speaker>ℐ⊣\ℱ≀∫⊔]∇¬ℐ⊣\⟩∫⊔⟨]√⟩≀\]]∇≀{⊔⟨]𝒢∇⟩⌈⇔⊔⟨]\]§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇↖
\]⊔<speaker>
</program>

What can we do with this?

- **Example 1.20.** Consider the following fragments:

  ℜ⊔⟩⊔↕]⊤𝒲𝒲𝒲∈′′∈
  𝒯⟨]]↕]⊑]\⊔⟨ℐ\⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕[𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋]ℜ∝⊔⟩⊔↕]⊤
  ℜ√↕⊣|]⊤𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜ℜ∝√↕⊣|]⊤
  ℜ⌈⊣⊔]⊤↖∞∞ℳ⊣†∈′′∈ℜ∝⌈⊣⊔]⊤

  Given the markup above, a machine agent can
  - parse ∞∞ℳ⊣†∈′′∈ as the date May 7 11 2002 and add this to the user's calendar,
  - parse 𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜 as a destination and find flights.
- **But:** do not be deceived by your ability to understand English!

SOME RIGHTS RESERVED

What the machine sees of the XML

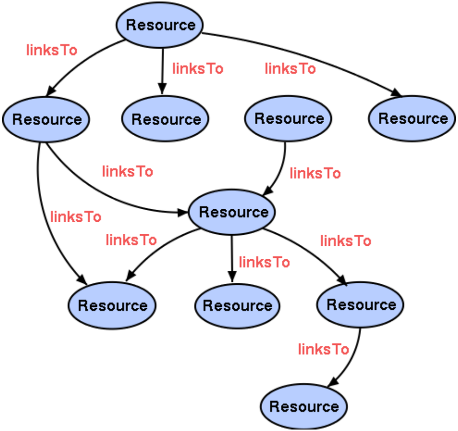
- **Example 1.21.** Here is what the machine sees of the XML

```
<title>WWW∈′′∈
T⟨]]↕]⊑]\⊔⟨I\⊔]∇\⊣⊔⟩≀\⊣↕W≀∇↕⌈W⟩⌈]W]⌊C≀\{]∇]\⌋]</⊔⟩⊔↕]>
<√↕⊣⌋]>S⟨]∇⊣⊔≀\W⊣⟩∥⟩∥⟩H≀⊔]↕H≀\≀↕⊓↕⊓⇔H⊣⊒⊣⟩⟩⇔USA</√↕⊣⌋]>
<⌈⊣⊔]>↖∞∞M⊣†∈′′∈</⌈⊣⊔]>
<√⊣∇⊔⟩⌋⟩√⊣\⊔∫>R]}⟩∫⊔]∇]⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫⌋≀⇕⟩\}{∇≀⇕
A⊓∫⊔∇⊣↕⟩⊣⇔C⊣\⊣⌈⊣⇔C⟨⟩↕]D]\⇕⊣∇∥⇔F∇⊣\⌋]⇔G]∇⇕⊣\†⇔G⟨⊣\⊣⇔H≀\}K≀\}⇔I\⌈⟩⊣⇔
I∇]↕⊣\⌈⇔I⊔⊣↕†⇔J⊣√⊣\⇔M⊣↕⊔⊣⇔N]⊒Z]⊣↕⊣\⌈⇔T⟨]N]⊔⟨]∇↕⊣\⌈∫⇔N≀∇⊒⊣†⇔
S⟩\}⊣√≀∇]⇔S⊒]⌈]\⇔S⊒⟩⊔‡]∇↕⊣\⌈⇔⊔⟨]U\⟩⊔]⌈K⟩\}⌈≀⇕⇔⊔⟨]U\⟩⊔]⌈S⊔⊣⊔]∫⇔V⟩]⊔\⊣⇕⇔Z⊣⇕⌊⟩⊣
</√⊣∇⊔⟩⌋⟩√⊣\⊔∫>
<⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>O\⊔⟨]⊔⟨M⊣†H≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈]⌈⊔⟨]⌊⊣⌋∥⌈∇≀√≀{⊔⟨]]↕]⊑]\⊔⟨I\⊔]∇↖
\⊣⊔⟩≀\⊣↕W≀∇↕⌈W⟩⌈]W]⌊C≀\{]∇]\⌋]↙</⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>
<√∇≀}∇⊣⇕>S√]⊣∥]∇∫⌋≀\{⟩∇⇕]⌈
<∫√]⊣∥]∇>T⟩⇕B]∇\]∇∫↖L]]¬T⟩⇕⟩∫⊔⟨]⊒]↕↕∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]W]⌊</∫√]⊣∥]∇>
<∫√]⊣∥]∇>I⊣\F≀∫⊔]∇¬I⊣\⟩∫⊔⟨]√⟩≀\]]∇≀{⊔⟨]G∇⟩⌈⇔⊔⟨]\]§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇↖
\]⊔</∫√]⊣∥]∇>
</√∇≀}∇⊣⇕>
```

SOME RIGHTS RESERVED

The Current Web

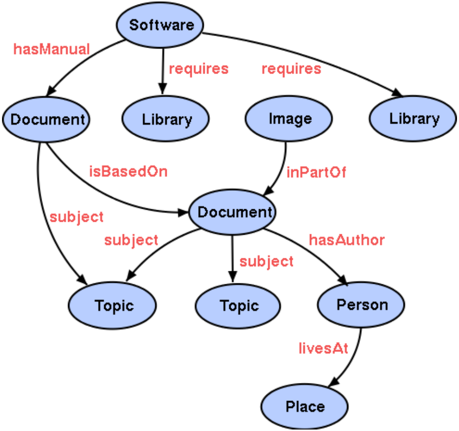
- **Resources:** identified by URIs, untyped
- **Links:** href, src, . . . limited, non-descriptive
- **User:** Exciting world - semantics of the resource, however, gleaned from content
- **Machine:** Very little information available - significance of the links only evident from the context around the anchor.



SOME RIGHTS RESERVED

The Semantic Web

- **Resources:** Globally identified by URIs or Locally scoped (Blank), Extensible, Relational.
- **Links:** Identified by URIs, Extensible, Relational.
- **User:** Even more exciting world, richer user experience.
- **Machine:** More processable information is available (Data Web).
- **Computers and people:** Work, learn and exchange knowledge effectively.



SOME RIGHTS RESERVED

Towards a "Machine-Actionable Web"

- **Recall:** We need external agreement on meaning of annotation tags.
- **Idea:** standardize them in a community process (e.g. DIN or ISO)
- **Problem:** Inflexible, Limited number of things can be expressed

SOME RIGHTS RESERVED

Towards a "Machine-Actionable Web"

- **Recall:** We need external agreement on meaning of annotation tags.
- **Idea:** standardize them in a community process (e.g. DIN or ISO)
- **Problem:** Inflexible, Limited number of things can be expressed
- **Better:** Use ontologies to specify meaning of annotations
  - Ontologies provide a vocabulary of terms
  - New terms can be formed by combining existing ones
  - Meaning (semantics) of such terms is formally specified
  - Can also specify relationships between terms in multiple ontologies

SOME RIGHTS RESERVED

Towards a "Machine-Actionable Web"

- **Recall:** We need external agreement on meaning of annotation tags.
- **Idea:** standardize them in a community process (e.g. DIN or ISO)
- **Problem:** Inflexible, Limited number of things can be expressed
- **Better:** Use ontologies to specify meaning of annotations
  - Ontologies provide a vocabulary of terms
  - New terms can be formed by combining existing ones
  - Meaning (semantics) of such terms is formally specified
  - Can also specify relationships between terms in multiple ontologies
- Inference with annotations and ontologies (get out more than you put in!)
  - Standardize annotations in RDF [**w3c:rdf-concepts**] or RDFa [**w3c:rdfa-primer**] and ontologies on OWL [**w3c:owl2-overview**]
  - Harvest RDF and RDFa in to a triplestore or OWL reasoner.
  - Query that for implied knowledge (e.g. chaining multiple facts from Wikipedia)

    SPARQL: Who was US President when Barack Obama was Born?
    DBPedia: John F. Kennedy (was president in August 1961)

SOME RIGHTS RESERVED

16.1.4 Other Knowledge Representation Approaches

SOME RIGHTS RESERVED

Frame Notation as Logic with Locality

- Predicate Logic: (where is the locality?)

$catch\_22 \in catch\_object$	There is an instance of catching
$catcher(catch\_22, jack\_2)$	Jack did the catching
$caught(catch\_22, ball\_5)$	He caught a certain ball

- **Definition 1.22.** Frames (group everything around the object)

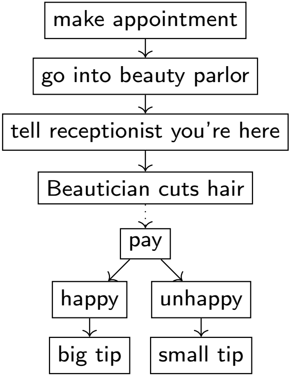
```
(catch_object catch_22
              (catcher jack_2)
              (caught ball_5))
```

+ Once you have decided on a frame, all the information is local
+ easy to define schemes for concept (aka. types in feature structures)
− how to determine frame, when to choose frame (log/chair)

SOME RIGHTS RESERVED

KR involving Time (Scripts [Shank '77])

- **Idea:** Organize typical event sequences, actors and props into representation.
  - **Definition 1.23.** A script is a structured representation describing a stereotyped sequence of events in a particular context. Structurally, scripts are very much like frames, except the values that fill the slots must be ordered.
  - **Example 1.24.** getting your hair cut (at a beauty parlor)
    - props, actors as "script variables"
    - events in a (generalized) sequence
  - use script material for
    - anaphora, bridging references
    - default common ground
    - to fill in missing material into situations



SOME RIGHTS RESERVED

Other Representation Formats (not covered)

- Procedural Representations (production systems)
- Analogical representations (interesting but not here)
- Iconic representations (interesting but very difficult to formalize)
- If you are interested, come see me off-line

SOME RIGHTS RESERVED

16.2 Logic-Based Knowledge Representation

SOME RIGHTS RESERVED

Logic-Based Knowledge Representation

- Logic (and related formalisms) have a well-defined semantics
  - explicitly (gives more understanding than statistical/neural methods)
  - transparently (symbolic methods are monotonic)
  - systematically (we can prove theorems about our systems)
- Problems with logic-based approaches
  - Where does the world knowledge come from? (Ontology problem)
  - How to guide search induced by logical calculi (combinatorial explosion)
- **One possible answer:** description logics. (next couple of times)

SOME RIGHTS RESERVED

16.2.1 Propositional Logic as a Set Description Language

SOME RIGHTS RESERVED

Propositional Logic as Set Description Language

- **Idea:** Use propositional logic as a set description language: (variant syntax/semantics)
- **Definition 2.1.** Let $\mathrm{PL}^0_{\mathrm{DL}}$ be given by the following grammar for the $\mathrm{PL}^0_{\mathrm{DL}}$ concepts. (formulae)

$$\mathcal{L} ::= C \mid \top \mid \bot \mid \overline{\mathcal{L}} \mid \mathcal{L} \sqcap \mathcal{L} \mid \mathcal{L} \sqcup \mathcal{L} \mid \mathcal{L} \sqsubseteq \mathcal{L} \mid \mathcal{L} \equiv \mathcal{L}$$

  i.e. $\mathrm{PL}^0_{\mathrm{DL}}$ formed from
  - atomic formulae ($\widehat{=}$ propositional variables)
  - concept intersection ($\sqcap$) ($\widehat{=}$ conjunction $\wedge$)
  - concept complement ($\overline{\cdot}$) ($\widehat{=}$ negation $\neg$)
  - concept union ($\sqcup$), subsumption ($\sqsubseteq$), and equivalence ($\equiv$) defined from these.($\widehat{=}$ $\vee$, $\Rightarrow$, and $\Leftrightarrow$)
- **Definition 2.2 (Formal Semantics).** Let $\mathcal{D}$ be a given set (called the domain of discourse) and $\varphi\colon \mathcal{V}_0 \to \mathcal{P}(\mathcal{D})$, then we define
  - $[\![P]\!] := \varphi(P)$, (remember $\varphi(P) \subseteq \mathcal{D}$).
  - $[\![\mathsf{A} \sqcap \mathsf{B}]\!] := [\![\mathsf{A}]\!] \cap [\![\mathsf{B}]\!]$ and $\left[\!\left[\overline{\mathsf{A}}\right]\!\right] := \mathcal{D} \backslash [\![\mathsf{A}]\!]$ ...

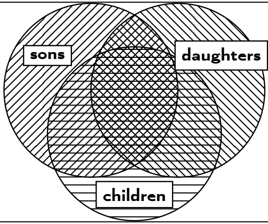
  We call this construction the set description semantics of PL^0.
- **Note:** $\langle \mathrm{PL}^0_{\mathrm{DL}}, \mathcal{S}, [\![\cdot]\!] \rangle$, where $\mathcal{S}$ is the class of possible domains forms a logical system.

SOME RIGHTS RESERVED

Concept Axioms

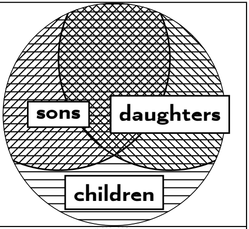
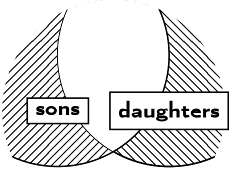
- **Observation:** Set-theoretic semantics of ‘true’ and ‘false’ ($\top := \varphi \sqcup \overline{\varphi}$ $\bot := \varphi \sqcap \overline{\varphi}$)

  $[\![\top]\!] = [\![p]\!] \cup [\![\overline{p}]\!] = [\![p]\!] \cup \mathcal{D} \backslash [\![p]\!] = \mathcal{D}$ Analogously: $[\![\bot]\!] = \emptyset$
- **Idea:** Use logical axioms to describe the world (Axioms restrict the class of admissible domain structures)
- **Definition 2.3.** A concept axiom is a $\mathrm{PL}^0_{\mathrm{DL}}$ formula A that is assumed to be true in the world.
- **Definition 2.4 (Set-Theoretic Semantics of Axioms).** A is true in domain of discourse $\mathcal{D}$ iff $[\![A]\!] = \mathcal{D}$.
- **Example 2.5.** A world with three concepts and no concept axioms

concepts	Set Semantics
child daughter son	

Effects of Axioms to Siblings

- **Example 2.6.** We can use concept axioms to describe the world from 2.5.

Axioms	Semantics
$\text{son} \sqsubseteq \text{child}$ iff $\llbracket\overline{\text{son}}\rrbracket \cup \llbracket\text{child}\rrbracket = \mathcal{D}$ iff $\llbracket\text{son}\rrbracket \subseteq \llbracket\text{child}\rrbracket$ $\text{daughter} \sqsubseteq \text{child}$ iff $\left\llbracket\overline{\text{daughter}}\right\rrbracket \cup \llbracket\text{child}\rrbracket = \mathcal{D}$ iff $\llbracket\text{daughter}\rrbracket \subseteq \llbracket\text{child}\rrbracket$	
$\overline{\text{son} \sqcap \text{daughter}}$ $\text{child} \sqsubseteq \text{son} \sqcup \text{daughter}$	

SOME RIGHTS RESERVED

Propositional Identities

Name	for $\sqcap$	for $\sqcup$
Idempot.	$\varphi \sqcap \varphi = \varphi$	$\varphi \sqcup \varphi = \varphi$
Identity	$\varphi \sqcap \top = \varphi$	$\varphi \sqcup \bot = \varphi$
Absorpt.	$\varphi \sqcup \top = \top$	$\varphi \sqcap \bot = \bot$
Commut.	$\varphi \sqcap \psi = \psi \sqcap \varphi$	$\varphi \sqcup \psi = \psi \sqcup \varphi$
Assoc.	$\varphi \sqcap (\psi \sqcap \theta) = (\varphi \sqcap \psi) \sqcap \theta$	$\varphi \sqcup (\psi \sqcup \theta) = (\varphi \sqcup \psi) \sqcup \theta$
Distrib.	$\varphi \sqcap (\psi \sqcup \theta) = (\varphi \sqcap \psi) \sqcup (\varphi \sqcap \theta)$	$\varphi \sqcup (\psi \sqcap \theta) = (\varphi \sqcup \psi) \sqcap (\varphi \sqcup \theta)$
Absorpt.	$\varphi \sqcap (\varphi \sqcup \theta) = \varphi$	$\varphi \sqcup \varphi \sqcap \theta = \varphi \sqcap \theta$
Morgan	$\overline{\varphi \sqcap \psi} = \overline{\varphi} \sqcup \overline{\psi}$	$\overline{\varphi \sqcup \psi} = \overline{\varphi} \sqcap \overline{\psi}$
dneg	$\overline{\overline{\varphi}} = \varphi$	

SOME RIGHTS RESERVED

Set-Theoretic Semantics and Predicate Logic

- **Definition 2.7.** Translation into PL^1 (borrow semantics from that)
  - recursively add argument variable x
  - change back $\sqcap, \sqcup, \sqsubseteq, \equiv$ to $\wedge, \vee, \Rightarrow, \Leftrightarrow$
  - universal closure for x at formula level.

Definition	Comment
$\overline{p}^{fo(x)} := p(x)$ $\overline{\overline{A}}^{fo(x)} := \neg\overline{A}^{fo(x)}$ $\overline{A \sqcap B}^{fo(x)} := \overline{A}^{fo(x)} \wedge \overline{B}^{fo(x)}$ $\overline{A \sqcup B}^{fo(x)} := \overline{A}^{fo(x)} \vee \overline{B}^{fo(x)}$ $\overline{A \sqsubseteq B}^{fo(x)} := \overline{A}^{fo(x)} \Rightarrow \overline{B}^{fo(x)}$ $\overline{A = B}^{fo(x)} := \overline{A}^{fo(x)} \Leftrightarrow \overline{B}^{fo(x)}$	 $\wedge$ vs. $\sqcap$ $\vee$ vs. $\sqcup$ $\Rightarrow$ vs. $\sqsubseteq$ $\Leftrightarrow$ vs. $=$
$\overline{A}^{fo} := (\forall x.\overline{A}^{fo(x)})$	for formulae

SOME RIGHTS RESERVED

Translation Examples

- **Example 2.8.** We translate the concept axioms from ??? to fortify our intuition:

$$\overline{\text{son} \sqsubseteq \text{child}}^{fo} = \forall x.\text{son}(x) \Rightarrow \text{child}(x)$$

$$\overline{\text{daughter} \sqsubseteq \text{child}}^{fo} = \forall x.\text{daughter}(x) \Rightarrow \text{child}(x)$$

$$\overline{\overline{\text{son} \sqcap \text{daughter}}}^{fo} = \forall x.\overline{\text{son}(x) \wedge \text{daughter}(x)}$$

$$\overline{\text{child} \sqsubseteq \text{son} \sqcup \text{daughter}}^{fo} = \forall x.\text{child}(x) \Rightarrow (\text{son}(x) \vee \text{daughter}(x))$$

- What are the advantages of translation to PL^1?
  - **theoretically**: A better understanding of the semantics
  - **computationally**: Description Logic Framework, but NOTHING for PL^0
    - we can follow this pattern for richer description logics.
    - many tests are decidable for PL^0, but not for PL^1. (Description Logics?)

16.2.2 Ontologies and Description Logics

SOME RIGHTS RESERVED

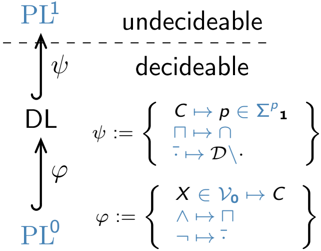
Ontologies aka. "World Descriptions"

- **Definition 2.9 (Classical).** An ontology is a representation of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse.
- **Remark:** 2.9 is very general, and depends on what we mean by "representation", "entities", "types", and "interrelationships".
  This may be a feature, and not a bug, since we can use the same intuitions across a variety of representations.
- **Definition 2.10.** An ontology consists of a formal system $\langle\mathcal{L},\mathcal{C},\mathcal{M},\vDash\rangle$ with concept axiom (expressed in $\mathcal{L}$) about
  - individuals: concrete entities in a domain of discourse,
  - concepts: particular collections of individuals that share properties and aspects – the instances of the concept, and
  - relations: ways in which individuals can be related to one another.
- **Example 2.11.** Semantic networks are ontologies. (relatively informal)
- **Example 2.12.** $\mathrm{PL}^0_{\mathrm{DL}}$ is an ontology format. (formal, but relatively weak)
- **Example 2.13.** PL^1 is an ontology format as well. (formal, expressive)

SOME RIGHTS RESERVED

The Description Logic Paradigm

- **Idea:** Build a whole family of logics for describing sets and their relations. (tailor their expressivity and computational properties)
- **Definition 2.14.** A description logic is a formal system for talking about collections of objects and their relations that is at least as expressive as PL^0 with set-theoretic semantics and offers individuals and relations.
  A description logic has the following four components:
  - a formal language $\mathcal{L}$ with logical constants $\sqcap$, $\overline{\cdot}$, $\sqcup$, $\sqsubseteq$, and $\equiv$,
  - a set-theoretic semantics $\langle\mathcal{D},[\![\cdot]\!]\rangle$,
  - a translation into first-order logic that is compatible with $\langle\mathcal{D},[\![\cdot]\!]\rangle$, and
  - a calculus for $\mathcal{L}$ that induces a decision procedure for $\mathcal{L}$-satisfiability.



$$\psi := \left\{\begin{array}{l} C \mapsto p \in {\Sigma^p}_1 \\ \sqcap \mapsto \cap \\ \overline{\cdot} \mapsto \mathcal{D}\backslash\cdot \end{array}\right\}$$

$$\varphi := \left\{\begin{array}{l} X \in \mathcal{V}_0 \mapsto C \\ \wedge \mapsto \sqcap \\ \neg \mapsto \overline{\cdot} \end{array}\right\}$$

- **Definition 2.15.** Given a description logic $\mathcal{D}$, a $\mathcal{D}$ ontology consists of
  - a terminology (or TBox): concepts and roles and a set of concept axioms that describe them, and
  - assertions (or ABox): a set of individuals and statements about concept membership and role relationships for them.

SOME RIGHTS RESERVED

TBoxes in Description Logics

- Let $\mathcal{D}$ be a description logic with concepts $\mathcal{C}$.
- **Definition 2.16.** A concept definition is a pair $c{=}\mathrm{C}$, where c is a new concept name and $\mathrm{C} \in \mathcal{C}$ is a $\mathcal{D}$-formula.
- **Example 2.17.** We can define $\mathrm{mother}{=}\mathrm{woman} \sqcap \mathrm{has\_child}$.
- **Definition 2.18.** A concept definition $c{=}C$ is called recursive, iff c occurs in C.
- **Definition 2.19.** An TBox is a finite set of concept definitions and concept axioms. It is called acyclic, iff it does not contain recursive definitions.
- **Definition 2.20.** A formula A is called normalized wrt. an TBox $\mathcal{T}$, iff it does not contain concepts defined in $\mathcal{T}$. (convenient)
- **Definition 2.21 (Algorithm).** (for arbitrary DLs)
  Input: A formula A and a TBox $\mathcal{T}$.
  - While [A contains concept c and $\mathcal{T}$ a concept definition $c{=}\mathrm{C}$]
    - substitute c by C in A.
- **Lemma 2.22.** *This algorithm terminates for acyclic TBoxes, but results can be exponentially large.*

SOME RIGHTS RESERVED

16.2.3 Description Logics and Inference

SOME RIGHTS RESERVED

Kinds of Inference in Description Logics

- **Definition 2.23.** Ontology systems employ three main reasoning services:
  - Consistency test: is a concept definition satisfiable?
  - Subsumption test: does a concept subsume another?
  - Instance test: is an individual an example of a concept?
- **Problem:** decidability, complexity, algorithm

SOME RIGHTS RESERVED

Consistency Test

- **Definition 2.24.** We call a concept C consistent, iff there is no concept A, with both $C \sqsubseteq A$ and $C \sqsubseteq \overline{A}$.
- Or equivalently:
- **Definition 2.25.** A concept C is called inconsistent, iff $[\![C]\!] = \emptyset$ for all $\mathcal{D}$.
- **Example 2.26 (T-Box in $\mathrm{PL}^0_{\mathrm{DL}}$).**

man	$=$	person $\sqcap$ has_Y	person with y-chromosome
woman	$=$	person $\sqcap$ $\overline{\text{has\_Y}}$	person without y-chromosome
hermaphrodite	$=$	man $\sqcap$ woman	man and woman

This specification is inconsistent, i.e. $[\![\text{hermaphrodite}]\!] = \emptyset$ for all $\mathcal{D}$.

- **Algorithm:** Satisfiability test (usually NP-hard)

we know how to do this, e.g. tableaux, resolution, DPLL in $\mathrm{PL}^0_{\mathrm{DL}}$.

SOME RIGHTS RESERVED

Subsumption Test

- **Example 2.27.** In this case trivial

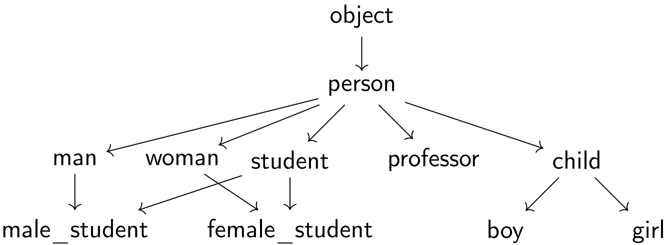
axiom	entailed subsumption relation
$\mathrm{man} = \mathrm{person} \sqcap \mathrm{has\_Y}$	$\mathrm{man} \sqsubseteq \mathrm{person}$
$\mathrm{woman} = \mathrm{person} \sqcap \overline{\mathrm{has\_Y}}$	$\mathrm{woman} \sqsubseteq \mathrm{person}$

- **Definition 2.28.** A subsumes B (modulo a set $\mathcal{A}$ of concept axioms), iff $[\![B]\!] \subseteq [\![A]\!]$ for all interpretations $\mathcal{D}$ that satisfy $\mathcal{A}$.
- **Observation:** Or equivalently, iff $\mathcal{A} \sqsubseteq B \sqsubseteq A = \top$
- **Reduction to consistency test:** (need to implement only one)
  In PL^0, $\mathcal{A} \Rightarrow (A \Rightarrow B)$ is valid iff $\mathcal{A} \wedge A \wedge \neg B$ is inconsistent.
- **In our example:** The concept person subsumes woman and man.

SOME RIGHTS RESERVED

Classification

- The subsumption relation among all concepts (subsumption graph)
- Visualization of the subsumption graph for inspection (plausibility)
- **Definition 2.29.** Classification is the computation of the subsumption graph.
- **Example 2.30.** (not always so trivial)



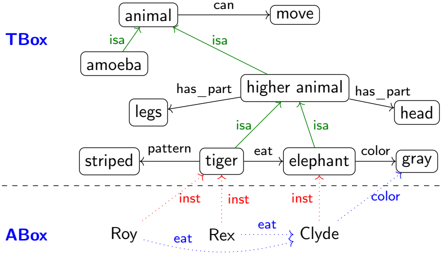
SOME RIGHTS RESERVED

Instance Test: Inferring Concept Membership

- **Definition 2.31.** An instance test computes whether given an ontology an individual is a member of a given concept.
- **Remark:** This is not something we can do in $\mathrm{PL}^0_{\mathrm{DL}}$, which is a TBox-only system. PL^1 (where concepts are predicate constants an assertions are atoms) suffices.
- **Example 2.32.** If we define a concept "mother" as "woman who has a child", and have the assertions "Mary is a woman" and "Jesus is a child of Mary", then we can infer that "Mary" is a "Mother", e.g. in the $\mathcal{ND}^1$:

$$\forall x.m(x) \Leftrightarrow w(x) \wedge (\exists y.hc(x,y)), w(M), hc(M,J) \vdash_{\mathcal{ND}^1} m(M)$$

- **Remark:** This only works in the presence of concept definitions, not in a purely descriptive framework like semantic networks:



SOME RIGHTS RESERVED

16.3 A simple Description Logic: ALC

16.3.1 Basic ALC: Concepts, Roles, and Quantification

SOME RIGHTS RESERVED

Motivation for $\mathcal{ALC}$ (Prototype Description Logic)

- Propositional logic (PL^0) is not expressive enough!
- **Example 3.1.** "mothers are women that have a child"
- **Reason:** There are no quantifiers in PL^0 (existential (∃) and universal (∀))
- **Idea:** Use first-order predicate logic (PL^1)

$$\forall x.mother(x) \Leftrightarrow woman(x) \wedge (\exists y.has\_child(x,y))$$

- **Problem:** Complex algorithms, non-termination (PL^1 is too expressive)
- **Idea:** Try to travel the middle ground
  More expressive than PL^0 (quantifiers) but weaker than PL^1. (still tractable)
- **Technique:** Allow only "restricted quantification", where quantified variables only range over values that can be reached via a binary relation like *has_child*.

SOME RIGHTS RESERVED

Syntax of $\mathcal{ALC}$

- **Definition 3.2 (Concepts).** (aka. "predicates" in PL^1 or "propositional variables" in $\mathrm{PL}^0_{\mathrm{DL}}$)
  Concepts in DLs represent collections of objects.
- . . . like classes in OOP.
- **Definition 3.3 (Special Concepts).** The top concept $\top$ (for "true" or "all") and the bottom concept $\bot$ (for "false" or "none").
- **Example 3.4.** person, woman, man, mother, professor, student, car, BMW, computer, computer program, heart attack risk, furniture, table, leg of a chair, . . .
- **Definition 3.5.** Roles represent binary relations (like in PL^1)
- **Example 3.6.** has_child, has_son, has_daughter, loves, hates, gives_course, executes_computer_program, has_leg_of_table, has_wheel, has_motor, . . .
- **Definition 3.7 (Grammar).** The formulae of $\mathcal{ALC}$ are given by the following grammar: $F_{\mathcal{ALC}} ::= C \mid \top \mid \bot \mid \overline{F_{\mathcal{ALC}}} \mid F_{\mathcal{ALC}} \sqcap F_{\mathcal{ALC}} \mid F_{\mathcal{ALC}} \sqcup F_{\mathcal{ALC}} \mid \exists \mathrm{R}.F_{\mathcal{ALC}} \mid \forall \mathrm{R}.F_{\mathcal{ALC}}$

SOME RIGHTS RESERVED

Syntax of $\mathcal{ALC}$: Examples

- **Example 3.8.** person $\sqcap$ $\exists$has_child.student
  $\widehat{=}$ The set of persons that have a child which is a student
  $\widehat{=}$ parents of students
- **Example 3.9.** person $\sqcap$ $\exists$has_child.$\exists$has_child.student
  $\widehat{=}$ grandparents of students
- **Example 3.10.** person $\sqcap$ $\exists$has_child.$\exists$has_child.(student $\sqcup$ teacher)
  $\widehat{=}$ grandparents of students or teachers
- **Example 3.11.** person $\sqcap$ $\forall$has_child.student
  $\widehat{=}$ parents whose children are all students
- **Example 3.12.** person $\sqcap$ $\forall \mathrm{haschild}$.$\exists$has_child.student
  $\widehat{=}$ grandparents, whose children all have at least one child that is a student

SOME RIGHTS RESERVED

More $\mathcal{ALC}$ Examples

- **Example 3.13.** $\text{car} \sqcap \exists \text{has\_part}.\exists \text{made\_in}.\overline{\text{EU}}$
  $\widehat{=}$ cars that have at least one part that has not been made in the EU
- **Example 3.14.** $\text{student} \sqcap \forall \text{audits\_course}.\text{graduatelevelcourse}$
  $\widehat{=}$ students, that only audit graduate level courses
- **Example 3.15.** $\text{house} \sqcap \forall \text{has\_parking}.\text{off\_street}$ $\widehat{=}$ houses with off-street parking
- **Note:** $p \sqsubseteq q$ can still be used as an abbreviation for $\overline{p} \sqcup q$.
- **Example 3.16.** $\text{student} \sqcap \forall \text{audits\_course}.(\exists \text{hastutorial}.\top \sqsubseteq \forall \text{has\_TA}.\text{woman})$
  $\widehat{=}$ students that only audit courses that either have no tutorial or tutorials that are TAed by women

SOME RIGHTS RESERVED

$\mathcal{ALC}$ Concept Definitions

- **Idea:** Define new concepts from known ones.
- **Definition 3.17.** A concept definition is a pair consisting of a new concept name (the definiendum) and an $\mathcal{ALC}$ formula (the definiens). Concepts that are not definienda are called primitive.
- We extend the $\mathcal{ALC}$ grammar from ??? by the production

$$CD_{\mathcal{ALC}} ::= C = F_{\mathcal{ALC}}$$

- **Example 3.18.**

Definition	rec?
$\text{man} = \text{person} \sqcap \exists\text{has\_chrom}.\text{Y\_chrom}$	-
$\text{woman} = \text{person} \sqcap \forall\text{has\_chrom}.\overline{\text{Y\_chrom}}$	-
$\text{mother} = \text{woman} \sqcap \exists\text{has\_child}.\text{person}$	-
$\text{father} = \text{man} \sqcap \exists\text{has\_child}.\text{person}$	-
$\text{grandparent} = \text{person} \sqcap \exists\text{has\_child}.(\text{mother} \sqcup \text{father})$	-
$\text{german} = \text{person} \sqcap \exists\text{has\_parents}.\text{german}$	+
$\text{number\_list} = \text{empty\_list} \sqcup \exists\text{is\_first}.\text{number} \sqcap \exists\text{is\_rest}.\text{number\_list}$	+

SOME RIGHTS RESERVED

TBox Normalization in $\mathcal{ALC}$

- **Definition 3.19.** We call an $\mathcal{ALC}$ formula φ normalized wrt. a set of concept definitions, iff all concepts occurring in φ are primitive.
- **Definition 3.20.** Given a set $\mathcal{D}$ of concept definitions, normalization is the process of replacing in an $\mathcal{ALC}$ formula φ all occurrences of definienda in $\mathcal{D}$ with their definientia.
- **Example 3.21 (Normalizing grandparent).**

$$\begin{array}{ll} & \textbf{grandparent} \\ \mapsto & \textbf{person} \sqcap \exists\textbf{has\_child}.(\textbf{mother} \sqcup \textbf{father}) \\ \mapsto & \textbf{person} \sqcap \exists\textbf{has\_child}.(\textbf{woman} \sqcap \exists\textbf{has\_child}.\textbf{person} \sqcap \textbf{man} \sqcap \exists\textbf{has\_child}.\textbf{person}) \\ \mapsto & \textbf{person} \sqcap \exists\textbf{has\_child}.(\textbf{person} \sqcap \exists\textbf{has\_chrom}.\textbf{Y\_chrom} \sqcap \exists\textbf{has\_child}.\textbf{person} \sqcap \textbf{person} \sqcap \exists\textbf{has\_chrom}.\textbf{Y\_chrom} \sqcap \exists\textbf{ha} \end{array}$$

- **Observation 3.22.** *Normalization results can be exponential.* (*contain redundancies*)
- **Observation 3.23.** *Normalization need not terminate on cyclic TBoxes.*
- **Example 3.24.**

$$\begin{array}{lll} \text{german} & \mapsto & \text{person} \sqcap \exists\text{has\_parents}.\text{german} \\ & \mapsto & \text{person} \sqcap \exists\text{has\_parents}.(\text{person} \sqcap \exists\text{has\_parents}.\text{german}) \\ & \mapsto & \ldots \end{array}$$

SOME RIGHTS RESERVED

Semantics of $\mathcal{ALC}$

- $\mathcal{ALC}$ semantics is an extension of the set-semantics of propositional logic.
- **Definition 3.25.** A model for $\mathcal{ALC}$ is a pair $\langle U_{\mathcal{A}}, [[\cdot]]\rangle$, where $U_{\mathcal{A}}$ is a non-empty set called the domain of discourse and $[[\cdot]]$ a mapping called the interpretation, such that

Op.	formula semantics
	$[\![c]\!] \subseteq U_{\mathcal{A}} = [\![\top]\!] \quad [\![\bot]\!] = \emptyset \quad [\![r]\!] \subseteq U_{\mathcal{A}} \times U_{\mathcal{A}}$
$\overline{\cdot}$	$[\![\overline{\varphi}]\!] = \overline{[\![\varphi]\!]} = U_{\mathcal{A}} \backslash [\![\varphi]\!]$
$\sqcap$	$[\![\varphi \sqcap \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$
$\sqcup$	$[\![\varphi \sqcup \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$
$\exists\mathrm{R}.$	$[\![\exists\mathrm{R}.\varphi]\!] = \{x \in U_{\mathcal{A}} \,\vert\, \exists y.\langle x,y\rangle \in [\![\mathrm{R}]\!] \text{ and } y \in [\![\varphi]\!]\}$
$\forall\mathrm{R}.$	$[\![\forall\mathrm{R}.\varphi]\!] = \{x \in U_{\mathcal{A}} \,\vert\, \forall y.\text{if } \langle x,y\rangle \in [\![\mathrm{R}]\!] \text{ then } y \in [\![\varphi]\!]\}$

- Alternatively we can define the semantics of $\mathcal{ALC}$ by translation into PL^1.
- **Definition 3.26.** The translation of $\mathcal{ALC}$ into PL^1 extends the one from ??? by the following quantifier rules:

$$\overline{\forall\mathrm{R}.\varphi}^{fo(x)} := (\forall y.\mathrm{R}(x,y) \Rightarrow \overline{\varphi}^{fo(y)}) \quad \overline{\exists\mathrm{R}.\varphi}^{fo(x)} := (\exists y.\mathrm{R}(x,y) \wedge \overline{\varphi}^{fo(y)})$$

- **Observation 3.27.** *The set-theoretic semantics from 3.25 and the "semantics-by-translation" from 3.26 induce the same notion of satisfiability.*

SOME RIGHTS RESERVED

$\mathcal{ALC}$ Identities

1	$\overline{\exists \mathsf{R}.\varphi} = \forall \mathsf{R}.\overline{\varphi}$	3	$\overline{\forall \mathsf{R}.\varphi} = \exists \mathsf{R}.\overline{\varphi}$
2	$\forall \mathsf{R}.(\varphi \sqcap \psi) = \forall \mathsf{R}.\varphi \sqcap \forall \mathsf{R}.\psi$	4	$\exists \mathsf{R}.(\varphi \sqcup \psi) = \exists \mathsf{R}.\varphi \sqcup \exists \mathsf{R}.\psi$

- Proof of 1

$$\begin{aligned}
[[\overline{\exists \mathsf{R}.\varphi}]] = \mathcal{D} \backslash \llbracket \exists \mathsf{R}.\varphi \rrbracket &= \mathcal{D} \backslash \{x \in \mathcal{D} \mid \exists y.(\langle x,y\rangle \in \llbracket \mathsf{R} \rrbracket) \text{ and } (y \in \llbracket \varphi \rrbracket)\} \\
&= \{x \in \mathcal{D} \mid \text{not } \exists y.(\langle x,y\rangle \in \llbracket \mathsf{R} \rrbracket) \text{ and } (y \in \llbracket \varphi \rrbracket)\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x,y\rangle \in \llbracket \mathsf{R} \rrbracket) \text{ then } (y \notin \llbracket \varphi \rrbracket)\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x,y\rangle \in \llbracket \mathsf{R} \rrbracket) \text{ then } (y \in (\mathcal{D} \backslash \llbracket \varphi \rrbracket))\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x,y\rangle \in \llbracket \mathsf{R} \rrbracket) \text{ then } (y \in \llbracket \overline{\varphi} \rrbracket)\} \\
&= \llbracket \forall \mathsf{R}.\overline{\varphi} \rrbracket
\end{aligned}$$

SOME RIGHTS RESERVED

Negation Normal Form

- **Definition 3.28 (NNF).** An $\mathcal{ALC}$ formula is in negation normal form (NNF), iff complement ($\overline{\cdot}$) is only applied to primitive concept.
- Use the $\mathcal{ALC}$ identities as rules to compute it. (in linear time)
- **Example 3.29.**

example	by rule
$\overline{\exists R.(\forall S.e \sqcap \overline{\forall S.d})}$	
$\mapsto \forall R.\overline{\forall S.e \sqcap \overline{\forall S.d}}$	$\overline{\exists R.\varphi} \mapsto \forall R.\overline{\varphi}$
$\mapsto \forall R.(\overline{\forall S.e} \sqcup \overline{\overline{\forall S.d}})$	$\overline{\varphi \sqcap \psi} \mapsto \overline{\varphi} \sqcup \overline{\psi}$
$\mapsto \forall R.(\exists S.\overline{e} \sqcup \overline{\overline{\forall S.d}})$	$\overline{\forall R.\varphi} \mapsto \exists R.\overline{\varphi}$
$\mapsto \forall R.(\exists S.\overline{e} \sqcup \forall S.d)$	$\overline{\overline{\varphi}} \mapsto \varphi$

SOME RIGHTS RESERVED

$\mathcal{ALC}$ with Assertions about Individuals

- **Definition 3.30.** We define the ABox assertions for $\mathcal{ALC}$:
  - Role assertions$a{:}\varphi$ (a is a φ)
  - a R b (a stands in relation R to b)

  assertions make up the ABox in $\mathcal{ALC}$.
- **Definition 3.31.** Let $\langle\mathcal{D},[[\cdot]]\rangle$ be a model for $\mathcal{ALC}$, then we define
  - $[\![a{:}\varphi]\!]=\mathsf{T}$, iff $[\![a]\!]\in[\![\varphi]\!]$, and
  - $[\![a\ \mathrm{R}\ b]\!]=\mathsf{T}$, iff $([\![a]\!],[\![b]\!])\in[\![\mathrm{R}]\!]$.
- **Definition 3.32.** We extend the PL^1 translation of $\mathcal{ALC}$ to $\mathcal{ALC}$ assertions:
  - $\overline{a{:}\varphi}^{fo}:=\overline{\varphi}^{fo(a)}$, and
  - $\overline{a\ \mathrm{R}\ b}^{fo}:=\mathrm{R}(a,b)$.

16.3.2 Inference for ALC

$\mathcal{T}_{\mathcal{ALC}}$: A Tableau-Calculus for $\mathcal{ALC}$

- **Recap Tableaux:** A tableau calculus develops an initial tableau in a tree-formed scheme using tableau extension rules.
  A saturated tableau (no rules applicable) constitutes a refutation, if all branches are closed (end in $\bot$).
- **Definition 3.33.** The $\mathcal{ALC}$ tableau calculus $\mathcal{T}_{\mathcal{ALC}}$ acts on assertions:
  - $x{:}\varphi$ (x inhabits concept φ)
  - $x\ \mathrm{R}\ y$ (x and y are in relation R)

  where φ is a normalized $\mathcal{ALC}$ concept in negation normal form with the following rules:

$$\frac{\begin{array}{c}x{:}c\\ x{:}\overline{c}\end{array}}{\bot}\,\mathcal{T}_{\bot} \qquad \frac{x{:}\varphi\sqcap\psi}{\begin{array}{c}x{:}\varphi\\ x{:}\psi\end{array}}\,\mathcal{T}_{\sqcap} \qquad \frac{x{:}\varphi\sqcup\psi}{x{:}\varphi \;\big|\; x{:}\psi}\,\mathcal{T}_{\sqcup} \qquad \frac{\begin{array}{c}x{:}\forall\mathrm{R}.\varphi\\ x\ \mathrm{R}\ y\end{array}}{y{:}\varphi}\,\mathcal{T}_{\forall} \qquad \frac{x{:}\exists\mathrm{R}.\varphi}{\begin{array}{c}x\ \mathrm{R}\ y\\ y{:}\varphi\end{array}}\,\mathcal{T}_{\exists}$$

- To test consistency of a concept φ, normalize φ to ψ, initialize the tableau with $x{:}\psi$, saturate. Open branches $\rightsquigarrow$ consistent. (x arbitrary)

SOME RIGHTS RESERVED

$\mathcal{T}_{\mathcal{ALC}}$ Examples

- **Example 3.34 (Tableau Proofs).** We have two similar conjectures about children.
  - $x{:}\forall\text{has\_child}.\text{man} \sqcap \exists\text{has\_child}.\overline{\text{man}}$ (all sons, but a daughter)

$x{:}\forall\text{has\_child}.\text{man} \sqcap \exists\text{has\_child}.\overline{\text{man}}$	initial
$x{:}\forall\text{has\_child}.\text{man}$	$\mathcal{T}_{\sqcap}$
$x{:}\exists\text{has\_child}.\overline{\text{man}}$	$\mathcal{T}_{\sqcap}$
x has_child y	$\mathcal{T}_{\exists}$
$y{:}\overline{\text{man}}$	$\mathcal{T}_{\exists}$
$\bot$	$\mathcal{T}_{\bot}$
inconsistent	

  - $x{:}\forall\text{has\_child}.\text{man} \sqcap \exists\text{has\_child}.\text{man}$ (only sons, and at least one)

$x{:}\forall\text{has\_child}.\text{man} \sqcap \exists\text{has\_child}.\text{man}$	initial
$x{:}\forall\text{has\_child}.\text{man}$	$\mathcal{T}_{\sqcap}$
$x{:}\exists\text{has\_child}.\text{man}$	$\mathcal{T}_{\sqcap}$
x has_child y	$\mathcal{T}_{\exists}$
$y{:}\text{man}$	$\mathcal{T}_{\exists}$
open	

This tableau shows a model: there are two persons, x and y. y is the only child of x, y is a man.

SOME RIGHTS RESERVED

Another $\mathcal{T}_{\mathcal{ALC}}$ Example

- **Example 3.35.** $\forall\text{has\_child}.(\text{ugrad}\sqcup\text{grad})\sqcap\exists\text{has\_child}.\overline{\text{ugrad}}$ is satisfiable.
  - Let's try it on the board

SOME RIGHTS RESERVED

Another $\mathcal{T}_{\mathcal{ALC}}$ Example

- **Example 3.36.** $\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has\_child}.\overline{\text{ugrad}}$ is satisfiable.
  - Let's try it on the board
  - Tableau proof for the notes

<table>
<tr><td>1</td><td colspan="2">$x{:}\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has\_child}.\overline{\text{ugrad}}$</td><td>initial</td></tr>
<tr><td>2</td><td colspan="2">$x{:}\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad})$</td><td>$\mathcal{T}_{\sqcap}$</td></tr>
<tr><td>3</td><td colspan="2">$x{:}\exists \text{has\_child}.\overline{\text{ugrad}}$</td><td>$\mathcal{T}_{\sqcap}$</td></tr>
<tr><td>4</td><td colspan="2">$x\ \text{has\_child}\ y$</td><td>$\mathcal{T}_{\exists}$</td></tr>
<tr><td>5</td><td colspan="2">$y{:}\overline{\text{ugrad}}$</td><td>$\mathcal{T}_{\exists}$</td></tr>
<tr><td>6</td><td colspan="2">$y{:}\text{ugrad} \sqcup \text{grad}$</td><td>$\mathcal{T}_{\forall}$</td></tr>
<tr><td>7</td><td>$y{:}\text{ugrad}$</td><td>$y{:}\text{grad}$</td><td>$\mathcal{T}_{\sqcup}$</td></tr>
<tr><td>8</td><td>$\bot$</td><td>open</td><td></td></tr>
</table>

The left branch is closed, the right one represents a model: y is a child of x, y is a graduate student, x hat exactly one child: y.

SOME RIGHTS RESERVED

Properties of Tableau Calculi

- We study the following properties of a tableau calculus $\mathcal{C}$:
  - Termination: there are no infinite sequences of inference rule applications.
  - Soundness: If φ is satisfiable, then $\mathcal{C}$ terminates with an open branch.
  - Completeness: If φ is in unsatisfiable, then $\mathcal{C}$ terminates and all branches are closed.
  - complexity of the algorithm (time and space complexity).
- Additionally, we are interested in the complexity of satisfiability itself (as a benchmark)

SOME RIGHTS RESERVED

Correctness

- **Lemma 3.37.** *If φ satisfiable, then $\mathcal{T}_{\mathcal{ALC}}$ terminates on $x{:}\varphi$ with open branch.*
- *Proof:* Let $\mathcal{M} := \langle \mathcal{D}, [\![\cdot]\!] \rangle$ be a model for φ and $w \in [\![\varphi]\!]$.

1. We define $[\![x]\!] := w$ and
$$\begin{array}{lll} \mathcal{M} \models (x{:}\psi) & \text{iff} & [\![x]\!] \in [\![\psi]\!] \\ \mathcal{M} \models x \; \mathrm{R} \; y & \text{iff} & \langle x, y \rangle \in [\![\mathrm{R}]\!] \\ \mathcal{M} \models S & \text{iff} & \mathcal{I} \models c \text{ for all } c \in S \end{array}$$
2. This gives us $\mathcal{M} \models (x{:}\varphi)$ (base case)
3. If the branch is satisfiable, then either
   - no rule applicable to leaf, (open branch)
   - or rule applicable and one new branch satisfiable. (inductive case: next)
4. There must be an open branch. (by termination)

□

Case analysis on the rules

$\mathcal{T}_{\sqcap}$ applies then $\mathcal{M}\models(x{:}\varphi\sqcap\psi)$, i.e. $[\![x]\!]\in[\![\varphi\sqcap\psi]\!]$
so $[\![x]\!]\in[\![\varphi]\!]$ and $[\![x]\!]\in[\![\psi]\!]$, thus $\mathcal{M}\models(x{:}\varphi)$ and $\mathcal{M}\models(x{:}\psi)$.

$\mathcal{T}_{\sqcup}$ applies then $\mathcal{M}\models(x{:}\varphi\sqcup\psi)$, i.e $[\![x]\!]\in[\![\varphi\sqcup\psi]\!]$
so $[\![x]\!]\in[\![\varphi]\!]$ or $[\![x]\!]\in[\![\psi]\!]$, thus $\mathcal{M}\models(x{:}\varphi)$ or $\mathcal{M}\models(x{:}\psi)$,
wlog. $\mathcal{M}\models(x{:}\varphi)$.

$\mathcal{T}_{\forall}$ applies then $\mathcal{M}\models(x{:}\forall\mathrm{R}.\varphi)$ and $\mathcal{M}\models x\ \mathrm{R}\ y$, i.e. $[\![x]\!]\in[\![\forall\mathrm{R}.\varphi]\!]$ and $\langle x,y\rangle\in[\![\mathrm{R}]\!]$,
so $[\![y]\!]\in[\![\varphi]\!]$

$\mathcal{T}_{\exists}$ applies then $\mathcal{M}\models(x{:}\exists\mathrm{R}.\varphi)$, i.e $[\![x]\!]\in[\![\exists\mathrm{R}.\varphi]\!]$,
so there is a $v\in D$ with $\langle[\![x]\!],v\rangle\in[\![\mathrm{R}]\!]$ and $v\in[\![\varphi]\!]$.
We define $[\![y]\!]:=v$, then $\mathcal{M}\models x\ \mathrm{R}\ y$ and $\mathcal{M}\models(y{:}\varphi)$

SOME RIGHTS RESERVED

Completeness of the Tableau Calculus

- **Lemma 3.38.** *Open saturated tableau branches for φ induce models for φ.*
- *Proof:* construct a model for the branch and verify for φ
  1. Let $\mathcal{B}$ be an open, saturated branch
     - we define

$$\begin{aligned}\mathcal{D} &:= \{x \mid x{:}\psi \in \mathcal{B} \text{ or } z \mathrel{\mathrm{R}} x \in \mathcal{B}\}\\ [\![c]\!] &:= \{x \mid x{:}c \in \mathcal{B}\}\\ [\![\mathrm{R}]\!] &:= \{\langle x, y\rangle \mid x \mathrel{\mathrm{R}} y \in \mathcal{B}\}\end{aligned}$$

     - well-defined since never $x{:}c, x{:}\overline{c} \in \mathcal{B}$ (otherwise $\mathcal{T}_\bot$ applies)
     - $\mathcal{M}$ satisfies all assertions $x{:}c$, $x{:}\overline{c}$ and $x \mathrel{\mathrm{R}} y$, (by construction)
  2. $\mathcal{M} \models (y{:}\psi)$, for all $y{:}\psi \in \mathcal{B}$ (on $k = \mathit{size}(\psi)$ next slide)
  3. $\mathcal{M} \models (x{:}\varphi)$.

☐

SOME RIGHTS RESERVED

Case Analysis for Induction

case $y{:}\psi = y{:}\psi_1 \sqcap \psi_2$ Then $\{y{:}\psi_1, y{:}\psi_2\} \subseteq \mathcal{B}$ ($\mathcal{T}_{\sqcap}$-rule, saturation)
so $\mathcal{M}\models(y{:}\psi_1)$ and $\mathcal{M}\models(y{:}\psi_2)$ and $\mathcal{M}\models(y{:}\psi_1 \sqcap \psi_2)$ (IH, Definition)

case $y{:}\psi = y{:}\psi_1 \sqcup \psi_2$ Then $y{:}\psi_1 \in \mathrm{B}$ or $y{:}\psi_2 \in \mathrm{B}$ ($\mathcal{T}_{\sqcup}$, saturation)
so $\mathcal{M}\models(y{:}\psi_1)$ or $\mathcal{M}\models(y{:}\psi_2)$ and $\mathcal{M}\models(y{:}\psi_1 \sqcup \psi_2)$ (IH, Definition)

case $y{:}\psi = y{:}\exists\mathbf{R}.\theta$ then $\{y \mathrm{\ R\ } z, z{:}\theta\} \subseteq \mathrm{B}$ (z new variable) ($\mathcal{T}_{\exists}$-rules, saturation)
so $\mathcal{M}\models(z{:}\theta)$ and $\mathcal{M}\models y \mathrm{\ R\ } z$, thus $\mathcal{M}\models(y{:}\exists\mathrm{R}.\theta)$. (IH, Definition)

case $y{:}\psi = y{:}\forall\mathbf{R}.\theta$ Let $\langle [\![y]\!], v\rangle \in [\![\mathrm{R}]\!]$ for some $r \in \mathcal{D}$
then $v = z$ for some variable z with $y \mathrm{\ R\ } z \in \mathrm{B}$ (construction of $[\![\mathrm{R}]\!]$)
So $z{:}\theta \in \mathcal{B}$ and $\mathcal{M}\models(z{:}\theta)$. ($\mathcal{T}_{\forall}$-rule, saturation, Def)
As v was arbitrary we have $\mathcal{M}\models(y{:}\forall\mathrm{R}.\theta)$.

SOME RIGHTS RESERVED

Termination

- **Theorem 3.39.** *$\mathcal{T}_{\mathcal{ALC}}$ terminates.*
- To prove termination of a tableau algorithm, find a well-founded measure (function) that is decreased by all rules

$$\frac{\begin{array}{c} x{:}c \\ x{:}\overline{c} \end{array}}{\bot}\ ALCTcutRule \qquad \frac{x{:}\varphi \sqcap \psi}{\begin{array}{c} x{:}\varphi \\ x{:}\psi \end{array}}\ \mathcal{T}_{\sqcap} \qquad \frac{x{:}\varphi \sqcup \psi}{x{:}\varphi \;\big|\; x{:}\psi}\ ALCTunionRule \qquad \frac{\begin{array}{c} x{:}\forall \mathrm{R}.\varphi \\ x\ \mathrm{R}\ y \end{array}}{y{:}\varphi}\ \mathcal{T}_{\forall}$$

- *Proof:* Sketch (full proof very technical)
  1. Any rule except $\mathcal{T}_{\forall}$ can only be applied once to $x{:}\psi$.
  2. Rule $\mathcal{T}_{\forall}$ applicable to $x{:}\forall \mathrm{R}.\psi$ at most as the number of R-successors of x. (those y with $x\ \mathrm{R}\ y$ above)
  3. The R-successors are generated by $x{:}\exists \mathrm{R}.\theta$ above, (number bounded by size of input formula)
  4. Every rule application to $x{:}\psi$ generates constraints $z{:}\psi'$, where ψ' a proper sub-formula of ψ.

□

SOME RIGHTS RESERVED

Complexity of $\mathcal{T}_{\mathcal{ALC}}$

- **Idea:** Work off tableau branches one after the other. (Branch size $\widehat{=}$ space complexity)
- **Observation 3.40.** *The size of the branches is polynomial in the size of the input formula:*

$$\text{branchsize} = \#(\textit{input formulae}) + \#(\exists\textit{-formulae}) \cdot \#(\forall\textit{-formulae})$$

- *Proof sketch:* Re-examine the termination proof and count: the first summand comes from ???, the second one from ??? and ???
- **Theorem 3.41.** *The satisfiability problem for $\mathcal{ALC}$ is in PSPACE.*
- **Theorem 3.42.** *The satisfiability problem for $\mathcal{ALC}$ is PSPACE-Complete.*
- *Proof sketch:* Reduce a PSPACE-complete problem to $\mathcal{ALC}$-satisfiability
- **Theorem 3.43 (Time Complexity).** *The $\mathcal{ALC}$ satisfiability problem is in EXPTIME.*
- *Proof sketch:* There can be exponentially many branches (already for PL^0)

SOME RIGHTS RESERVED

The functional Algorithm for $\mathcal{ALC}$

- **Observation:** (leads to a better treatment for $\exists$)
  - the $\mathcal{T}_{\exists}$-rule generates the constraints $x\ \mathrm{R}\ y$ and $y{:}\psi$ from $x{:}\exists\mathrm{R}.\psi$
  - this triggers the $\mathcal{T}_{\forall}$-rule for $x{:}\forall\mathrm{R}.\theta_i$, which generate $y{:}\theta_1, \ldots, y{:}\theta_n$
  - for y we have $y{:}\psi$ and $y{:}\theta_1, \ldots, y{:}\theta_n$. (do all of this in a single step)
  - we are only interested in non-emptiness, not in particular witnesses (leave them out)
- **Definition 3.44.** The functional algorithm for $\mathcal{T}_{\mathcal{ALC}}$ is

```
consistent(S) =
  if $\{c, \overline{c}\} \subseteq S$ then false
  elif $‘\varphi \sqcap \psi' \in S$ and $(‘\varphi' \notin S$ or $‘\psi' \notin S)$
    then consistent($S \cup \{\varphi, \psi\}$)
  elif $‘\varphi \sqcup \psi' \in S$ and $\{\varphi, \psi\} \notin S$
     then consistent($S \cup \{\varphi\}$) or consistent($S \cup \{\psi\}$)
  elif forall $‘\exists \mathrm{R}.\psi' \in S$
   consistent($\{\psi\} \cup \{\theta \in \theta \mid ‘\forall \mathrm{R}.\theta' \in S\}$)
  else true
```

- Relatively simple to implement. (good implementations optimized)
- **But:** This is restricted to $\mathcal{ALC}$. (extension to other DL difficult)

SOME RIGHTS RESERVED

Extending the Tableau Algorithm by Concept Axioms

- concept axioms, e.g. child $\sqsubseteq$ son $\sqcup$ daughter cannot be handled in $\mathcal{T}_{\mathcal{ALC}}$ yet.
- **Idea:** Whenever a new variable y is introduced (by $\mathcal{T}_{\exists}$-rule) add the information that axioms hold for y.
  - Initialize tableau with $\{x{:}\varphi\} \cup \mathcal{CA}$ ($\mathcal{CA}{:} =$ set of concept axioms)
  - New rule for $\exists$: $$\frac{x{:}\exists \mathrm{R}.\varphi \quad \mathcal{CA} = \{\alpha_1, \ldots, \alpha_n\}}{\begin{array}{c} y{:}\varphi \\ x\ \mathrm{R}\ y \\ y{:}\alpha_1 \\ \vdots \\ y{:}\alpha_n \end{array}}\ \mathcal{T}^{\exists}_{\mathcal{CA}}$$ (instead of $\mathcal{T}_{\exists}$)
- **Problem:** $\mathcal{CA} := \{\exists \mathrm{R}.c\}$ and start tableau with $x{:}d$ (non-termination)

SOME RIGHTS RESERVED

Non-Termination of $\mathcal{T}_{\mathcal{ALC}}$ with Concept Axioms

- **Problem:** $\mathcal{CA} := \{\exists \mathsf{R}.c\}$ and start tableau with $x{:}d$. (non-termination)

$x{:}d$	start
$x{:}\exists \mathsf{R}.c$	in $\mathcal{CA}$
$x \ \mathsf{R} \ y_1$	$\mathcal{T}_{\exists}$
$y_1{:}c$	$\mathcal{T}_{\exists}$
$y_1{:}\exists \mathsf{R}.c$	$\mathcal{T}_{\mathcal{CA}}^{\exists}$
$y_1 \ \mathsf{R} \ y_2$	$\mathcal{T}_{\exists}$
$y_2{:}c$	$\mathcal{T}_{\exists}$
$y_2{:}\exists \mathsf{R}.c$	$\mathcal{T}_{\mathcal{CA}}^{\exists}$
...	

Solution: Loop-Check:

- Instead of a new variable y take an old variable z, if we can guarantee that whatever holds for y already holds for z.
- We can only do this, iff the $\mathcal{T}_{\forall}$-rule has been exhaustively applied.

- **Theorem 3.45.** *The consistency problem of $\mathcal{ALC}$ with concept axioms is decidable.*

  Proof sketch: $\mathcal{T}_{\mathcal{ALC}}$ with a suitable loop check terminates.

16.3.3 ABoxes, Instance Testing, and ALC

SOME RIGHTS RESERVED

Instance Test: Concept Membership

- **Definition 3.46.** An instance test computes whether given an $\mathcal{ALC}$ ontology an individual is a member of a given concept.
- **Example 3.47 (An Ontology).**

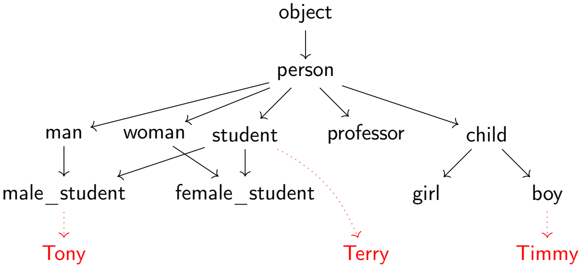
TBox (terminological Box)	ABox (assertional Box, data base)	
$\text{woman} = \text{person} \sqcap \overline{\text{has\_Y}}$	tony:person	Tony is a person
$\text{man} = \text{person} \sqcap \text{has\_Y}$	tony:has_Y	Tony has a y-chrom

This entails: tony:man (Tony is a man).

- **Problem:** Can we compute this?

Realization

- **Definition 3.48.** Realization is the computation of all instance relations between ABox objects and TBox concepts.
- **Observation:** It is sufficient to remember the lowest concepts in the subsumption graph. (rest by subsumption)



- **Example 3.49.** If tony:male_student is known, we do not need tony:man.

ABox Inference in $\mathcal{ALC}$: Phenomena

- There are different kinds of interactions between TBox and ABox in $\mathcal{ALC}$ and in description logics in general.
- **Example 3.50.**

property	example
internally inconsistent	$\text{tony}{:}\text{student}, \text{tony}{:}\overline{\text{student}}$
inconsistent with a TBox	TBox: $\overline{\text{student} \sqcap \text{prof}}$ ABox: $\text{tony}{:}\text{student}, \text{tony}{:}\text{prof}$
implicit info that is not explicit	ABox: $\text{tony}{:}\forall\text{has\_grad}.\text{genius}$ tony has_grad mary $\models \text{mary}{:}\text{genius}$
information that can be combined with TBox info	TBox: $\text{happy\_prof} = \text{prof} \sqcap \forall\text{has\_grad}.\text{genius}$ ABox: $\text{tony}{:}\text{happy\_prof}$, tony has_grad mary $\models \text{mary}{:}\text{genius}$

SOME RIGHTS RESERVED

Tableau-based Instance Test and Realization

- **Query:** Do the ABox and TBox together entail $a{:}\varphi$? ($a \in \varphi$?)
- **Algorithm:** Test $a{:}\overline{\varphi}$ for consistency with ABox and TBox. (use our tableau algorithm)
- **Necessary changes:** (no big deal)
  - Normalize ABox wrt. TBox. (definition expansion)
  - Initialize the tableau with ABox in NNF. (so it can be used)
- **Example 3.51.**

Example: add mary:$\overline{\text{genius}}$ to determine *ABox*, *TBox* $\models$ mary:genius

TBox	happy_prof = prof $\sqcap$ $\forall$has_grad.genius
ABox	tony:happy_prof tony has_grad mary

tony:prof $\sqcap$ $\forall$has_grad.genius	TBox
tony has_grad mary	ABox
mary:$\overline{\text{genius}}$	Query
tony:prof	$\mathcal{T}_{\sqcap}$
tony:$\forall$has_grad.genius	$\mathcal{T}_{\sqcap}$
mary:genius	$\mathcal{T}_{\forall}$
$\bot$	$\mathcal{T}_{\bot}$

- **Note:** The instance test is the base for realization. (remember?)
- **Idea:** Extend to more complex ABox queries. (e.g. give me all instances of φ)

SOME RIGHTS RESERVED

16.4 Description Logics and the Semantic Web

Resource Description Framework

- **Definition 4.1.** The Resource Description Framework (RDF) is a framework for describing resources on the web. It is an XML vocabulary developed by the W3C.
- **Note:** RDF is designed to be read and understood by computers, not to be displayed to people. (it shows)
- **Example 4.2.** RDF can be used for describing (all "objects on the WWW")
  - properties for shopping items, such as price and availability
  - time schedules for web events
  - information about web pages (content, author, created and modified date)
  - content and rating for web pictures
  - content for search engines
  - electronic libraries

SOME RIGHTS RESERVED

Resources and URIs

- RDF describes resources with properties and property values.
- RDF uses Web identifiers (URIs) to identify resources.
- **Definition 4.3.** A resource is anything that can have a URI, such as `http://www.fau.de`.
- **Definition 4.4.** A property is a resource that has a name, such as "*author*" or "*homepage*", and a property value is the value of a property, such as "*Michael Kohlhase*" or `http://kwarc.info/kohlhase`. (a property value can be another resource)
- **Definition 4.5.** A RDF statement s (also known as a triple) consists of a resource (the subject of s), a property (the predicate of s), and a property value (the object of s). A set of RDF triples is called an RDF graph.
- **Example 4.6.** Statements: "*[This slide]*^{subj} *has been [author]*^{pred}*ed by [Michael Kohlhase]*^{obj}"

SOME RIGHTS RESERVED

XML Syntax for RDF

- RDF is a concrete XML vocabulary for writing statements
- **Example 4.7.** The following RDF document could describe the slides as a resource

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
         xmlns:dc= "http://purl.org/dc/elements/1.1/">
  <rdf:Description about="https://.../CompLog/kr/en/rdf.tex">
    <dc:creator>Michael Kohlhase</dc:creator>
    <dc:source>http://www.w3schools.com/rdf</dc:source>
  </rdf:Description>
</rdf:RDF>
```

  This RDF document makes two statements:
  - The subject of both is given in the about attribute of the rdf:Description element
  - The predicates are given by the element names of its children
  - The objects are given in the elements as URIs or literal content.
- **Intuitively:** RDF is a web-scalable way to write down ABox information.

SOME RIGHTS RESERVED

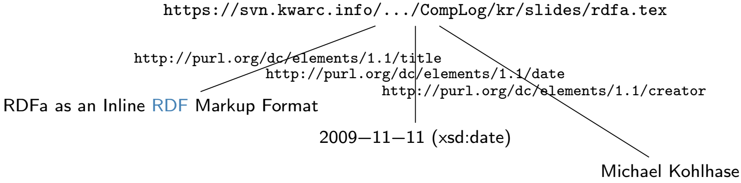
RDFa as an Inline RDF Markup Format

- **Problem:** RDF is a standoff markup format (annotate by URIs pointing into other files)

  Definition 4.8. RDFa (RDF annotations) is a markup scheme for inline annotation (as XML attributes) of RDF triples.

- **Example 4.9.**

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/" id="address">
    <h2 about="#address" property="dc:title">RDF as an Inline RDF Markup Format</h2>
    <h3 about="#address" property="dc:creator">Michael Kohlhase</h3>
    <em about="#address" property="dc:date" datatype="xsd:date"
        content="2009−11−11">November 11., 2009</em>
</div>
```



`https://svn.kwarc.info/.../CompLog/kr/slides/rdfa.tex`

`http://purl.org/dc/elements/1.1/title`

`http://purl.org/dc/elements/1.1/date`

`http://purl.org/dc/elements/1.1/creator`

RDFa as an Inline RDF Markup Format

2009−11−11 (xsd:date)

Michael Kohlhase

RDF as an ABox Language for the Semantic Web

- **Idea:** RDF triples are ABox entries h R s or $h{:}\varphi$.
- **Example 4.10.** h is the resource for Ian Horrocks, s is the resource for Ulrike Sattler, R is the relation “hasColleague”, and φ is the class foaf:Person

```
<rdf:Description about="some.uri/person/ian_horrocks">
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <hasColleague resource="some.uri/person/uli_sattler"/>
</rdf:Description>
```

- **Idea:** Now, we need an similar language for TBoxes (based on $\mathcal{ALC}$)

SOME RIGHTS RESERVED

OWL as an Ontology Language for the Semantic Web

- **Task:** Complement RDF (ABox) with a TBox language.
- **Idea:** Make use of resources that are values in rdf:type. (called Classes)
- **Definition 4.11.** OWL (the ontology web language) is a language for encoding TBox information about RDF classes.
- **Example 4.12 (A concept definition for "Mother").**
  $\text{Mother} = \text{Woman} \sqcap \text{Parent}$ is represented as

XML Syntax

```
<EquivalentClasses>
   <Class IRI="Mother"/>
   <ObjectIntersectionOf>
      <Class IRI="Woman"/>
      <Class IRI="Parent"/>
   </ObjectIntersectionOf>
 </EquivalentClasses>
```

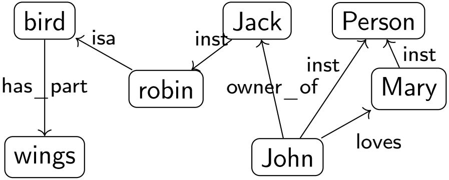
Functional Syntax

```
EquivalentClasses(
   :Mother
   ObjectIntersectionOf(
      :Woman
      :Parent
   )
 )
```

SOME RIGHTS RESERVED

Extended OWL Example in Functional Syntax

- **Example 4.13.** The semantic network from 1.5 can be expressed in OWL (in functional syntax)



  - ClassAssertion formalizes the "inst" relation,
  - ObjectPropertyAssertion formalizes relations,
  - SubClassOf formalizes the "isa" relation,
  - for the "has_part" relation, we have to specify that "*all birds have a part that is a wing*" or equivalently "*the class of birds is a subclass of all objects that have some wing*".

SOME RIGHTS RESERVED

Extended OWL Example in Functional Syntax

- **Example 4.14.** The semantic network from 1.5 can be expressed in OWL (in functional syntax)

```
ClassAssertion (:Jack :robin)
ClassAssertion(:John :person)
ClassAssertion (:Mary :person)
ObjectPropertyAssertion(:loves :John :Mary)
ObjectPropertyAssertion(:owner :John :Jack)
SubClassOf(:robin :bird)
SubClassOf (:bird ObjectSomeValuesFrom(:hasPart :wing))
```

  - ClassAssertion formalizes the “inst” relation,
  - ObjectPropertyAssertion formalizes relations,
  - SubClassOf formalizes the “isa” relation,
  - for the “has_part” relation, we have to specify that “*all birds have a part that is a wing*” or equivalently “*the class of birds is a subclass of all objects that have some wing*”.

SOME RIGHTS RESERVED

SPARQL an RDF Query language

- **Definition 4.15.** SPARQL, the "SPARQL Protocol and RDF Query Language" is an RDF query language, able to retrieve and manipulate data stored in RDF. The SPARQL language was standardized by the World Wide Web Consortium in 2008 [**PruSea08:sparql**].
- SPARQL is pronounced like the word ""*sparkle*"".
- **Definition 4.16.** A system is called a SPARQL endpoint, iff it answers SPARQL queries.
- **Example 4.17.** Query for person names and their e-mails from a triplestore with FOAF data.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

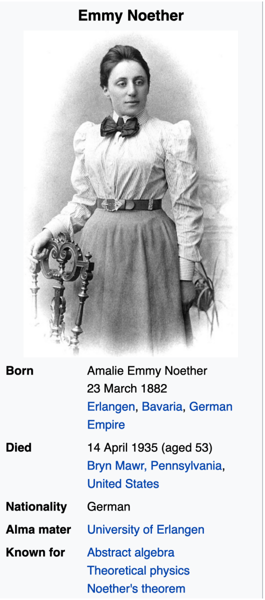
SOME RIGHTS RESERVED

SPARQL Applications: DBPedia

- **Typical Application:** DBPedia screen-scrapes Wikipedia fact boxes for RDF triples and uses SPARQL for querying the induced triplestore.
- **Example 4.18 (DBPedia Query).** People who were born in Erlangen before 1900 (`http://dbpedia.org/snorql`)

```
SELECT ?name ?birth ?death ?person WHERE {
     ?person dbo:birthPlace :Erlangen .
     ?person dbo:birthDate ?birth .
     ?person foaf:name ?name .
     ?person dbo:deathDate ?death .
     FILTER (?birth < "1900—01—01"^^xsd:date) .
}
ORDER BY ?name
```

- The answers include Emmy Noether and Georg Simon Ohm.



Emmy Noether

Born	Amalie Emmy Noether 23 March 1882 Erlangen, Bavaria, German Empire
Died	14 April 1935 (aged 53) Bryn Mawr, Pennsylvania, United States
Nationality	German
Alma mater	University of Erlangen
Known for	Abstract algebra Theoretical physics Noether's theorem

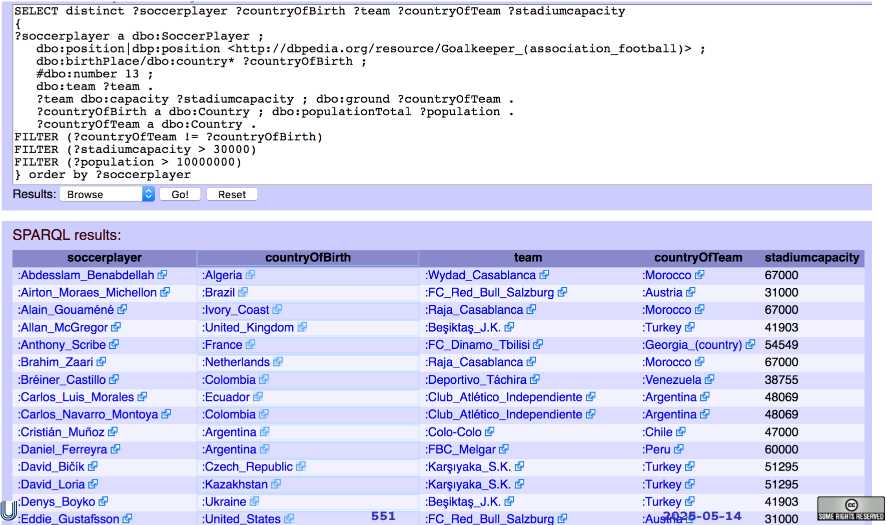
SOME RIGHTS RESERVED

A more complex DBPedia Query

- **Demo:** DBPedia `http://dbpedia.org/snorql/`

  Query: Soccer players born in a country with more than 10 M inhabitants, who play as goalie in a club that has a stadium with more than 30.000 seats.

  Answer: computed by DBPedia from a SPARQL query



```
SELECT distinct ?soccerplayer ?countryOfBirth ?team ?countryOfTeam ?stadiumcapacity
{
?soccerplayer a dbo:SoccerPlayer ;
   dbo:position|dbp:position <http://dbpedia.org/resource/Goalkeeper_(association_football)> ;
   dbo:birthPlace/dbo:country* ?countryOfBirth ;
   #dbo:number 13 ;
   dbo:team ?team .
   ?team dbo:capacity ?stadiumcapacity ; dbo:ground ?countryOfTeam .
   ?countryOfBirth a dbo:Country ; dbo:populationTotal ?population .
   ?countryOfTeam a dbo:Country .
FILTER (?countryOfTeam != ?countryOfBirth)
FILTER (?stadiumcapacity > 30000)
FILTER (?population > 10000000)
} order by ?soccerplayer
```

SPARQL results:

soccerplayer	countryOfBirth	team	countryOfTeam	stadiumcapacity
:Abdesslam_Benabdellah	:Algeria	:Wydad_Casablanca	:Morocco	67000
:Airton_Moraes_Michellon	:Brazil	:FC_Red_Bull_Salzburg	:Austria	31000
:Alain_Gouaméné	:Ivory_Coast	:Raja_Casablanca	:Morocco	67000
:Allan_McGregor	:United_Kingdom	:Beşiktaş_J.K.	:Turkey	41903
:Anthony_Scribe	:France	:FC_Dinamo_Tbilisi	:Georgia_(country)	54549
:Brahim_Zaari	:Netherlands	:Raja_Casablanca	:Morocco	67000
:Bréiner_Castillo	:Colombia	:Deportivo_Táchira	:Venezuela	38755
:Carlos_Luis_Morales	:Ecuador	:Club_Atlético_Independiente	:Argentina	48069
:Carlos_Navarro_Montoya	:Colombia	:Club_Atlético_Independiente	:Argentina	48069
:Cristián_Muñoz	:Argentina	:Colo-Colo	:Chile	47000
:Daniel_Ferreyra	:Argentina	:FBC_Melgar	:Peru	60000
:David_Bičík	:Czech_Republic	:Karşıyaka_S.K.	:Turkey	51295
:David_Loria	:Kazakhstan	:Karşıyaka_S.K.	:Turkey	51295
:Denys_Boyko	:Ukraine	:Beşiktaş_J.K.	:Turkey	41903
:Eddie_Gustafsson	:United_States	:FC_Red_Bull_Salzburg	:Austria	31000

SOME RIGHTS RESERVED

Triple Stores: the Semantic Web Databases

- **Definition 4.19.** A triplestore or RDF store is a purpose-built database for the storage RDF graphs and retrieval of RDF triples usually through variants of SPARQL.
- Common triplestores include
  - Virtuoso: `https://virtuoso.openlinksw.com/` (used in DBpedia)
  - GraphDB: `http://graphdb.ontotext.com/` (often used in WissKI)
  - blazegraph: `https://blazegraph.com/` (open source; used in WikiData)
- **Definition 4.20.** A description logic reasoner implements of reaonsing services based on a satisfiabiltiy test for description logics.
- Common description logic reasoners include
  - FACT++: `http://owl.man.ac.uk/factplusplus/`
  - HermiT: `http://www.hermit-reasoner.com/`
- **Intuition:** Triplestores concentrate on querying very large ABoxes with partial consideration of the TBox, while DL reasoners concentrate on the full set of ontology inference services, but fail on large ABoxes.

SOME RIGHTS RESERVED

Part 4
Planning & Acting

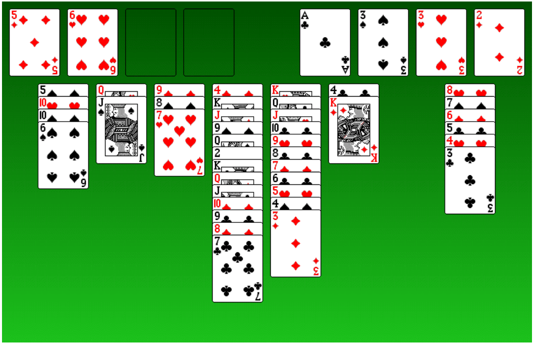
Chapter 17

Planning I: Framework

SOME RIGHTS RESERVED

Reminder: Classical Search Problems

- **Example 0.1 (Solitaire as a Search Problem).**



  - States: Card positions (e.g. position_Jspades=Qhearts).
  - Actions: Card moves (e.g. move_Jspades_Qhearts_freecell4).
  - Initial state: Start configuration.
  - Goal states: All cards “home”.
  - Solutions: Card moves solving this game.

SOME RIGHTS RESERVED

Planning

- **Ambition:** Write one program that can solve all classical search problems.
- **Idea:** For CSP, going from "state/action-level search" to "problem-description level search" did the trick.
- **Definition 0.2.** Let Π be a search problem (see)
  - The blackbox description of Π is an API providing functionality allowing to construct the state space: $\mathrm{InitialState}()$, $\mathrm{GoalTest}(s)$, …
    - "Specifying the problem" $\widehat{=}$ programming the API.
  - The declarative description of Π comes in a problem description language. This allows to implement the API, and much more.
    - "Specifying the problem" $\widehat{=}$ writing a problem description.
- Here, "problem description language" $\widehat{=}$ planning language. (up next)
- **But Wait:** Didn't we do this already in the last chapter with logics? (For the Wumpus?)

SOME RIGHTS RESERVED

17.1 Logic-Based Planning

SOME RIGHTS RESERVED

Fluents: Time-Dependent Knowledge in Planning

- **Recall from :** We can represent the Wumpus rules in logical systems. (propositional/first-order/ALC)
  - Use inference systems to deduce new world knowledge from percepts and actions.
- **Problem:** Representing (changing) percepts immediately leads to contradictions!
- **Example 1.1.** If the agent moves and a cell with a draft at (a perceived breeze) is followed by one without.

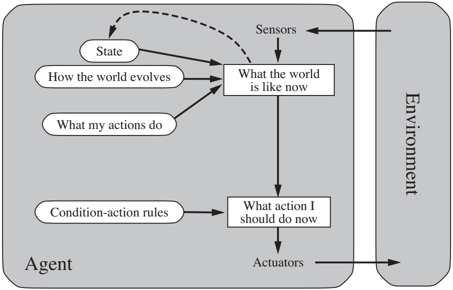
Fluents: Time-Dependent Knowledge in Planning

- **Recall from :** We can represent the Wumpus rules in logical systems. (propositional/first-order/ALC)
  - Use inference systems to deduce new world knowledge from percepts and actions.
- **Problem:** Representing (changing) percepts immediately leads to contradictions!
- **Example 1.4.** If the agent moves and a cell with a draft at (a perceived breeze) is followed by one without.
- **Obvious Idea:** Make representations of percepts time-dependent
- **Example 1.5.** D^t for $t \in \mathbb{N}$ for PL^0 and $\mathrm{draft}(t)$ in PL^1 and $\mathrm{PL}^{\mathrm{nq}}$.
- **Definition 1.6.** We use the word fluent to refer (the representation of) an aspect of the world that changes, all others we call atemporal.

SOME RIGHTS RESERVED

Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percept, and actions.



SOME RIGHTS RESERVED

Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percept, and actions.

```
function KB−AGENT (percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time
  TELL(KB, MAKE−PERCEPT−SENTENCE(percept,t))
  action := ASK(KB, MAKE−ACTION−QUERY(t))
  TELL(KB, MAKE−ACTION−SENTENCE(action,t))
  t := t+1
  return action
```

SOME RIGHTS RESERVED

Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percept, and actions.

```
function KB−AGENT (percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time
  TELL(KB, MAKE−PERCEPT−SENTENCE(percept,t))
  action := ASK(KB, MAKE−ACTION−QUERY(t))
  TELL(KB, MAKE−ACTION−SENTENCE(action,t))
  t := t+1
  return action
```

- **Still Unspecified:** (up next)
  - MAKE−PERCEPT−SENTENCE: the effects of percepts.
  - MAKE−ACTION−QUERY: what is the best next action?
  - MAKE−ACTION−SENTENCE: the effects of that action.

  In particular, we will look at the effect of time/change. (neglected so far)

SOME RIGHTS RESERVED

Fluents: Modeling the Agent's Sensors

- **Definition 1.7.** We call the fluents that describe the agent's percepts (somewhat unimaginatively) percept fluents.
- **Idea:** Relate percept fluents to atemporal cell attributes.
- **Example 1.8.** E.g., if the agent perceives a draft at at time t, when it is in cell $[x, y]$, then there must be a breeze there:

$$\forall t, x, y.\mathrm{Ag@}(t, x, y) \Rightarrow (\mathrm{draft}(t) \Leftrightarrow \mathrm{breeze}(x, y))$$

- Axioms like these model the agent's sensors – here that they are totally reliable: there is a breeze, iff the agent feels a draft at.
- **Definition 1.9.** We call fluents that describe the agent's sensors sensor axioms.
- **Problem:** Where do fluents like $\mathrm{Ag@}(t, x, y)$ come from?

SOME RIGHTS RESERVED

Digression: Fluents and Finite Temporal Domains

- **Observation:** Fluents like $\forall t, x, y.\mathrm{Ag@}(t, x, y) \Rightarrow (\mathrm{draft}(t) \Leftrightarrow \mathrm{breeze}(x, y))$ from 1.8 are best represented in first-order logic. In PL^0 and $\mathrm{PL}^{\mathrm{nq}}$ we would have to use concrete instances like $\mathrm{Ag@}(7, 2, 1) \Rightarrow (\mathrm{draft}(7) \Leftrightarrow \mathrm{breeze}(2, 1))$ for all suitable t, x, and y.
- **Problem:** Unless we restrict ourselves to finite domains and an end time t_{end} we have infinitely many axioms. Even then, formalization in PL^0 and $\mathrm{PL}^{\mathrm{nq}}$ is very tedious.
- **Solution:** Formalize in first-order logic and then compile down:
  1. enumerate ranges of bound variables, instantiate body, $(\rightsquigarrow \mathrm{PL}^{\mathrm{nq}})$
  2. translate $\mathrm{PL}^{\mathrm{nq}}$ atoms to propositional variables. $(\rightsquigarrow \mathrm{PL}^0)$
- **In Practice:** The choice of domain, end time, and logic is up to agent designer, weighing expressivity vs. efficiency of inference.
- WLOG: We will use PL^1 in the following. (easier to read)

Fluents: Effect Axioms for the Transition Model

- **Problem:** Where do fluents like $\mathrm{Ag@}(t,x,y)$ come from?
- **Thus:** We also need fluents to keep track of the agent's actions. (The transition model of the underlying search problem).
- **Idea:** We also use fluents for the representation of actions.
- **Example 1.10.** The action of "going forward" at time t is captured by the fluent $\mathrm{forw}(t)$.
- **Definition 1.11.** Effect axioms describe how the environment changes under an agent's actions.
- **Example 1.12.** If the agent is in cell $[1,1]$ facing east at time 0 and goes forward, she is in cell $[2,1]$ and no longer in $[1,1]$:

  $$\mathrm{Ag@}(0,1,1) \land \mathrm{faceeast}(0) \land \mathrm{forw}(0) \Rightarrow \mathrm{Ag@}(1,2,1) \land \neg \mathrm{Ag@}(1,1,1)$$

  Generally: (barring exceptions for domain border cells)

  $$\forall t,x,y.\mathrm{Ag@}(t,x,y) \land \mathrm{faceeast}(t) \land \mathrm{forw}(t) \Rightarrow \mathrm{Ag@}(t+1,x+1,y) \land \neg \mathrm{Ag@}(t+1,x,y)$$

  This compiles down to $16 \cdot t_{\mathrm{end}}$ $\mathrm{PL^{nq}}/\mathrm{PL^0}$ axioms.

SOME RIGHTS RESERVED

Frames and Frame Axioms

- **Problem:** Effect axioms are not enough.
- **Example 1.13.** Say that the agent has an arrow at time 0, and then moves forward at into $[2,1]$, perceives a glitter, and knows that the Wumpus is ahead. To evaluate the action $\mathrm{shoot}(1)$ the corresponding effect axiom needs to know $\mathrm{havarrow}(1)$, but cannot prove it from $\mathrm{havarrow}(0)$.

  Problem: The information of having an arrow has been lost in the move forward.
- **Definition 1.14.** The frame problem describes that for a representation of actions we need to formalize their effects on the aspects they change, but also their non-effect on the static frame of reference.
- **Partial Solution:** (there are many many more; some better)

  Frame axioms formalize that particular fluents are invariant under a given action.
- **Problem:** For an agent with *n* actions and an environment with *m* fluents, we need $\mathcal{O}(nm)$ frame axioms.

  Representing and reasoning with them easily drowns out the sensor and transition models.

SOME RIGHTS RESERVED

A Hybrid Agent for the Wumpus World

- **Example 1.15 (A Hybrid Agent).** This agent uses
  - logic inference for sensor and transition modeling,
  - special code and A^* for action selection & route planning.

```
function HYBRID−WUMPUS−AGENT(percept) returns an action
   inputs: percept, a list, [stench,breeze,glitter,bump,scream]
   persistent: KB, a knowledge base, initially the atemporal
                     "wumpus physics"
               t, a counter, initially 0, indicating time
               plan, an action sequence, initially empty
  TELL(KB, MAKE−PERCEPT−SENTENCE(percept,t))
```

then some special code for action selection, and then (up next)

```
  action := POP(plan)
  TELL(KB, MAKE−ACTION−SENTENCE(action,t))
  $t := t + 1$
  return action
```

So far, not much new over our original version.

SOME RIGHTS RESERVED

A Hybrid Agent: Custom Action Selection

- **Example 1.16 (A Hybrid Agent (continued)).** So that we can plan the best strategy:

```
TELL($KB$, the temporal "physics" sentences for time $t$)
$safe := \{[x,y] \mid \mathrm{ASK}(KB,\mathrm{OK}(t,x,y))=\mathrm{T}\}$
if ASK($KB$,$\mathrm{glitter}(t)$) = T then
     $plan := [\mathrm{grab}] + \mathrm{PLAN{-}ROUTE}(current,\{[1,1]\},safe) + [\mathrm{exit}]$
if $plan$ is empty then
    $unvisited := \{[x,y] \mid \mathrm{ASK}(KB,\mathrm{Ag@}(t',x,y))=\mathrm{F}\}$ for all $t' \leq t$
    $plan := \mathrm{PLAN{-}ROUTE}(current,unvisited \cup safe,safe)$
if $plan$ is empty and ASK($KB$,$\mathrm{havarrow}(t)$) = T then
    $possible\_wumpus := \{x,y \mid [x,y]\}\mathrm{ASK}(KB,\neg\mathrm{wumpus}(t,x,y)) = \mathrm{F}$
    $plan := \mathrm{PLAN{-}SHOT}(current,possible\_wumpus,safe)$
if $plan$ is empty then // no choice but to take a risk
    $not\_unsafe := \{[x,y] \mid \mathrm{ASK}(KB,\neg\mathrm{OK}(t,x,y)) = \mathrm{F}\}$
    $plan := \mathrm{PLAN{-}ROUTE}(current,unvisited \cup not\_unsafe,safe)$
if $plan$ is empty then
    $plan := \mathrm{PLAN{-}ROUTE}(current,\{[1,1]\},safe) + [\mathrm{exit}]$
```

Note that OK wumpus, and glitter are fluents, since the Wumpus might have died or the gold might have been grabbed.

SOME RIGHTS RESERVED

A Hybrid Agent: Custom Action Selection

- **Example 1.17 (Action Selection).** And the code for PLAN−ROUTE (PLAN−SHOT similar)

```
function PLAN−ROUTE(curr,goals,allowed) returns an action sequence
  inputs: curr, the agent's current position
          goals, a set of squares;
                 try to plan a route to one of them
          allowed, a set of squares that can form part of the route
  problem := ROUTE−PROBLEM(curr,goals,allowed)
  return $A^*$(problem)
```

- **Evaluation:** Even though this works for the Wumpus world, it is not the "universal, logic-based problem solver" we dreamed of!
- Planning tries to solve this with another representation of actions. (up next)

SOME RIGHTS RESERVED

17.2 Planning: Introduction

SOME RIGHTS RESERVED

How does a planning language describe a problem?

- **Definition 2.1.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)

How does a planning language describe a problem?

- **Definition 2.3.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
  - initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)

SOME RIGHTS RESERVED

How does a planning language describe a problem?

- **Definition 2.5.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
  - initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)
  - goal states G (vs. a goal test). (E.g.: $Eq(x,2)$.)

SOME RIGHTS RESERVED

How does a planning language describe a problem?

- **Definition 2.7.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
  - initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)
  - goal states G (vs. a goal test). (E.g.: $Eq(x,2)$.)
  - set A of actions in terms of preconditions and effects (vs. functions returning applicable actions and successor states). (E.g.: "increment x: pre $Eq(x,1)$, iff $Eq(x \wedge 2) \wedge \neg Eq(x,1)$".)

  A logical description of all of these is called a planning task.

SOME RIGHTS RESERVED

How does a planning language describe a problem?

- **Definition 2.9.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
  - initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)
  - goal states G (vs. a goal test). (E.g.: $Eq(x,2)$.)
  - set A of actions in terms of preconditions and effects (vs. functions returning applicable actions and successor states). (E.g.: "increment x: pre $Eq(x,1)$, iff $Eq(x \wedge 2) \wedge \neg Eq(x,1)$".)

  A logical description of all of these is called a planning task.
- **Definition 2.10.** Solution (plan) $\widehat{=}$ sequence of actions from $\mathcal{A}$, transforming $\mathcal{I}$ into a state that satisfies $\mathcal{G}$. (E.g.: "increment x".)
  The process of finding a plan given a planning task is called planning.

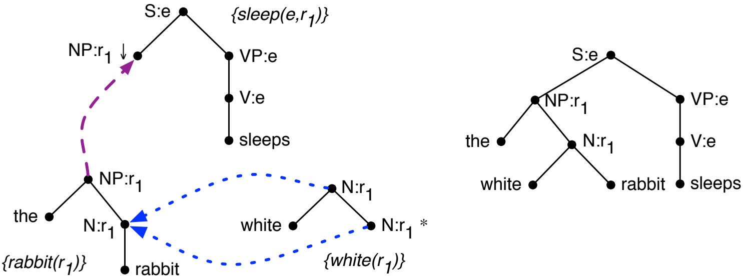
SOME RIGHTS RESERVED

Planning Language Overview

- **Disclaimer:** Planning languages go way beyond classical search problems. There are variants for inaccessible, stochastic, dynamic, continuous, and multi-agent settings.
- We focus on classical search for simplicity (and practical relevance).
- For a comprehensive overview, see [**ghallab:etal:04**].

SOME RIGHTS RESERVED

Application: Natural Language Generation

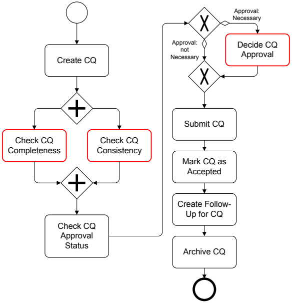


- **Input**: Tree-adjoining grammar, intended meaning.
- **Output**: Sentence expressing that meaning.

SOME RIGHTS RESERVED

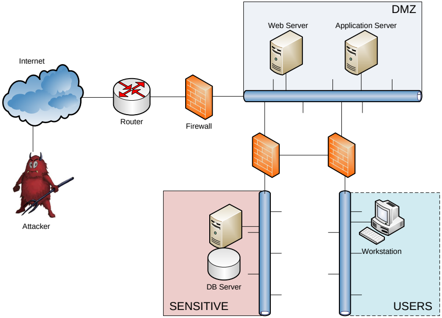
Application: Business Process Templates at SAP

Action name	precondition	effect
Check CQ Completeness	CQ.archiving:notArchived	CQ.completeness:complete OR CQ.completeness:notComplete
Check CQ Consistency	CQ.archiving:notArchived	CQ.consistency:consistent OR CQ.consistency:notConsistent
Check CQ Approval Status	CQ.archiving:notArchived AND CQ.approval:notChecked AND CQ.completeness:complete AND CQ.consistency:consistent	CQ.approval:necessary OR CQ.approval:notNecessary
Decide CQ Approval	CQ.archiving:notArchived AND CQ.approval:necessary	CQ.approval:granted OR CQ.approval:notGranted
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Create Follow-Up for CQ	CQ.archiving:notArchived AND CQ.acceptance:accepted	CQ.followUp:documentCreated
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived



- **Input**: model of behavior of activities on business objects, process endpoint.
- **Output**: Process template leading to this point.

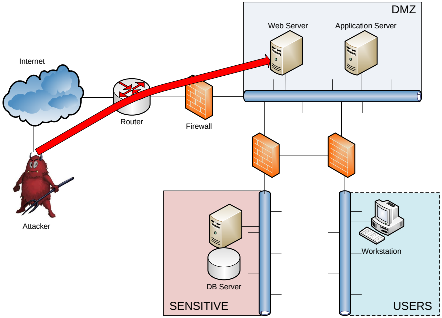
Application: Automatic Hacking



- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

SOME RIGHTS RESERVED

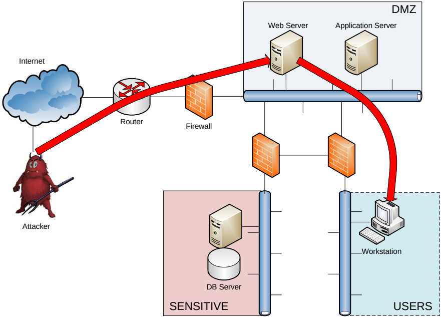
Application: Automatic Hacking



- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

SOME RIGHTS RESERVED

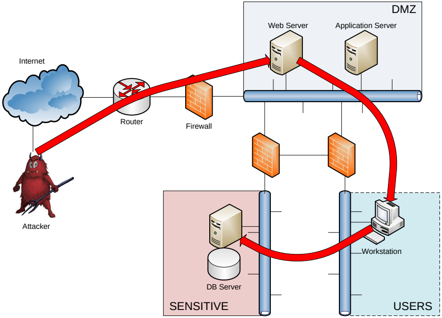
Application: Automatic Hacking



- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

SOME RIGHTS RESERVED

Application: Automatic Hacking



- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

SOME RIGHTS RESERVED

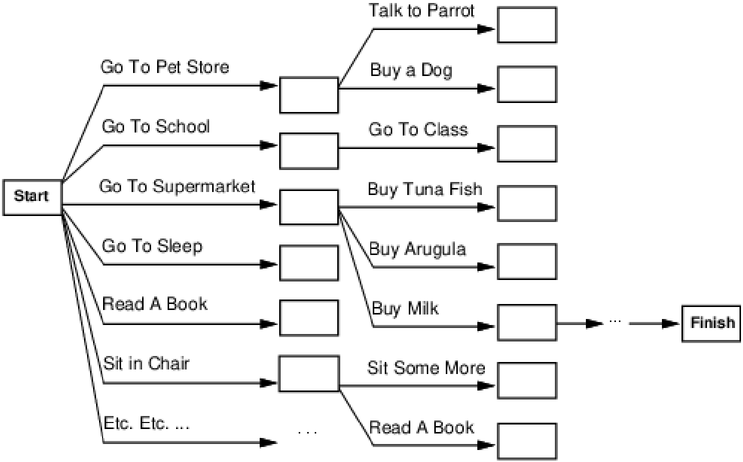
Reminder: General Problem Solving, Pros and Cons

- **Powerful:** In some applications, generality is absolutely necessary. (E.g. SAP)
- **Quick:** Rapid prototyping: 10s lines of problem description vs. 1000s lines of C++ code. (E.g. language generation)
- **Flexible:** Adapt/maintain *the description*. (E.g. network security)
- **Intelligent:** Determines automatically how to solve a complex problem efficiently! (The ultimate goal, no?!)
- **Efficiency loss:** Without any domain-specific knowledge about chess, you don't beat Kasparov . . .
  - Trade-off between "automatic and general" vs. "manual work but efficient".
- **Research Question:** How to make fully automatic algorithms efficient?

SOME RIGHTS RESERVED

Search vs. planning

- Consider the task "*get milk, bananas, and a cordless drill*".
- Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate

SOME RIGHTS RESERVED

Search vs. planning (cont.)

- Planning systems do the following:
  1. open up action and goal representation to allow selection
  2. divide-and-conquer by subgoaling
- relax requirement for sequential construction of solutions

	Search	**Planning**
States	`Lisp data structures`	Logical sentences
Actions	`Lisp` code	Preconditions/outcomes
Goal	`Lisp` code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

SOME RIGHTS RESERVED

Reminder: Greedy Best-First Search and A^*

- **Recall:** Our heuristic search algorithms (duplicate pruning omitted for simplicity)

```
function Greedy_Best−First_Search (problem)
  returns a solution, or failure
  $n$ := node with $n$.state=problem.InitialState
  frontier := priority queue ordered by ascending $h$, initially [$n$]
  loop do
    if Empty?(frontier) then return failure
    $n$ := Pop(frontier)
    if problem.GoalTest($n$.state) then return Solution($n$)
    for each action $a$ in problem.Actions($n$.state) do
      $n'$ := ChildNode(problem,$n$,$a$)
      Insert($n'$, $h(n')$, frontier)
```

For A^*

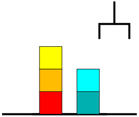
- order *frontier* by $g+h$ instead of h (line 4)
- insert $g(n')+h(n')$ instead of $h(n')$ to *frontier* (last line)

- Is greedy best-first search optimal? No ⤳ satisficing planning.
- Is A^* optimal? Yes, but only if h is admissible ⤳ optimal planning, **with such** h.

SOME RIGHTS RESERVED

ps. "Making Fully Automatic Algorithms Efficient"

- **Example 2.11.**



- *n* blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

- **Observation 2.12.** *State spaces typically are huge even for simple problems.*
- **In other words:** Even solving "simple problems" automatically (without help from a human) requires a form of intelligence.
- With blind search, even the largest super computer in the world won't scale beyond 20 blocks!

SOME RIGHTS RESERVED

Algorithmic Problems in Planning

- **Definition 2.13.** We speak of satisficing planning if
  Input: A planning task Π.
  Output: A plan for Π, or "unsolvable" if no plan for Π exists.
  and of optimal planning if
  Input: A planning task Π.
  Output: An optimal plan for Π, or "unsolvable" if no plan for Π exists.
- The techniques successful for either one of these are almost disjoint. And satisficing planning is *much* more efficient in practice.
- **Definition 2.14.** Programs solving these problems are called (optimal) planner, planning system, or planning tool.

SOME RIGHTS RESERVED

Our Agenda for This Topic

- **Now:** Background, planning languages, complexity.
  - Sets up the framework. Computational complexity is essential to distinguish different algorithmic problems, and for the design of heuristic functions. (see next)
- **Next:** How to automatically generate a heuristic function, given planning language input?
  - Focussing on heuristic search as the solution method, this is the main question that needs to be answered.

SOME RIGHTS RESERVED

Our Agenda for This Chapter

1. **The History of Planning**: How did this come about?
   - Gives you some background, and motivates our choice to focus on heuristic search.
2. **The STRIPS Planning Formalism**: Which concrete planning formalism will we be using?
   - Lays the framework we'll be looking at.
3. **The PDDL Language**: What do the input files for off-the-shelf planning software look like?
   - So you can actually play around with such software. (Exercises!)
4. **Planning Complexity**: How complex is planning?
   - The price of generality is complexity, and here's what that "price" is, exactly.

SOME RIGHTS RESERVED

17.3 The History of Planning

SOME RIGHTS RESERVED

Planning History: In the Beginning . . .

- **In the beginning:** **Man invented Robots:**
  - "Planning" as in "the making of plans by an autonomous robot".
  - Shakey the Robot (Full video here)
- **In a little more detail:**
  - [**newell:simon:ct-63**] introduced *general problem solving*.
  - *. . . not much happened (well not much we still speak of today) . . .*
  - 1966-72, Stanford Research Institute developed a robot named "Shakey".
  - They needed a "planning" component taking decisions.
  - They took inspiration from general problem solving and theorem proving, and called the resulting algorithm STRIPS.

SOME RIGHTS RESERVED

History of Planning Algorithms

- **Compilation into Logics/Theorem Proving:**
  - e.g. $\exists s_0, a, s_1 . at(A, s_0) \wedge execute(s_0, a, s_1) \wedge at(B, s_1)$
  - **Popular when**: Stone Age – 1990.
  - **Approach**: *From planning task description, generate PL1 formula φ that is satisfiable iff there exists a plan; use a theorem prover on φ.*
  - **Keywords/cites**: Situation calculus, frame problem, . . .
- Partial order planning
  - e.g. $open = \{at(B)\}$; apply $move(A, B)$; $\rightsquigarrow$ $open = \{at(A)\}$ . . .
  - **Popular when**: 1990 – 1995.
  - **Approach**: *Starting at goal, extend partially ordered set of searchprob/actions by inserting achievers for open sub-goals, or by adding ordering constraints to avoid conflicts.*
  - **Keywords/cites**: **UCPOP** [**penberthy:weld:kr-92**], causal links, flaw selection strategies, . . .

SOME RIGHTS RESERVED

History of Planning Algorithms, ctd.

- GraphPlan
  - e.g. $F_0 = at(A); A_0 = \{move(A, B)\}; F_1 = \{at(B)\}$; mutex $A_0 = \{move(A, B), move(A, C)\}$.
  - **Popular when**: 1995 – 2000.
  - **Approach**: *In a forward phase, build a layered "planning graph" whose "time steps" capture which pairs of action can achieve which pairs of facts; in a backward phase, search this graph starting at goals and excluding options proved to not be feasible.*
  - **Keywords/cites**: [**blum:furst:ijcai-95**; **blum:furst:ai-97**; **koehler:etal:ecp-97**], action/fact mutexes, step-optimal plan, . . .
- **Planning as SAT:**
  - SAT variables $at(A)_0$, $at(B)_0$, $move(A, B)_0$, $move(A, C)_0$, $at(A)_1$, $at(B)_1$; clauses to encode transition behavior e.g. $at(B)_1{}^F \vee move(A, B)_0{}^T$; unit clauses to encode initial state $at(A)_0{}^T$, $at(B)_0{}^T$; unit clauses to encode goal $at(B)_1{}^T$.
  - **Popular when**: 1996 – today.
  - **Approach**: *From planning task description, generate propositional CNF formula φ_k that is satisfiable iff there exists a plan with k steps; use a SAT solver on φ_k, for different values of k.*
  - **Keywords/cites**: [**kautz:selman:ecai-92**; **kautz:selman:aaai-96**; **rintanen:etal:ai-06**; **rintanen:cp-10**], SAT encoding schemes, BlackBox, . . .

SOME RIGHTS RESERVED

History of Planning Algorithms, ctd.

- **Planning as Heuristic Search:**
  - init $at(A)$; apply $move(A,B)$; generates state $at(B)$; ...
  - **Popular when**: 1999 – today.
  - **Approach**: Devise a method $\mathcal{R}$ to simplify ("relax") any planning task Π; given Π, solve $\mathcal{R}(\Pi)$ to generate a heuristic function h for informed search.
  - **Keywords/cites**: [**bonet:geffner:ecp-99**; **haslum:geffner:aips-00**; **bonet:geffner:ai-01**; **hoffmann:nebel:jair-01**; **edelkamp:ecp-01**; **gerevini:etal:jair-03**; **helmert:jair-06**; **helmert:etal:icaps-07**; **helmert:geffner:icaps-08**; **karpas:domshlak:ijcai-09**; **helmert:domshlak:icaps-09**; **richter:westphal:jair-10**; **nissim:etal:ijcai-11**; **katz:etal:icaps-12**; **keyder:etal:icaps-12**; **katz:etal:icaps-13**; **domshlak:etal:ai-15**], critical path heuristics, ignoring delete lists, relaxed plans, landmark heuristics, abstractions, partial delete relaxation, ...

SOME RIGHTS RESERVED

The International Planning Competition (IPC)

- **Definition 3.1.** The International Planning Competition (IPC) is an event for benchmarking planners (`http://ipc.icapsconference.org/`)
  - **How**: Run competing planners on a set of benchmarks.
  - **When**: Runs every two years since 2000, annually since 2014.
  - **What**: Optimal track vs. satisficing track; others: uncertainty, learning, . . .
- **Prerequisite/Result:**
  - Standard representation language: PDDL [**pddl-handbook**; **fox:long:jair-03**; **hoffmann:edelkamp:jair-05**; **gerevini:etal:ai-09**]
  - Problem Corpus: ≈ 50 **domains**, $\gg 1000$ **instances**, 74 (!!) planners in 2011

SOME RIGHTS RESERVED

International Planning Competition

- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?

SOME RIGHTS RESERVED

International Planning Competition

- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?
- **Answer:** Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a "winner"! On other domains and/or according to other criteria, you may well be better off with the "looser".

SOME RIGHTS RESERVED

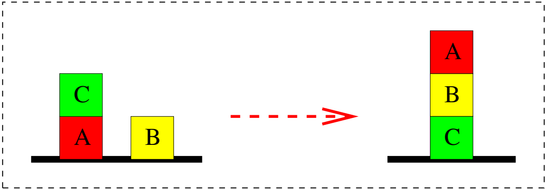
International Planning Competition

- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?
- **Answer:** Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a "winner"! On other domains and/or according to other criteria, you may well be better off with the "looser".
- **Generally:** Assessing AI System suitability is complicated, over-simplification is dangerous. (But, of course, nevertheless is being done all the time)

SOME RIGHTS RESERVED

Planning History, p.s.: Planning is Non-Trivial!

- **Example 3.2.** The Sussman anomaly is a simple blocksworld planning problem:

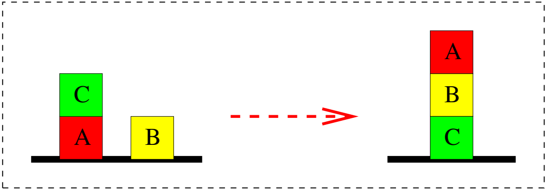


Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

SOME RIGHTS RESERVED

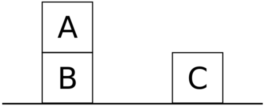
Planning History, p.s.: Planning is Non-Trivial!

- **Example 3.3.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A,B)$ and $\mathrm{on}(B,C)$ fail:

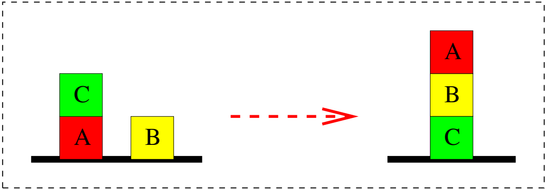
- If we pursue $\mathrm{on}(A,B)$ by unstacking C, and moving A onto B, we achieve the first subgoal, but cannot achieve the second without undoing the first.



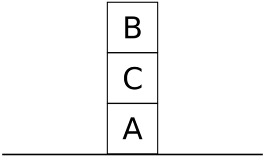
SOME RIGHTS RESERVED

Planning History, p.s.: Planning is Non-Trivial!

- **Example 3.4.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\text{on}(A,B)$ and $\text{on}(B,C)$ fail:



- If we pursue $\text{on}(A,B)$ by unstacking C, and moving A onto B, we achieve the first subgoal, but cannot achieve the second without undoing the first.
- If we pursue $\text{on}(B,C)$ by moving B onto C, we achieve the second subgoal, but cannot achieve the first without undoing the second.

17.4 The STRIPS Planning Formalism

STRIPS Planning

- **Definition 4.1.** STRIPS = Stanford Research Institute Problem Solver.
  STRIPS is the simplest possible (reasonably expressive) logics based planning language.
- STRIPS has only propositional variables as atomic formulae.
- Its preconditions/effects/searchprob/goal states are as canonical as imaginable:
  - Preconditions, searchprob/goal states: conjunctions of atoms.
  - Effects: conjunctions of literals
- We use the common special-case notation for this simple formalism.
- I'll outline some extensions beyond STRIPS later on, when we discuss PDDL.
- **Historical note:** STRIPS [**fikes:nilsson:ai-71**] was originally a planner (cf. Shakey), whose language actually wasn't quite that simple.

SOME RIGHTS RESERVED

STRIPS Planning: Syntax

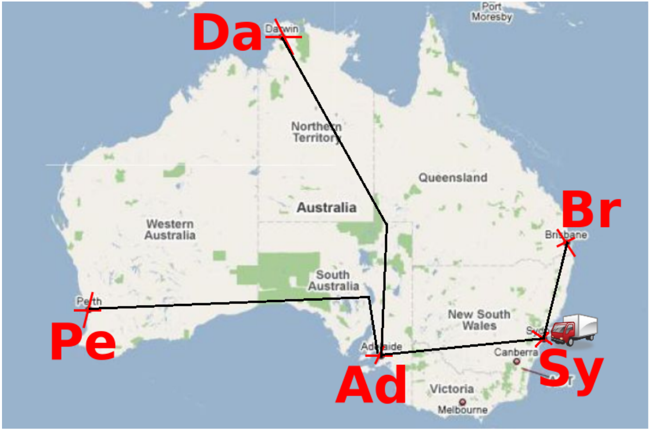
- **Definition 4.2.** A STRIPS task is a search problem $\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, where
  1. the states in $\mathcal{S}$ are sets of facts, i.e. atomic proposition in PL^0.
  2. the actions $a\in\mathcal{A}$ are triples $a=\langle\mathrm{pre}_a,\mathrm{add}_a,\mathrm{del}_a\rangle$ of subsets of $\mathcal{S}$. The components are referred to as the action's preconditions, add list, and delete list respectively; we require that $\mathrm{add}_a\cap\mathrm{del}_a=\emptyset$.
  3. The transition model $\mathcal{T}$ is given by $\mathcal{T}(a,s){:=}(s\cup\mathrm{add}_a)\backslash\mathrm{del}_a$, iff $\mathrm{pre}_a\subseteq s$, otherwise $\mathcal{T}(a,s)$ is undefined.

  A solution to a STRIPS task is called a plan.
- **Note:** As $\mathcal{I},\mathcal{G}\subseteq\mathcal{S}$ they are also sets of atoms.
- **Idea:** An searchprob/action $a\in\mathcal{A}$ is applicable in a searchprob/state s, if all preconditions are met ($\mathrm{pre}_a\subseteq s$). The result is s minius del_a plus add_a.
- **Remark:** Instead of PL^0, we can also use $\mathrm{PL}^{\mathrm{nq}}$. (more practical)
- **Note:** We assume, for simplicity, that every searchprob/action has cost 1. (Unit costs, cf.)

SOME RIGHTS RESERVED

“TSP” in Australia

- **Example 4.3 (Salesman Travelling in Australia).**

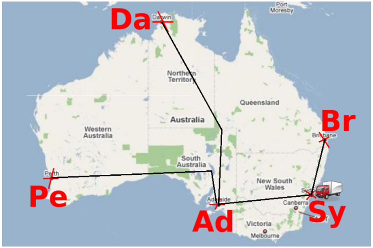


Strictly speaking, this is not actually a TSP problem instance; simplified/adapted for illustration.

SOME RIGHTS RESERVED

STRIPS Encoding of a "TSP"

- **Example 4.4 (continuing).**

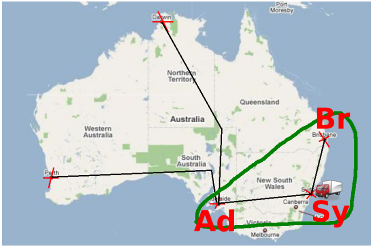


- Facts P: $\{\mathrm{at}(x), \mathrm{vis}(x) \,|\, x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}, \mathrm{Pe}, \mathrm{Da}\}\}$.
- Searchprob/initial state I: $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
- Searchprob/goal state G:$\{\mathrm{at}(\mathrm{Sy})\} \cup \{\mathrm{vis}(x) \,|\, x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}, \mathrm{Pe}, \mathrm{Da}\}\}$.
- Searchprob/actions $a \in A$: $\mathrm{drv}(x, y)$ where x and y have a road.
  Preconditions pre_a: $\{\mathrm{at}(x)\}$.
  Add list add_a: $\{\mathrm{at}(y), \mathrm{vis}(y)\}$.
  Delete list del_a: $\{\mathrm{at}(x)\}$.
- Plan: $\langle \mathrm{drv}(\mathrm{Sy}, \mathrm{Br}), \mathrm{drv}(\mathrm{Br}, \mathrm{Sy}), \mathrm{drv}(\mathrm{Sy}, \mathrm{Ad}), \mathrm{drv}(\mathrm{Ad}, \mathrm{Pe}), \mathrm{drv}(\mathrm{Pe}, \mathrm{Ad}), \ldots$
  $\ldots, \mathrm{drv}(\mathrm{Ad}, \mathrm{Da}), \mathrm{drv}(\mathrm{Da}, \mathrm{Ad}), \mathrm{drv}(\mathrm{Ad}, \mathrm{Sy})\rangle$

SOME RIGHTS RESERVED

STRIPS Encoding of Simplified TSP

- **Example 4.5 (Simplified Traveling Salesman Problem in Australia).**

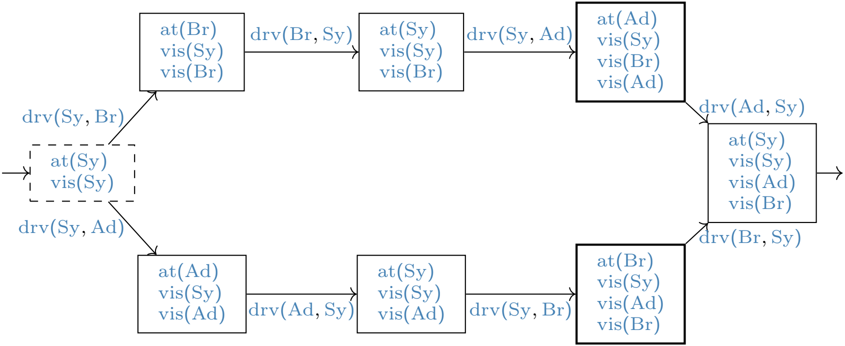


Let TSP_{-} be the STRIPS task, $\langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply} \rangle$, where

- Facts $\mathcal{F}$: $\{\mathrm{at}(x), \mathrm{vis}(x) \mid x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}\}\}$.
- Searchprob/initial state searchprob/state $\mathcal{I}$: $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
- Searchprob/goal state $\mathcal{G}$: $\{\mathrm{vis}(x) \mid x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}\}\}$ (note: $\mathrm{noat}(\mathrm{Sy})$)
- Searchprob/actions $\mathcal{A}$: $a \in A$: $\mathrm{drv}(x, y)$ where x y have a road.
  - preconditions pre_a: $\{\mathrm{at}(x)\}$.
  - add list add_a: $\{\mathrm{at}(y), \mathrm{vis}(y)\}$.
  - delete list del_a: $\{\mathrm{at}(x)\}$.

Questionaire: State Space of TSP_

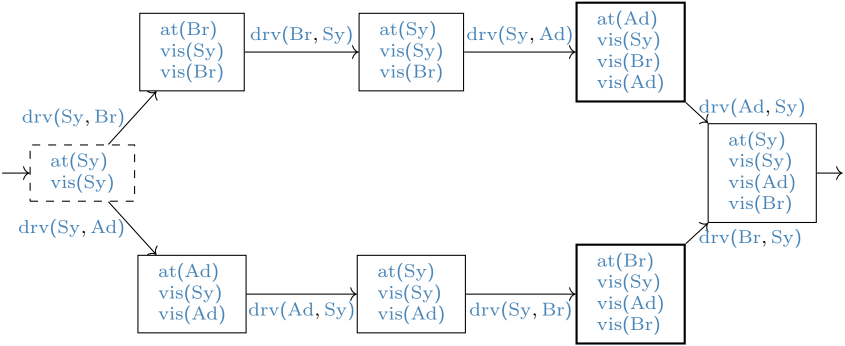
- The state space of TSP_ from 4.5 is



- **Question:** Are there any plans for TSP_ in this graph?

Questionaire: State Space of TSP_{-}

- The state space of TSP_{-} from 4.5 is

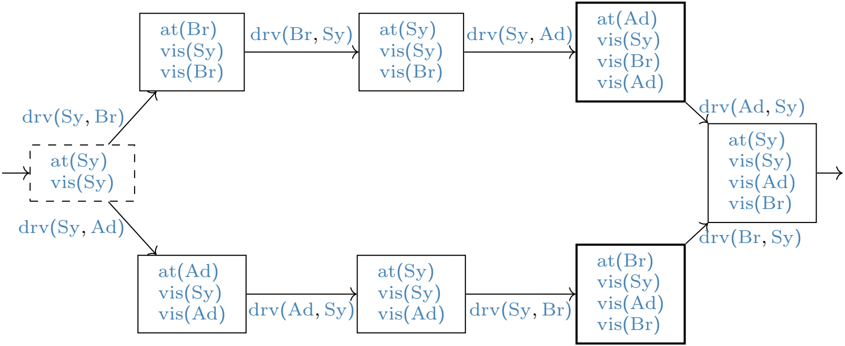


- **Question:** Are there any plans for TSP_{-} in this graph?
- **Answer:** Yes, two – dashed node $\widehat{=} \mathcal{I}$, thick nodes $\widehat{=} \mathcal{G}$:
  - $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br}), \mathrm{drv}(\mathrm{Br}, \mathrm{Sy}), \mathrm{drv}(\mathrm{Sy}, \mathrm{Ad})$ (upper path)
  - $\mathrm{drv}(\mathrm{Sy}, \mathrm{Ad}), \mathrm{drv}(\mathrm{Ad}, \mathrm{Sy}), \mathrm{drv}(\mathrm{Sy}, \mathrm{Br})$. (lower path)

SOME RIGHTS RESERVED

Questionaire: State Space of TSP_-

- The state space of TSP_- from 4.5 is

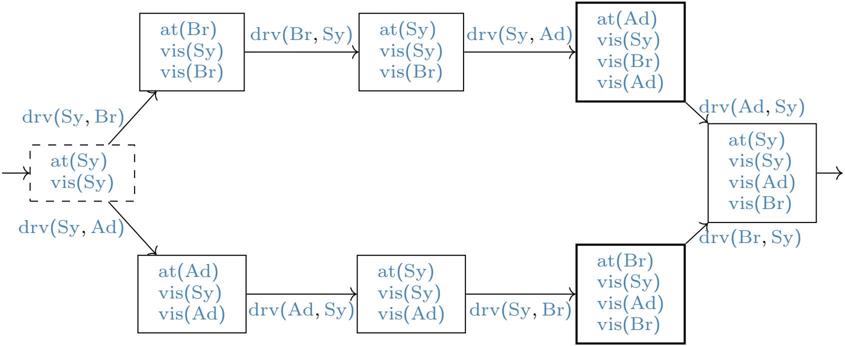


- **Question:** Are there any plans for TSP_- in this graph?
- **Answer:** Yes, two – dashed node $\widehat{=}\ \mathcal{I}$, thick nodes $\widehat{=}\ \mathcal{G}$:
  - $\mathrm{drv}(\mathrm{Sy},\mathrm{Br}),\mathrm{drv}(\mathrm{Br},\mathrm{Sy}),\mathrm{drv}(\mathrm{Sy},\mathrm{Ad})$ (upper path)
  - $\mathrm{drv}(\mathrm{Sy},\mathrm{Ad}),\mathrm{drv}(\mathrm{Ad},\mathrm{Sy}),\mathrm{drv}(\mathrm{Sy},\mathrm{Br})$. (lower path)
- **Question:** Is the graph above actually the state space induced by TSP_-?

SOME RIGHTS RESERVED

Questionaire: State Space of $\mathrm{TSP}_{\_}$

- The state space of $\mathrm{TSP}_{\_}$ from 4.5 is

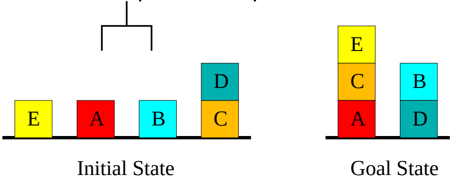


- **Question:** Are there any plans for $\mathrm{TSP}_{\_}$ in this graph?
- **Answer:** Yes, two – dashed node $\widehat{=}\ \mathcal{I}$, thick nodes $\widehat{=}\ \mathcal{G}$:
  - $\mathrm{drv}(\mathrm{Sy},\mathrm{Br}),\mathrm{drv}(\mathrm{Br},\mathrm{Sy}),\mathrm{drv}(\mathrm{Sy},\mathrm{Ad})$ (upper path)
  - $\mathrm{drv}(\mathrm{Sy},\mathrm{Ad}),\mathrm{drv}(\mathrm{Ad},\mathrm{Sy}),\mathrm{drv}(\mathrm{Sy},\mathrm{Br})$. (lower path)
- **Question:** Is the graph above actually the state space induced by $\mathrm{TSP}_{\_}$?
- **Answer:** No, only the part reachable from $\mathcal{I}$. The state space of $\mathrm{TSP}_{\_}$ also includes e.g. the searchprob/states $\{\mathrm{vis}(\mathrm{Sy})\}$ and $\{\mathrm{at}(\mathrm{Sy}),\mathrm{at}(\mathrm{Br})\}$.

SOME RIGHTS RESERVED

The Blocksworld

- **Definition 4.6.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.7.**

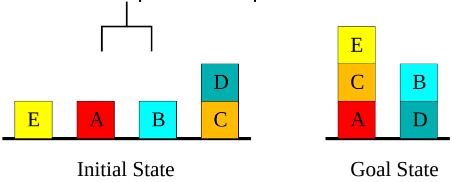


  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.

SOME RIGHTS RESERVED

The Blocksworld

- **Definition 4.8.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.9.**

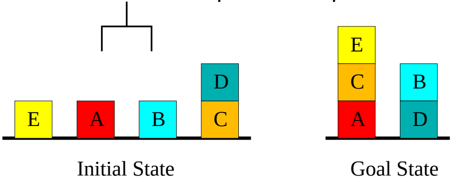


  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - searchprob/initial state: $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D,C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.

SOME RIGHTS RESERVED

The Blocksworld

- **Definition 4.10.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.11.**

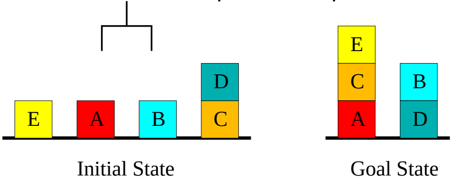


  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - searchprob/initial state: $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D,C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.
  - Searchprob/goal state: $\{\mathrm{on}(E,C), \mathrm{on}(C,A), \mathrm{on}(B,D)\}$.

SOME RIGHTS RESERVED

The Blocksworld

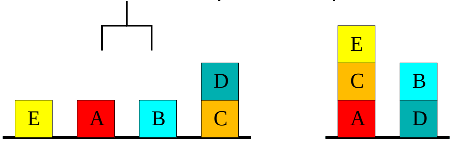
- **Definition 4.12.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.13.**



  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - searchprob/initial state: $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D,C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.
  - Searchprob/goal state: $\{\mathrm{on}(E,C), \mathrm{on}(C,A), \mathrm{on}(B,D)\}$.
  - Searchprob/actions: $\mathrm{stack}(x,y)$, $\mathrm{unstack}(x,y)$, $\mathrm{putdown}(x)$, $\mathrm{pickup}(x)$.

The Blocksworld

- **Definition 4.14.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.15.**



Initial State

Goal State

  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - searchprob/initial state:
    $\{\mathrm{onTable}(E),\mathrm{clear}(E),\ldots,\mathrm{onTable}(C),\mathrm{on}(D,C),\mathrm{clear}(D),\mathrm{armEmpty}\}$.
  - Searchprob/goal state: $\{\mathrm{on}(E,C),\mathrm{on}(C,A),\mathrm{on}(B,D)\}$.
  - Searchprob/actions: $\mathrm{stack}(x,y)$, $\mathrm{unstack}(x,y)$, $\mathrm{putdown}(x)$, $\mathrm{pickup}(x)$.
  - $\mathrm{stack}(x,y)$?
    $\mathrm{pre}:\{\mathrm{holding}(x),\mathrm{clear}(y)\}$
    $\mathrm{add}:\{\mathrm{on}(x,y),\mathrm{armEmpty},\mathit{clearx}\}$
    $\mathrm{del}:\{\mathrm{holding}(x),\mathrm{clear}(y)\}$.

SOME RIGHTS RESERVED

STRIPS for the Blocksworld

- **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\mathrm{pickup}(x)$ action schema?

  (A) $\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
  $\{\mathrm{holding}(x)\}$
  $\{\mathrm{onTable}(x)\}$

  (B) $\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
  $\{\mathrm{holding}(x)\}$
  $\{\mathrm{armEmpty}\}$

  (C) $\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
  $\{\mathrm{holding}(x)\}$
  $\{\mathrm{onTable}(x), \mathrm{armEmpty}, \mathrm{clear}(x)\}$

  (D) $\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
  $\{\mathrm{holding}(x)\}$
  $\{\mathrm{onTable}(x), \mathrm{armEmpty}\}$

  Recall: an actions *a* represented by a tuple $\langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of lists of facts.
- **Hint:** The only differences between them are the delete lists

SOME RIGHTS RESERVED

STRIPS for the Blocksworld

- **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\mathrm{pickup}(x)$ action schema?

(A)	$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$ $\{\mathrm{holding}(x)\}$ $\{\mathrm{onTable}(x)\}$	(B)	$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$ $\{\mathrm{holding}(x)\}$ $\{\mathrm{armEmpty}\}$
(C)	$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$ $\{\mathrm{holding}(x)\}$ $\{\mathrm{onTable}(x), \mathrm{armEmpty}, \mathrm{clear}(x)\}$	(D)	$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$ $\{\mathrm{holding}(x)\}$ $\{\mathrm{onTable}(x), \mathrm{armEmpty}\}$

  Recall: an actions *a* represented by a tuple $\langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of lists of facts.
- **Hint:** The only differences between them are the delete lists
- **Answer:**

  (A) No, must delete $\mathrm{armEmpty}$

SOME RIGHTS RESERVED

STRIPS for the Blocksworld

- **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\mathrm{pickup}(x)$ action schema?

  (A)
  $\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
  $\{\mathrm{holding}(x)\}$
  $\{\mathrm{onTable}(x)\}$

  (B)
  $\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
  $\{\mathrm{holding}(x)\}$
  $\{\mathrm{armEmpty}\}$

  (C)
  $\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
  $\{\mathrm{holding}(x)\}$
  $\{\mathrm{onTable}(x), \mathrm{armEmpty}, \mathrm{clear}(x)\}$

  (D)
  $\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
  $\{\mathrm{holding}(x)\}$
  $\{\mathrm{onTable}(x), \mathrm{armEmpty}\}$

  Recall: an actions *a* represented by a tuple $\langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of lists of facts.
- **Hint:** The only differences between them are the delete lists
- **Answer:**
  - (A) No, must delete $\mathrm{armEmpty}$
  - (B) No, must delete $\mathrm{onTable}(x)$.

STRIPS for the Blocksworld

- **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\mathrm{pickup}(x)$ action schema?

(A)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x)\}$

(B)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{armEmpty}\}$

(C)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}, \mathrm{clear}(x)\}$

(D)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}\}$

Recall: an actions *a* represented by a tuple $\langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of lists of facts.

- **Hint:** The only differences between them are the delete lists
- **Answer:**

(A) No, must delete $\mathrm{armEmpty}$

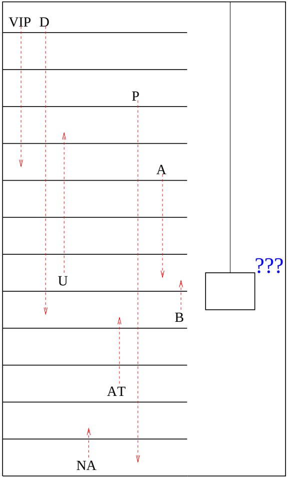
(B) No, must delete $\mathrm{onTable}(x)$.

(C) (D) Both yes: We can, but don't have to, encode the *single-arm* Blocksworld so that the block currently in the hand is not clear.
For (C), $\mathrm{stack}(x, y)$ and $\mathrm{putdown}(x)$ need to add $\mathrm{clear}(x)$, so the encoding on the previous slide does not work.

SOME RIGHTS RESERVED

Miconic-10: A Real-World Example

- **Example 4.16.** Elevator control as a planning problem; details at **[KoeSch:ecpp00]**
  Specify mobility needs before boarding, let a planner schedule/otimize trips



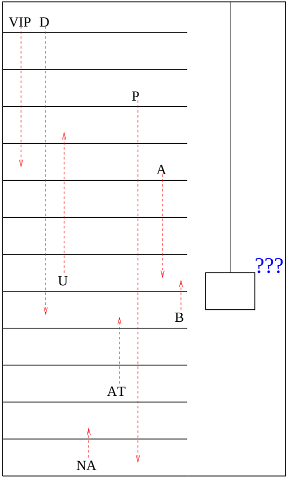
- VIP:
- D:

- NA:
- AT: .
- A, B:
- P:

SOME RIGHTS RESERVED

Miconic-10: A Real-World Example

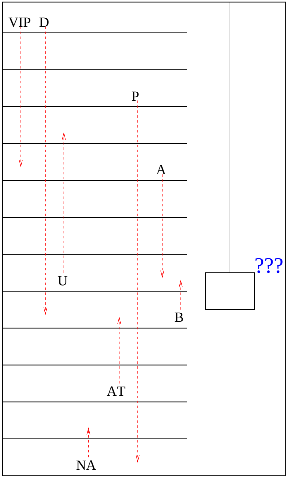
- **Example 4.17.** Elevator control as a planning problem; details at **[KoeSch:ecpp00]**
  Specify mobility needs before boarding, let a planner schedule/otimize trips



  - VIP: Served first.
  - D:
  - NA:
  - AT: .
  - A, B:
  - P:

Miconic-10: A Real-World Example

- **Example 4.18.** Elevator control as a planning problem; details at [**KoeSch:ecpp00**]
  Specify mobility needs before boarding, let a planner schedule/otimize trips

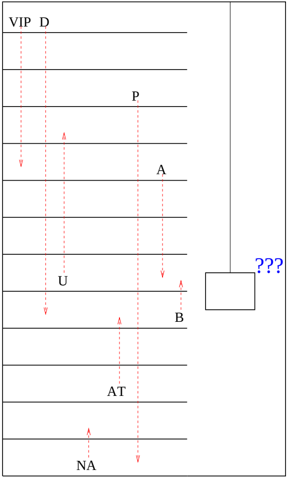


- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA:
- AT: .
- A, B:
- P:

SOME RIGHTS RESERVED

Miconic-10: A Real-World Example

- **Example 4.19.** Elevator control as a planning problem; details at [**KoeSch:ecpp00**]
  Specify mobility needs before boarding, let a planner schedule/otimize trips

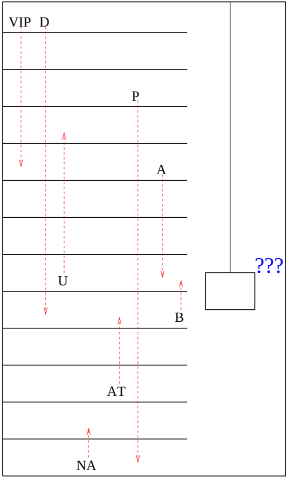


  - VIP: Served first.
  - D: Lift may only go *down* when inside; similar for U.
  - NA: Never-alone
  - AT: .
  - A, B:
  - P:

SOME RIGHTS RESERVED

Miconic-10: A Real-World Example

- **Example 4.20.** Elevator control as a planning problem; details at [**KoeSch:ecpp00**]
  Specify mobility needs before boarding, let a planner schedule/otimize trips

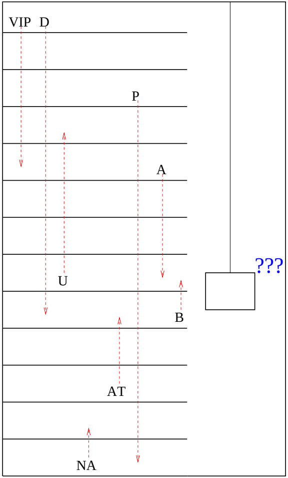


  - VIP: Served first.
  - D: Lift may only go *down* when inside; similar for U.
  - NA: Never-alone
  - AT: Attendant.
  - A, B:
  - P:

SOME RIGHTS RESERVED

Miconic-10: A Real-World Example

- **Example 4.21.** Elevator control as a planning problem; details at [**KoeSch:ecpp00**]
  Specify mobility needs before boarding, let a planner schedule/otimize trips

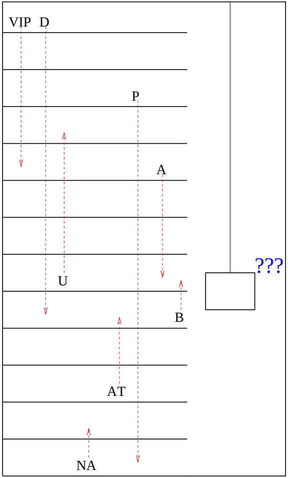


- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: Attendant.
- A, B: Never together in the same elevator
- P:

SOME RIGHTS RESERVED

Miconic-10: A Real-World Example

- **Example 4.22.** Elevator control as a planning problem; details at [**KoeSch:ecpp00**]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: Attendant.
- A, B: Never together in the same elevator
- P: Normal passenger

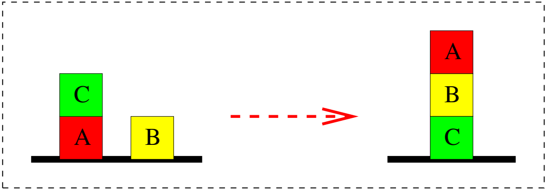
SOME RIGHTS RESERVED

17.5 Partial Order Planning

SOME RIGHTS RESERVED

Planning History, p.s.: Planning is Non-Trivial!

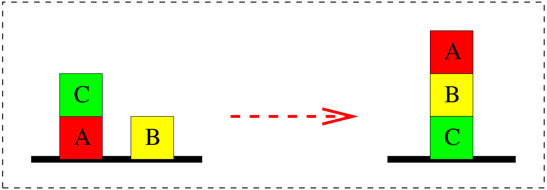
- **Example 5.1.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

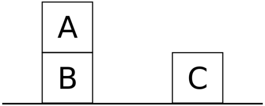
Planning History, p.s.: Planning is Non-Trivial!

- **Example 5.2.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

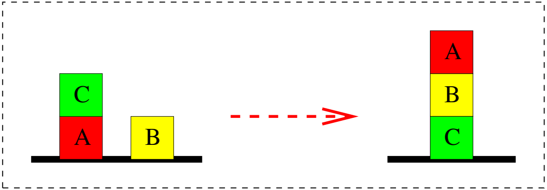
- If we pursue $\mathrm{on}(A, B)$ by unstacking C, and moving A onto B, we achieve the first subgoal, but cannot achieve the second without undoing the first.



SOME RIGHTS RESERVED

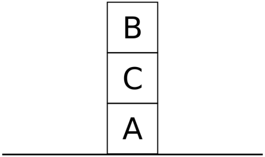
Planning History, p.s.: Planning is Non-Trivial!

- **Example 5.3.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A,B)$ and $\mathrm{on}(B,C)$ fail:

- If we pursue $\mathrm{on}(A,B)$ by unstacking C, and moving A onto B, we achieve the first subgoal, but cannot achieve the second without undoing the first.
- If we pursue $\mathrm{on}(B,C)$ by moving B onto C, we achieve the second subgoal, but cannot achieve the first without undoing the second.



Partial Order Planning

- **Definition 5.4.** Any algorithm that can place two searchprob/actions into a plan without specifying which comes first is called as partial order planning.

Partial Order Planning

- **Definition 5.5.** Any algorithm that can place two searchprob/actions into a plan without specifying which comes first is called as partial order planning.
- **Ideas** for partial order planning:
  - Organize the planning steps in a DAG that supports multiple paths from initial to goal state
    - nodes (steps) are labeled with searchprob/actions(searchprob/actions can occur multiply)
    - edges with propositions added by source and presupposed by target

    acyclicity of the graph induces a partial ordering on steps.
  - additional temporal constraints resolve subgoal interactions and induce a linear order.

Partial Order Planning

- **Definition 5.6.** Any algorithm that can place two searchprob/actions into a plan without specifying which comes first is called as partial order planning.
- **Ideas** for partial order planning:
  - Organize the planning steps in a DAG that supports multiple paths from initial to goal state
    - nodes (steps) are labeled with searchprob/actions(searchprob/actions can occur multiply)
    - edges with propositions added by source and presupposed by target

    acyclicity of the graph induces a partial ordering on steps.
  - additional temporal constraints resolve subgoal interactions and induce a linear order.
- **Advantages** of partial order planning:
  - problems can be decomposed $\leadsto$ can work well with non-cooperative environments.
  - efficient by least-commitment strategy
  - causal links (edges) pinpoint unworkable subplans early.

SOME RIGHTS RESERVED

Partially Ordered Plans

- **Definition 5.7.** Let $\langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply} \rangle$ be a STRIPS task, then a partially ordered plan $\mathcal{P} = \langle V, E \rangle$ is a labeled DAG, where the nodes in V (called steps) are labeled with searchprob/actions from $\mathcal{A}$, or are a
  - start step, which has label "effect" $\mathcal{I}$, or a
  - finish step, which has label "precondition" $\mathcal{G}$.

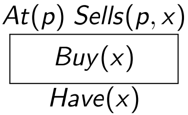
  Every edge $(S, T) \in E$ is either labeled by:
  - A non-empty set $p \subseteq \mathcal{F}$ of facts that are effects of the searchprob/action of S and the preconditions of that of T. We call such a labeled edge a causal link and write it $S \xrightarrow{p} T$.
  - $\prec$, then call it a temporal constraint and write it as $S \prec T$.

  An open condition is a precondition of a step not yet causally linked.
- **Definition 5.8.** Let Π be a partially ordered plan, then we call a step U possibly intervening in a causal link $S \xrightarrow{p} T$, iff $\Pi \cup \{S \prec U, U \prec T\}$ is acyclic.
- **Definition 5.9.** A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it.
- **Definition 5.10.** A partially ordered plan Π is called complete iff every precondition is achieved.
- **Definition 5.11.** Partial order planning is the process of computing complete and acyclic partially ordered plans for a given planning task.

SOME RIGHTS RESERVED

A Notation for STRIPS Actions

- **Definition 5.12 (Notation).** In diagrams, we often write STRIPS searchprob/actions into boxes with preconditions above and effects below.
- **Example 5.13.**
  - Searchprob/actions: $Buy(x)$
  - Preconditions: $At(p)$, $Sells(p,x)$
  - Effects: $Have(x)$



- **Notation:** A causal link $S\xrightarrow{p}T$ can also be denoted by a direct arrow between the effects p of S and the preconditions p of T in the STRIPS action notation above.

  Show temporal constraints as dashed arrows.

Planning Process

- **Definition 5.14.** Partial order planning is search in the space of partial plans via the following operations:
  - add link from an existing action to an open precondition,
  - add step (an action with links to other steps) to fulfil an open precondition,
  - order one step wrt. another (by adding temporal constraints) to remove possible conflicts.
- **Idea:** Gradually move from incomplete/vague plans to complete, correct plans. backtrack if an open condition is unachievable or if a conflict is unresolvable.

SOME RIGHTS RESERVED

Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.15.**

Start

$Sell(SM, Milk)$ $At(Home)$ $Sell(HWS, Drill)$ $Sell(SM, Ban)$

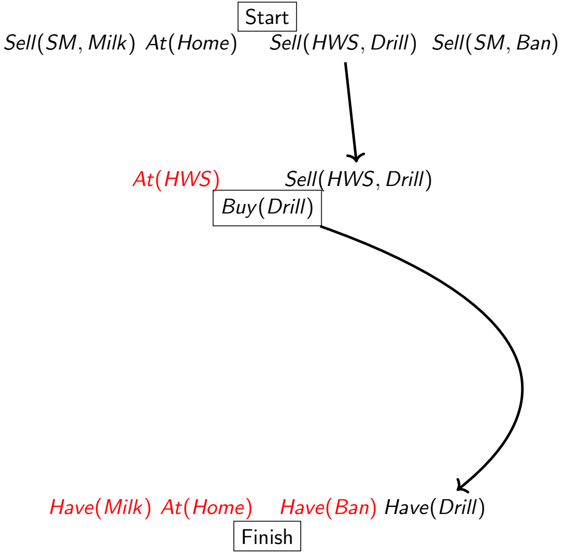
$Have(Milk)$ $At(Home)$ $Have(Ban)$ $Have(Drill)$

Finish

SOME RIGHTS RESERVED

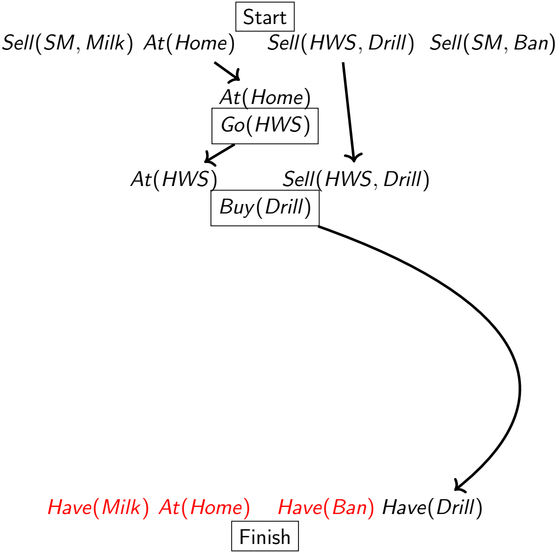
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.16.**


Start
Sell(*SM*, *Milk*) *At*(*Home*) *Sell*(*HWS*, *Drill*) *Sell*(*SM*, *Ban*)
At(*HWS*) *Sell*(*HWS*, *Drill*)
Buy(*Drill*)
Have(*Milk*) *At*(*Home*) *Have*(*Ban*) *Have*(*Drill*)
Finish

Example: Shopping for Bananas, Milk, and a Cordless Drill

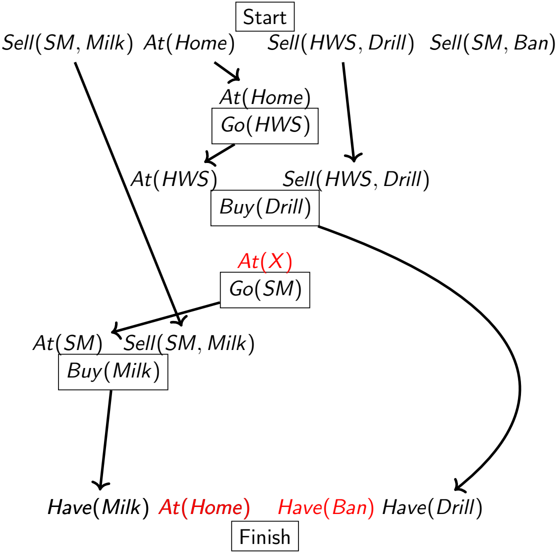
- **Example 5.17.**


Start
Sell(*SM*, *Milk*) *At*(*Home*) *Sell*(*HWS*, *Drill*) *Sell*(*SM*, *Ban*)
At(*Home*)
Go(*HWS*)
At(*HWS*) *Sell*(*HWS*, *Drill*)
Buy(*Drill*)
Have(*Milk*) *At*(*Home*) *Have*(*Ban*) *Have*(*Drill*)
Finish

SOME RIGHTS RESERVED

Example: Shopping for Bananas, Milk, and a Cordless Drill

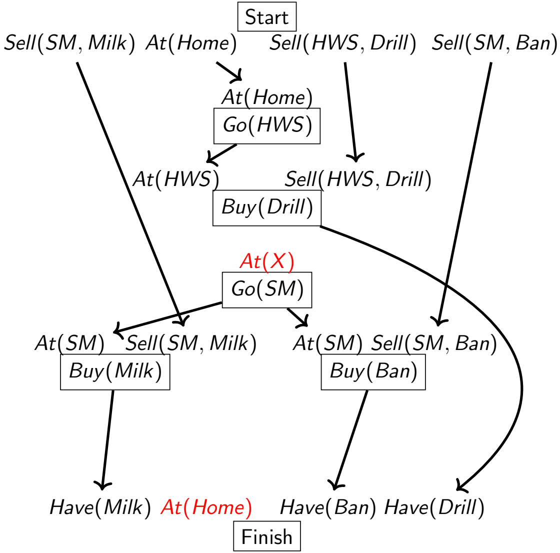
- **Example 5.18.**



SOME RIGHTS RESERVED

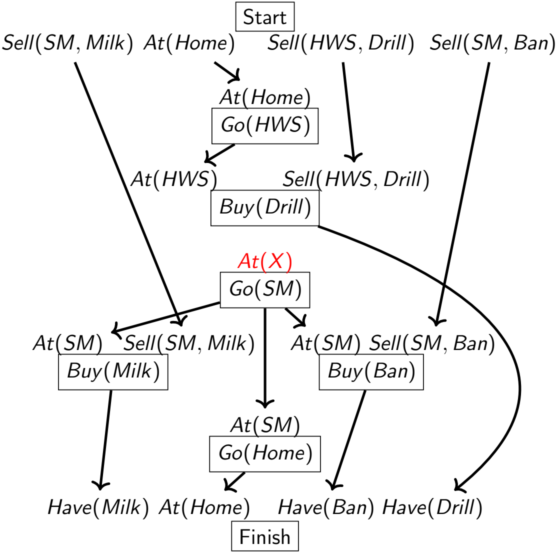
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.19.**



Example: Shopping for Bananas, Milk, and a Cordless Drill

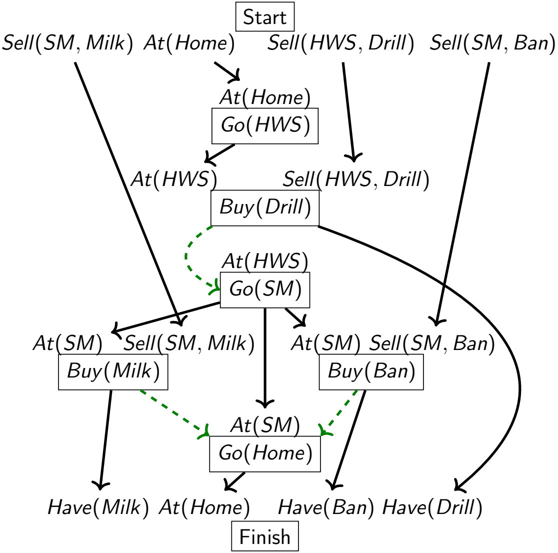
- **Example 5.20.**


Start
$Sell(SM, Milk)$ $At(Home)$ $Sell(HWS, Drill)$ $Sell(SM, Ban)$
$At(Home)$
$Go(HWS)$
$At(HWS)$ $Sell(HWS, Drill)$
$Buy(Drill)$
$At(X)$
$Go(SM)$
$At(SM)$ $Sell(SM, Milk)$
$Buy(Milk)$
$At(SM)$ $Sell(SM, Ban)$
$Buy(Ban)$
$At(SM)$
$Go(Home)$
$Have(Milk)$ $At(Home)$ $Have(Ban)$ $Have(Drill)$
Finish

SOME RIGHTS RESERVED

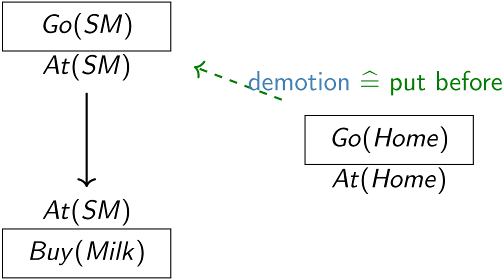
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.21.**


Start
Sell(SM, Milk) *At(Home)* *Sell(HWS, Drill)* *Sell(SM, Ban)*
At(Home)
Go(HWS)
At(HWS) *Sell(HWS, Drill)*
Buy(Drill)
At(HWS)
Go(SM)
At(SM) *Sell(SM, Milk)*
Buy(Milk)
At(SM) *Sell(SM, Ban)*
Buy(Ban)
At(SM)
Go(Home)
Have(Milk) *At(Home)* *Have(Ban)* *Have(Drill)*
Finish

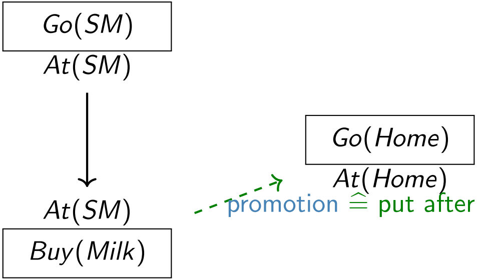
Clobbering and Promotion/Demotion

- **Definition 5.22.** In a partially ordered plan, a step C clobbers a causal link $L := S \xrightarrow{p} T$, iff it destroys the condition p achieved by L.
- **Definition 5.23.** If C clobbers $S \xrightarrow{p} T$ in a partially ordered plan Π, then we can solve the induced conflict by
  - demotion: add a temporal constraint $C \prec S$ to Π, or
  - promotion: add $T \prec C$ to Π.
- **Example 5.24.** $Go(Home)$ clobbers $At(Supermarket)$:



Clobbering and Promotion/Demotion

- **Definition 5.25.** In a partially ordered plan, a step C clobbers a causal link $L := S \xrightarrow{p} T$, iff it destroys the condition p achieved by L.
- **Definition 5.26.** If C clobbers $S \xrightarrow{p} T$ in a partially ordered plan Π, then we can solve the induced conflict by
  - demotion: add a temporal constraint $C \prec S$ to Π, or
  - promotion: add $T \prec C$ to Π.
- **Example 5.27.** $Go(Home)$ clobbers $At(Supermarket)$:



POP algorithm sketch

- **Definition 5.28.** The POP algorithm for constructing complete partially ordered plans:

```
function POP (initial, goal, operators) : plan
    plan:= Make−Minimal−Plan(initial, goal)
    loop do
        if Solution?(goal,plan) then return plan
        $S_{need}$, $c$ := Select−Subgoal(plan)
        Choose−Operator(plan, operators, $S_{need}$,c)
        Resolve−Threats(plan)
    end

function Select−Subgoal (plan, $S_{need}$, $c$)
    pick a plan step $S_{need}$ from Steps(plan)
        with a precondition $c$ that has not been achieved
    return $S_{need}$, $c$
```

SOME RIGHTS RESERVED

POP algorithm contd.

- **Definition 5.29.** The missing parts for the POP algorithm.

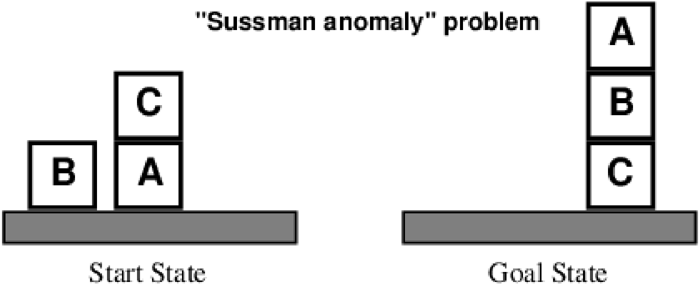
```
function Choose—Operator (plan, operators, $S_{need}$, c)
    choose a step $S_{add}$ from operators or Steps(plan) that has $c$ as an effect
    if there is no such step then fail
    add the causal—link $S_{add} \xrightarrow{c} S_{need}$ to Links(plan)
    add the temporal—constraint $S_{add} \prec S_{need}$ to Orderings(plan)
    if $S_{add}$ is a newly added \step from operators then
        add $S_{add}$ to Steps(plan)
        add $Start \prec S_{add} \prec Finish$ to Orderings(plan)

function Resolve—Threats (plan)
    for each $S_{threat}$ that threatens a causal—link $S_i \xrightarrow{c} S_j$ in Links(plan) do
        choose either
            demotion: Add $S_{threat} \prec S_i$ to Orderings(plan)
            promotion: Add $S_j \prec S_{threat}$ to Orderings(plan)
        if not Consistent(plan) then fail
```

SOME RIGHTS RESERVED

Properties of POP

- Nondeterministic algorithm: backtracks at choice points on failure:
  - choice of S_{add} to achieve S_{need},
  - choice of demotion or promotion for clobberer,
  - selection of S_{need} is irrevocable.
- **Observation 5.30.** *POP is sound, complete, and systematic i.e. no repetition*
- There are extensions for disjunction, universals, negation, conditionals.
- It can be made efficient with good heuristics derived from problem description.
- Particularly good for problems with many loosely related subgoals.

Example: Solving the Sussman Anomaly



Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

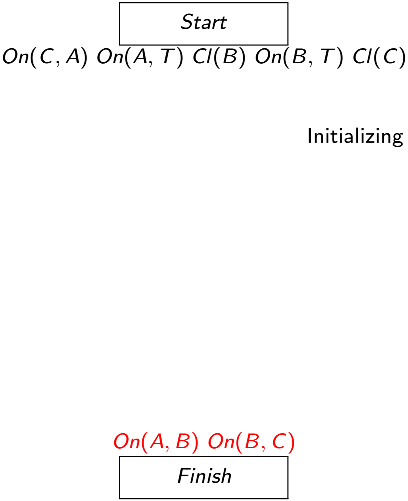
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

SOME RIGHTS RESERVED

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.31.** Solving the Sussman anomaly



Start

$On(C,A)$ $On(A,T)$ $Cl(B)$ $On(B,T)$ $Cl(C)$

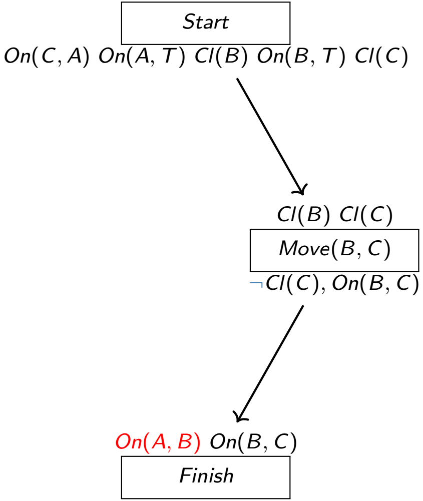
$On(A,B)$ $On(B,C)$

Finish

Initializing the partial order plan with with Start and Finish.

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.32.** Solving the Sussman anomaly



Start

$On(C,A)\ On(A,T)\ Cl(B)\ On(B,T)\ Cl(C)$

Refining for the subgoal $On(B,C)$.

$Cl(B)\ Cl(C)$

Move(B,C)

$\neg Cl(C), On(B,C)$

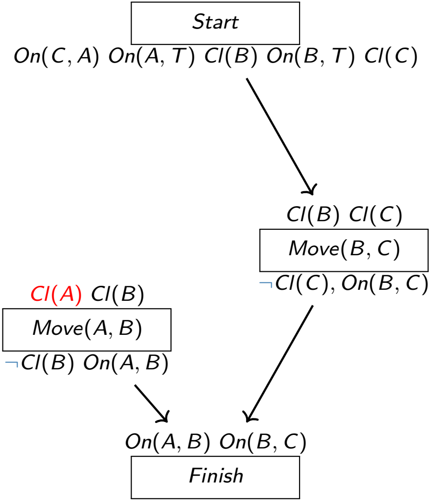
$On(A,B)\ On(B,C)$

Finish

SOME RIGHTS RESERVED

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.33.** Solving the Sussman anomaly

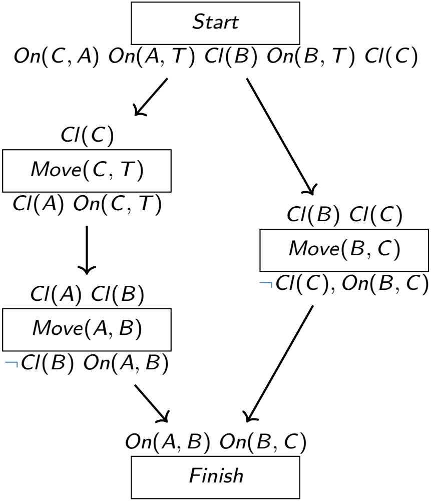


Refining for the subgoal $ON(A,C)$.

SOME RIGHTS RESERVED

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.34.** Solving the Sussman anomaly



Start

$On(C,A)$ $On(A,T)$ $Cl(B)$ $On(B,T)$ $Cl(C)$

$Cl(C)$

Move(C,T)

$Cl(A)$ $On(C,T)$

$Cl(A)$ $Cl(B)$

Move(A,B)

$\neg Cl(B)$ $On(A,B)$

$Cl(B)$ $Cl(C)$

Move(B,C)

$\neg Cl(C), On(B,C)$

$On(A,B)$ $On(B,C)$

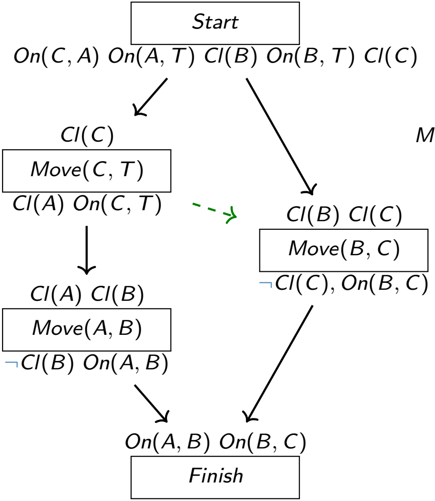
Finish

Refining for the subgoal $Cl(A)$.

SOME RIGHTS RESERVED

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.35.** Solving the Sussman anomaly



Start

$On(C,A)$ $On(A,T)$ $Cl(B)$ $On(B,T)$ $Cl(C)$

$Cl(C)$

$Move(C,T)$

$Cl(A)$ $On(C,T)$

$Cl(A)$ $Cl(B)$

$Move(A,B)$

$\neg Cl(B)$ $On(A,B)$

$Cl(B)$ $Cl(C)$

$Move(B,C)$

$\neg Cl(C), On(B,C)$

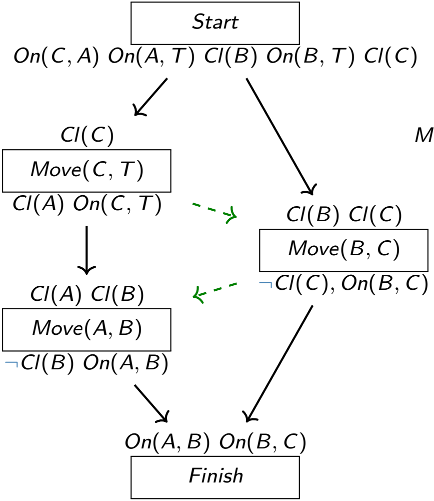
$On(A,B)$ $On(B,C)$

Finish

$Move(A,B)$ clobbers $Cl(B)$ ⇝ demote.

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.36.** Solving the Sussman anomaly



Start

$On(C,A)$ $On(A,T)$ $Cl(B)$ $On(B,T)$ $Cl(C)$

$Cl(C)$

$Move(C,T)$

$Cl(A)$ $On(C,T)$

$Cl(B)$ $Cl(C)$

$Move(B,C)$

$\neg Cl(C), On(B,C)$

$Cl(A)$ $Cl(B)$

$Move(A,B)$

$\neg Cl(B)$ $On(A,B)$

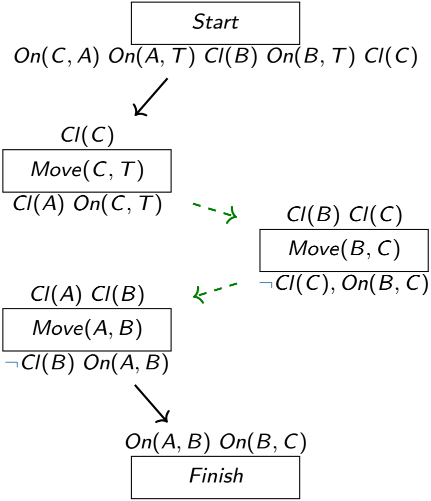
$On(A,B)$ $On(B,C)$

Finish

$Move(B,C)$ clobbers $Cl(C)$ $\rightsquigarrow$ demote.

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.37.** Solving the Sussman anomaly



Start

$On(C,A)$ $On(A,T)$ $Cl(B)$ $On(B,T)$ $Cl(C)$

$Cl(C)$

Move(C,T)

$Cl(A)$ $On(C,T)$

$Cl(B)$ $Cl(C)$

Move(B,C)

$\neg Cl(C), On(B,C)$

$Cl(A)$ $Cl(B)$

Move(A,B)

$\neg Cl(B)$ $On(A,B)$

$On(A,B)$ $On(B,C)$

Finish

A totally ordered plan.

SOME RIGHTS RESERVED

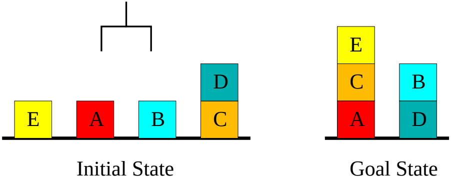
17.6 The PDDL Language

PDDL: Planning Domain Description Language

- **Definition 6.1.** The Planning Domain Description Language (PDDL) is a standardized representation language for planning benchmarks in various extensions of the STRIPS formalism.
- **Definition 6.2.** PDDL is not a propositional language
  - Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic)
  - Action schemas parameterized by objects.
  - Predicates to be instantiated with objects.
- **Definition 6.3.** A PDDL planning task comes in two pieces
  - The problem file gives the objects, the initial state, and the goal state.
  - The domain file gives the predicates and the searchprob/actions.

SOME RIGHTS RESERVED

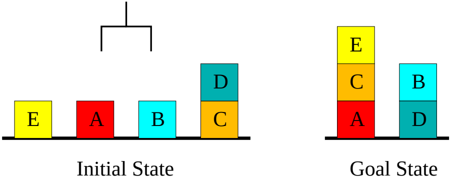
The Blocksworld in PDDL: Domain File



```
(define (domain blocksworld)
  (:predicates (clear ?x) (holding ?x) (on ?x ?y)
               (on−table ?x) (arm−empty))
  (:action stack
   :parameters (?x ?y)
   :precondition (and (clear ?y) (holding ?x))
   :effect (and (arm−empty) (on ?x ?y)
                (not (clear ?y)) (not (holding ?x))))
  . . .)
```

SOME RIGHTS RESERVED

The Blocksworld in PDDL: Problem File



```
(define (problem bw—abcde)
  (:domain blocksworld)
  (:objects a b c d e)
  (:init (on—table a) (clear a)
    (on—table b) (clear b)
    (on—table e) (clear e)
    (on—table c) (on d c) (clear d)
    (arm—empty))
  (:goal (and (on e c) (on c a) (on b d))))
```

Miconic-ADL "Stop" Action Schema in PDDL

```
(:action stop
 :parameters (?f — floor)
 :precondition (and (lift—at ?f)
  (imply
   (exists
    (?p — conflict—A)
    (or (and (not (served ?p))
               (origin ?p ?f))
         (and (boarded ?p)
              (not (destin ?p ?f)))))
   (forall
    (?q — conflict—B)
    (and (or (destin ?q ?f)
             (not (boarded ?q)))
         (or (served ?q)
             (not (origin ?q ?f))))))
  (imply (exists
          (?p — conflict—B)
          (or (and (not (served ?p))
                   (origin ?p ?f))
              (and (boarded ?p)
                   (not (destin ?p ?f)))))
         (forall
          (?q — conflict—A)
          (and (or (destin ?q ?f)
                   (not (boarded ?q)))
               (or (served ?q)
                   (not (origin ?q ?f))))))
  (imply
   (exists
    (?p — never—alone)
    (or (and (origin ?p ?f)
             (not (served ?p)))
        (and (boarded ?p)
             (not (destin ?p ?f)))))
   (exists
    (?q — attendant)
    (or (and (boarded ?q)
             (not (destin ?q ?f)))
        (and (not (served ?q))
             (origin ?q ?f)))))
  (forall
   (?p — going—nonstop)
   (imply (boarded ?p) (destin ?p ?f)))
  (or (forall
       (?p — vip) (served ?p))
      (exists
       (?p — vip)
       (or (origin ?p ?f) (destin ?p ?f))))
  (forall
   (?p — passenger)
   (imply
    (no—access ?p ?f) (not (boarded ?p)))))
)
```

SOME RIGHTS RESERVED

Planning Domain Description Language

- **Question:** What is PDDL good for?
  - (A) Nothing.
  - (B) Free beer.
  - (C) Those AI planning guys.
  - (D) Being lazy at work.

SOME RIGHTS RESERVED

Planning Domain Description Language

- **Question:** What is PDDL good for?
  - (A) Nothing.
  - (B) Free beer.
  - (C) Those AI planning guys.
  - (D) Being lazy at work.
- **Answer:**
  - (A) Nah, it's definitely good for *something* (see remaining answers)

Planning Domain Description Language

- **Question:** What is PDDL good for?
  - (A) Nothing.
  - (B) Free beer.
  - (C) Those AI planning guys.
  - (D) Being lazy at work.
- **Answer:**
  - (A) Nah, it's definitely good for *something* (see remaining answers)
  - (B) Generally, no. Sometimes, yes: PDDL is needed for the IPC, and if you win the IPC you get price money (= free beer).

SOME RIGHTS RESERVED

Planning Domain Description Language

- **Question:** What is PDDL good for?
  - (A) Nothing.
  - (B) Free beer.
  - (C) Those AI planning guys.
  - (D) Being lazy at work.
- **Answer:**
  - (A) Nah, it's definitely good for *something* (see remaining answers)
  - (B) Generally, no. Sometimes, yes: PDDL is needed for the IPC, and if you win the IPC you get price money (= free beer).
  - (C) Yep. (Initially, every system had its own language, so running experiments felt a lot like "Lost in Translation".)

SOME RIGHTS RESERVED

Planning Domain Description Language

- **Question:** What is PDDL good for?
  - (A) Nothing.
  - (B) Free beer.
  - (C) Those AI planning guys.
  - (D) Being lazy at work.
- **Answer:**
  - (A) Nah, it's definitely good for *something* (see remaining answers)
  - (B) Generally, no. Sometimes, yes: PDDL is needed for the IPC, and if you win the IPC you get price money (= free beer).
  - (C) Yep. (Initially, every system had its own language, so running experiments felt a lot like "Lost in Translation".)
  - (D) Yep. You can be a busy bee, programming a solver yourself. Or you can be lazy and just write the PDDL. (I think I said that before . . .)

SOME RIGHTS RESERVED

17.7 Conclusion

SOME RIGHTS RESERVED

Summary

- General problem solving attempts to develop solvers that perform well across a large class of problems.
- Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems. (Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.)
- Heuristic search planning has dominated the International Planning Competition (IPC). We focus on it here.
- STRIPS is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines searchprob/actions in terms of precondition, add list, and delete list.
- PDDL is the de-facto standard language for describing planning problems.
- Plan existence (bounded or not) is PSPACE-complete to decide for STRIPS. If we bound plans polynomially, we get down to NP-completeness.

SOME RIGHTS RESERVED

Chapter 18
Planning II: Algorithms

SOME RIGHTS RESERVED

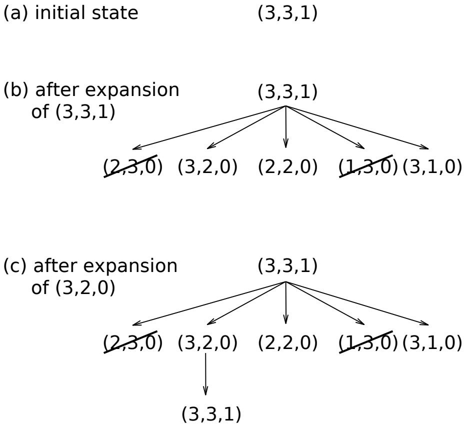
18.1 Introduction

Reminder: Our Agenda for This Topic

- : Background, planning languages, complexity.
  - Sets up the framework. computational complexity is essential to distinguish different algorithmic problems, and for the design of heuristic functions.
- **This Chapter:** How to automatically generate a heuristic function, given planning language input?
  - Focussing on heuristic search as the solution method, this is the main question that needs to be answered.

Reminder: Search

- Starting at initial state, produce all successor states step by step:



(a) initial state (3,3,1)

(b) after expansion of (3,3,1)

(3,3,1)

~~(2,3,0)~~ (3,2,0) (2,2,0) ~~(1,3,0)~~ (3,1,0)

(c) after expansion of (3,2,0)

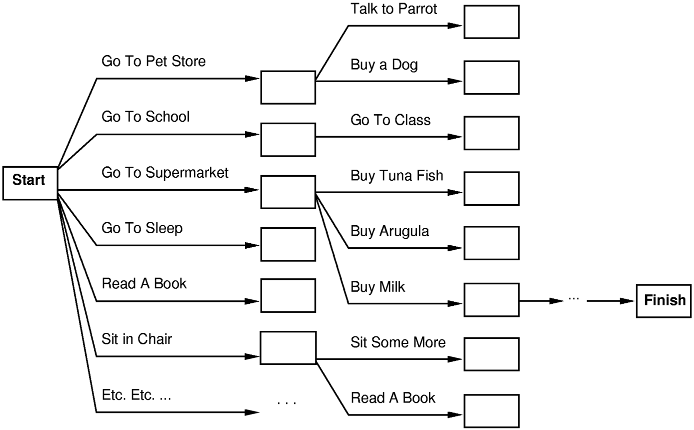
(3,3,1)

~~(2,3,0)~~ (3,2,0) (2,2,0) ~~(1,3,0)~~ (3,1,0)

(3,3,1)

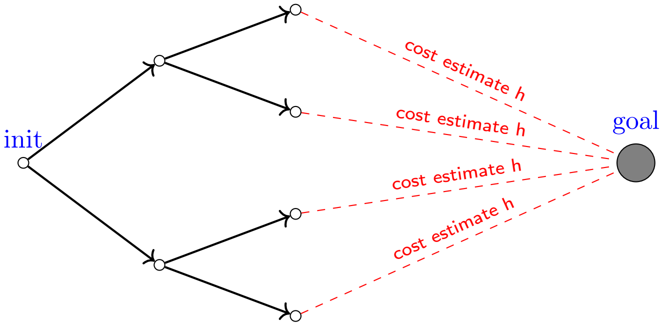
In planning, this is referred to as forward search, or forward state-space search.

Search in the State Space?



- Use heuristic function to guide the search towards the goal!

Reminder: Informed Search



- Heuristic function h estimates the cost of an optimal path from a state s to the goal state; search prefers to expand states s with small $h(s)$.
- Live Demo vs. Breadth-First Search:

`http://qiao.github.io/PathFinding.js/visual/`

SOME RIGHTS RESERVED

Reminder: Heuristic Functions

- **Definition 1.1.** Let Π be a STRIPS task with states S. A heuristic function, short heuristic, for Π is a function $h\colon S \to \mathbb{N} \cup \{\infty\}$ so that $h(s) = 0$ whenever s is a goal state.
- Exactly like our definition from . Except, because we assume unit costs here, we use $\mathbb{N}$ instead of $\mathbb{R}^+$.
- **Definition 1.2.** Let Π be a STRIPS task with states S. The perfect heuristic h^* assigns every $s \in S$ the length of a shortest path from s to a goal state, or ∞ if no such path exists. A heuristic h for Π is admissible if, for all $s \in S$, we have $h(s) \leq h^*(s)$.
- Exactly like our definition from , except for path *length* instead of path *cost* (cf. above).
- In all cases, we attempt to approximate $h^*(s)$, the length of an optimal plan for s. Some algorithms guarantee to lower bound $h^*(s)$.

SOME RIGHTS RESERVED

Our (Refined) Agenda for This Chapter

- **How to Relax:** How to relax a problem?
  - Basic principle for generating heuristic functions.
- **The Delete Relaxation:** How to relax a planning problem?
  - The delete relaxation is the most successful method for the *automatic* generation of heuristic functions. It is a key ingredient to almost all IPC winners of the last decade. It relaxes STRIPS tasks by ignoring the delete lists.
- **The h^+ Heuristic:** What is the resulting heuristic function?
  - h^+ is the "ideal" delete relaxation heuristic.
- **Approximating h^+:** How to actually compute a heuristic?
  - Turns out that, in practice, we must approximate h^+.

SOME RIGHTS RESERVED

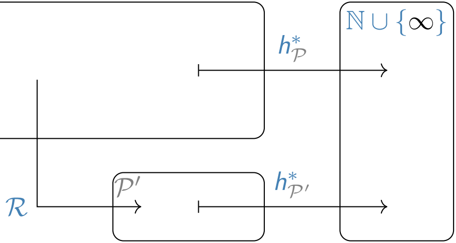
18.2 How to Relax in Planning

How to Relax

- **Recall:** We introduced the concept of a relaxed search problem (allow cheating) to derive heuristics from them.
- **Observation:** This can be generalized to arbitrary problem solving.

How to Relax

- **Recall:** We introduced the concept of a relaxed search problem (allow cheating) to derive heuristics from them.
- **Observation:** This can be generalized to arbitrary problem solving.
- **Definition 2.3 (The General Case).**

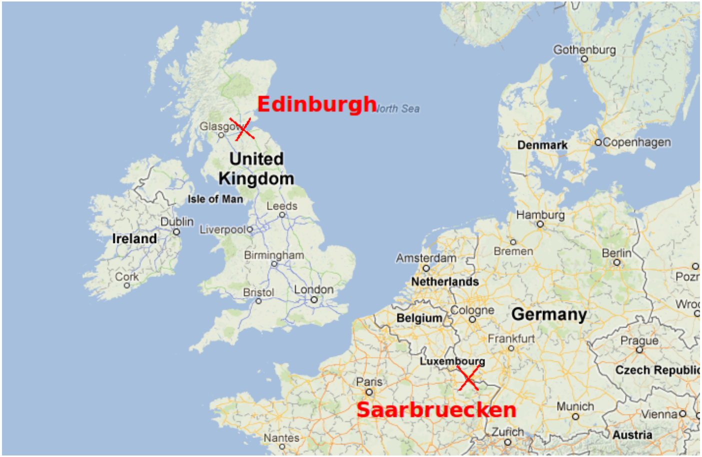


1. You have a class $\mathcal{P}$ of problems, whose perfect heuristic $h^*_{\mathcal{P}}$ you wish to estimate.
2. You define a class $\mathcal{P}'$ of *simpler problems*, whose perfect heuristic $h^*_{\mathcal{P}'}$ can be used to estimate $h^*_{\mathcal{P}}$.
3. You define a transformation – the relaxation mapping $\mathcal{R}$ – that maps instances $\Pi \in \mathcal{P}$ into instances $\Pi' \in \mathcal{P}'$.
4. Given $\Pi \in \mathcal{P}$, you let $\Pi' := \mathcal{R}(\Pi)$, and estimate $h^*{}_{\mathcal{P}}(\Pi)$ by $h^*{}_{\mathcal{P}'}(\Pi')$.

- **Definition 2.4.** For planning tasks, we speak of relaxed planning.

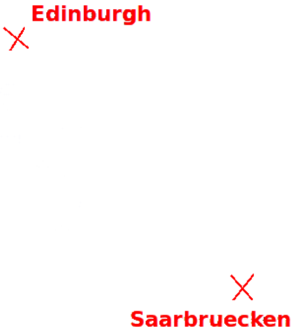
SOME RIGHTS RESERVED

Reminder: Heuristic Functions from Relaxed Problems



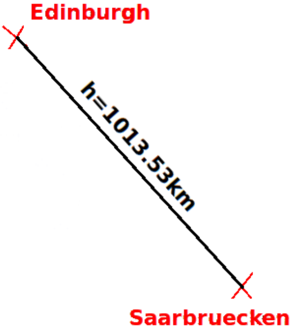
- Problem Π: Find a route from Saarbrücken to Edinburgh.

Reminder: Heuristic Functions from Relaxed Problems



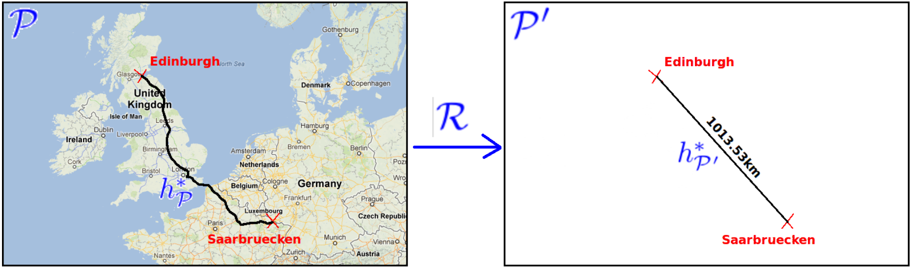
- Relaxed Problem Π': Throw away the map.

Reminder: Heuristic Functions from Relaxed Problems



- Heuristic function h: Straight line distance.

Relaxation in Route-Finding

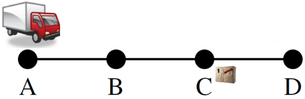


- **Problem class** $\mathcal{P}$: Route finding.
- **Perfect heuristic** $h^*_{\mathcal{P}}$ **for** $\mathcal{P}$: Length of a shortest route.
- **Simpler problem class** $\mathcal{P}'$: Route finding on an empty map.
- **Perfect heuristic** $h^*_{\mathcal{P}'}$ **for** $\mathcal{P}'$: Straight-line distance.
- **Transformation** $\mathcal{R}$: Throw away the map.

SOME RIGHTS RESERVED

How to Relax in Planning? (A Reminder!)

- **Example 2.5 (Logistics).**

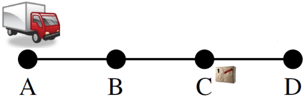


  - facts P: $\{\mathrm{truck}(x)\,|\,x\in\{A,B,C,D\}\}\cup\{\mathrm{pack}(x)\,|\,x\in\{A,B,C,D,T\}\}$.
  - searchprob/initial state I: $\{\mathrm{truck}(A),\mathrm{pack}(C)\}$.
  - searchprob/goal state G: $\{\mathrm{truck}(A),\mathrm{pack}(D)\}$.
  - searchprob/actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
    - $\mathrm{drive}(x,y)$, where x and y have a road: "$\mathrm{truck}(x)\Rightarrow\mathrm{truck}(y),\neg\mathrm{truck}(x)$".
    - $\mathrm{load}(x)$: "$\mathrm{truck}(x),\mathrm{pack}(x)\Rightarrow\mathrm{pack}(T),\neg\mathrm{pack}(x)$".
    - $\mathrm{unload}(x)$: "$\mathrm{truck}(x),\mathrm{pack}(T)\Rightarrow\mathrm{pack}(x),\neg\mathrm{pack}(T)$".

SOME RIGHTS RESERVED

How to Relax in Planning? (A Reminder!)

- **Example 2.7 (Logistics).**

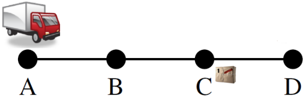


  - facts P: $\{\text{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\text{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.
  - searchprob/initial state I: $\{\text{truck}(A), \text{pack}(C)\}$.
  - searchprob/goal state G: $\{\text{truck}(A), \text{pack}(D)\}$.
  - searchprob/actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
    - $\text{drive}(x, y)$, where x and y have a road: "$\text{truck}(x) \Rightarrow \text{truck}(y), \neg\text{truck}(x)$".
    - $\text{load}(x)$: "$\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T), \neg\text{pack}(x)$".
    - $\text{unload}(x)$: "$\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x), \neg\text{pack}(T)$".
- **Example 2.8 ("Only-Adds" Relaxation).** Drop the preconditions and deletes.
  - "$\text{drive}(x, y)$: $\Rightarrow \text{truck}(y)$";
  - "$\text{load}(x)$: $\Rightarrow \text{pack}(T)$";
  - "$\text{unload}(x)$: $\Rightarrow \text{pack}(x)$".
- Heuristics value for I is?

SOME RIGHTS RESERVED

How to Relax in Planning? (A Reminder!)

- **Example 2.9 (Logistics).**

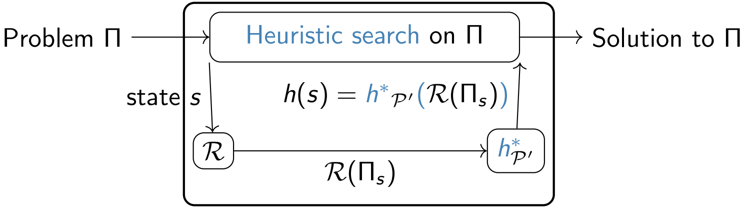


  - facts P: $\{\mathrm{truck}(x)\,|\,x \in \{A, B, C, D\}\} \cup \{\mathrm{pack}(x)\,|\,x \in \{A, B, C, D, T\}\}$.
  - searchprob/initial state I: $\{\mathrm{truck}(A), \mathrm{pack}(C)\}$.
  - searchprob/goal state G: $\{\mathrm{truck}(A), \mathrm{pack}(D)\}$.
  - searchprob/actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
    - $\mathrm{drive}(x, y)$, where x and y have a road: "$\mathrm{truck}(x) \Rightarrow \mathrm{truck}(y), \neg\mathrm{truck}(x)$".
    - $\mathrm{load}(x)$: "$\mathrm{truck}(x), \mathrm{pack}(x) \Rightarrow \mathrm{pack}(T), \neg\mathrm{pack}(x)$".
    - $\mathrm{unload}(x)$: "$\mathrm{truck}(x), \mathrm{pack}(T) \Rightarrow \mathrm{pack}(x), \neg\mathrm{pack}(T)$".
- **Example 2.10 ("Only-Adds" Relaxation).** Drop the preconditions and deletes.
  - "$\mathrm{drive}(x, y)$: $\Rightarrow \mathrm{truck}(y)$";
  - "$\mathrm{load}(x)$: $\Rightarrow \mathrm{pack}(T)$";
  - "$\mathrm{unload}(x)$: $\Rightarrow \mathrm{pack}(x)$".
- Heuristics value for I is?
- $h^{\mathcal{R}}(I) = 1$: A plan for the relaxed task is $[\mathrm{unload}(D)]$.

SOME RIGHTS RESERVED

How to Relax During Search: Overview

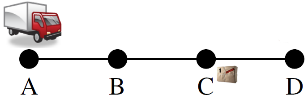
- **Attention:** Search uses the real (un-relaxed) Π. The relaxation is applied (e.g., in Only-Adds, the simplified searchprob/actions are used) **only within the call to $h(s)$!!!**



  - Here, Π_s is Π with initial state replaced by s, i.e., $\Pi := \langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply} \rangle$ changed to $\Pi^s := \langle \mathcal{F}, \mathcal{A}, \{s\}, \mathcal{G} \rangle$: The task of finding a plan for search state s.
  - A common student error is to instead apply the relaxation once to the whole problem, then doing the whole search "within the relaxation".
  - The next slide illustrates the correct search process in detail.

SOME RIGHTS RESERVED

How to Relax During Search: Only-Adds

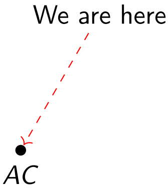


Greedy best-first search:

Real problem:

- Searchprob/initial state I: AC; goal G: AD.
- Searchprob/actions A: pre, add, del.
- $drXY$, loX, ulX.

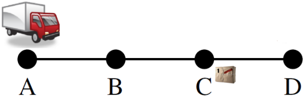
(tie-breaking: alphabetic)



We are here

AC

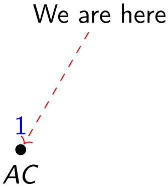
How to Relax During Search: Only-Adds



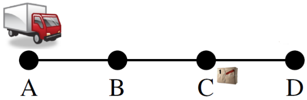
Relaxed problem:

- State s: AC; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



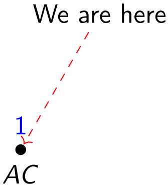
How to Relax During Search: Only-Adds



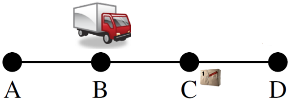
Relaxed problem:

- State s: AC; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s) = 1$: $[ulD]$.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

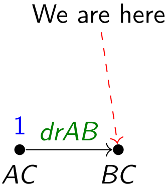


Real problem:

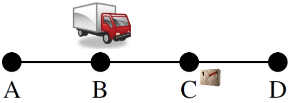
- State s: BC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search:

(tie-breaking: alphabetic)


We are here

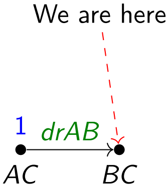
How to Relax During Search: Only-Adds



Relaxed problem:

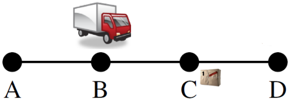
- State s: BC; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

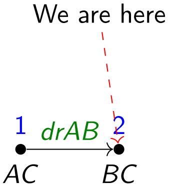
How to Relax During Search: Only-Adds



Relaxed problem:

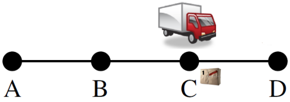
- State s: BC; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $[drBA, ulD]$.

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

How to Relax During Search: Only-Adds

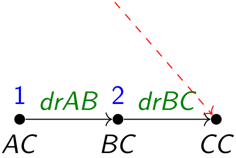


Real problem:

- State s: CC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $BC \xrightarrow{drBC} CC$.

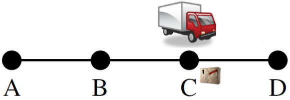
Greedy best-first search: (tie-breaking: alphabetic)

We are here



SOME RIGHTS RESERVED

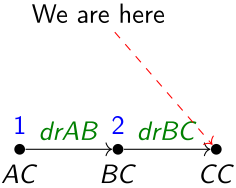
How to Relax During Search: Only-Adds



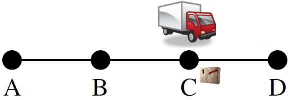
Relaxed problem:

- State s: CC; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



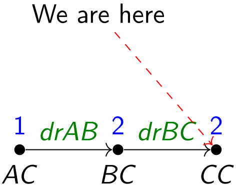
How to Relax During Search: Only-Adds



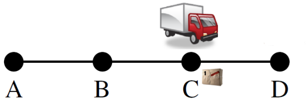
Relaxed problem:

- State s: CC; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s)$ =2: [$drBA$, ulD].

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

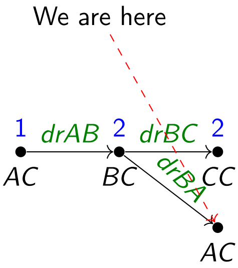


Real problem:

- State s: AC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $BC \xrightarrow{drBA} AC$.

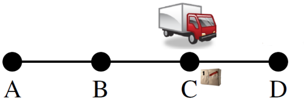
Greedy best-first search:

(tie-breaking: alphabetic)



SOME RIGHTS RESERVED

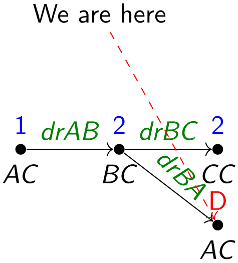
How to Relax During Search: Only-Adds



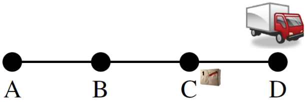
Real problem:

- State s: AC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



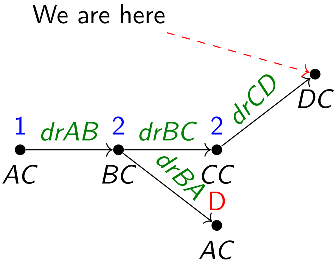
How to Relax During Search: Only-Adds



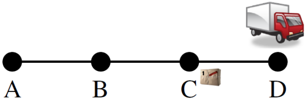
Real problem:

- State s: DC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search: (tie-breaking: alphabetic)



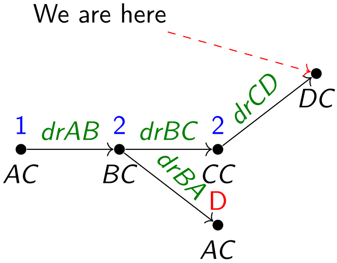
How to Relax During Search: Only-Adds



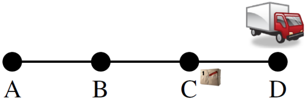
Relaxed problem:

- State s: DC; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



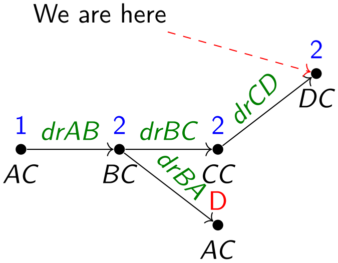
How to Relax During Search: Only-Adds



Relaxed problem:

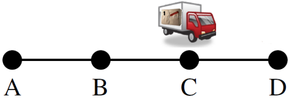
- State s: DC; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $[drBA, ulD]$.

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

How to Relax During Search: Only-Adds



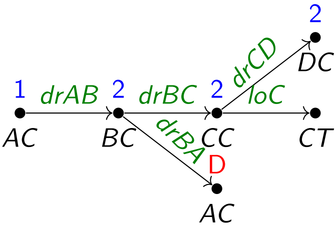
Real problem:

- State s: CT; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{loC} CT$.

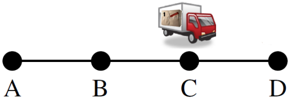
Greedy best-first search:

(tie-breaking: alphabetic)

We are here



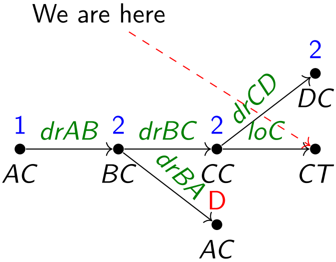
How to Relax During Search: Only-Adds



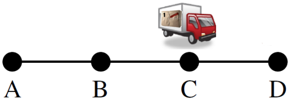
Relaxed problem:

- State s: CT; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



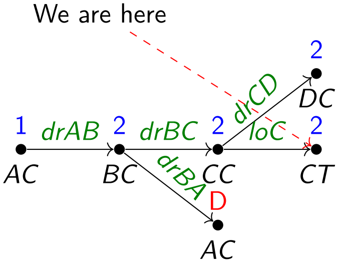
How to Relax During Search: Only-Adds



Relaxed problem:

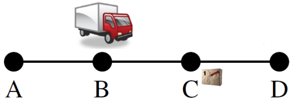
- State s: CT; goal G: AD.
- Searchprob/actions A: add.
- $h^{\mathcal{R}}(s)$ =2: $[drBA, ulD]$.

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

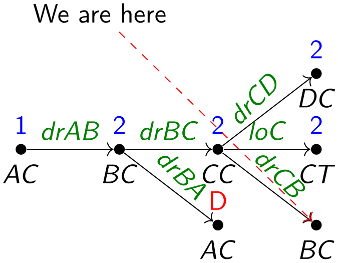
How to Relax During Search: Only-Adds



Real problem:

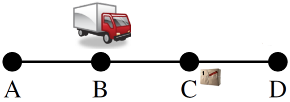
- State s: BC; goal G: AD.
- Searchprob/actions A: pre, add, del.
- $CC \xrightarrow{drCB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

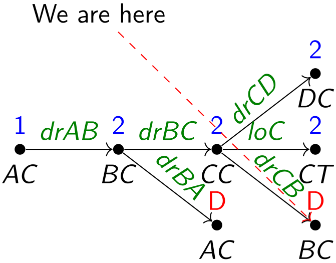
How to Relax During Search: Only-Adds



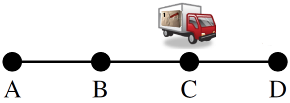
Real problem:

- State s: BC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

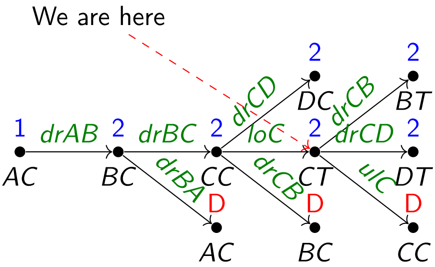


Real problem:

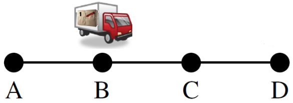
- State s: CT; goal G: AD.
- Searchprob/actions A: pre, add, del.
- Successors: BT, DT, CC.

Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

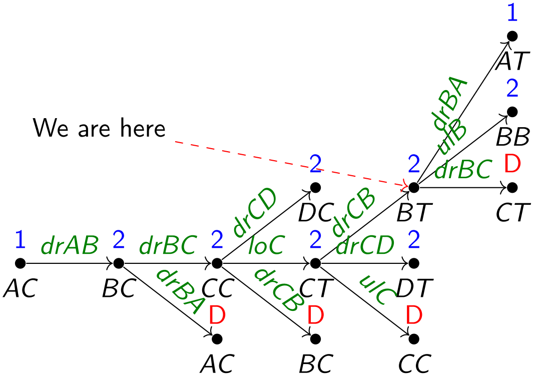


Real problem:

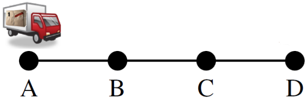
- State s: BT; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Successors: AT, BB, CT.

Greedy best-first search:

(tie-breaking: alphabetic)



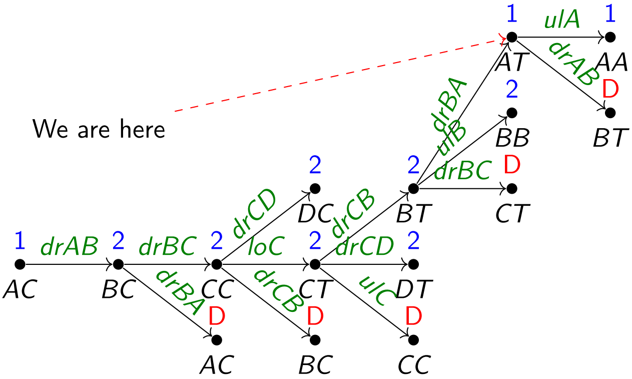
How to Relax During Search: Only-Adds



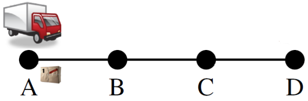
Real problem:

- State s: AT; goal G: AD.
- Searchprob/actions A: pre, add, del.
- Successors: AA, BT.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

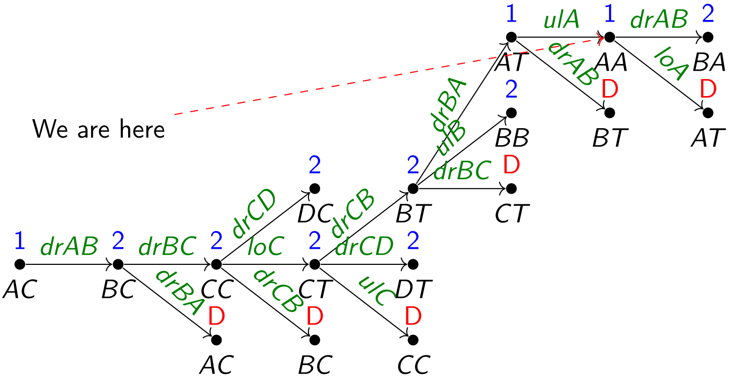


Real problem:

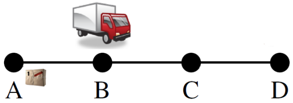
- State s: AA; goal G: AD.
- Searchprob/actions A: pre, add, del.
- Successors: BA, AT.

Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

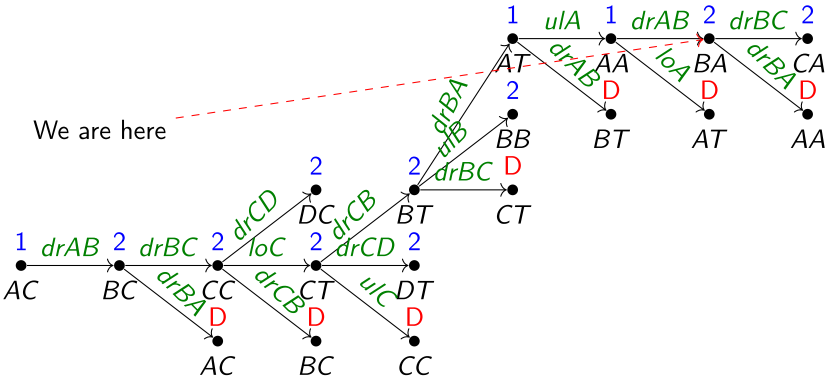


Real problem:

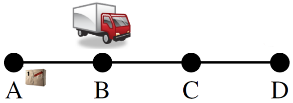
- State s: BA; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Successors: CA, AA.

Greedy best-first search:

(tie-breaking: alphabetic)



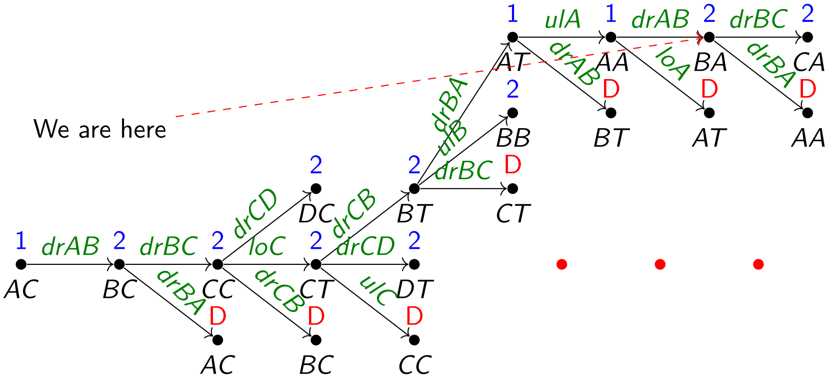
How to Relax During Search: Only-Adds



Real problem:

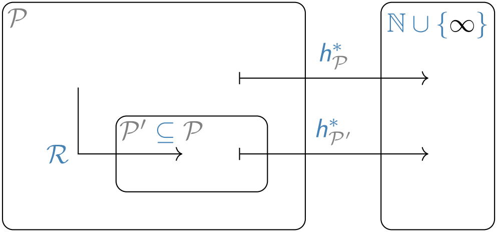
- State *s*: *BA*; goal *G*: *AD*.
- Searchprob/actions *A*: pre, add, del.
- Successors: *CA*, *AA*.

Greedy best-first search: (tie-breaking: alphabetic)



Only-Adds is a "Native" Relaxation

- **Definition 2.11 (Native Relaxations).** Confusing special case where $\mathcal{P}' \subseteq \mathcal{P}$.



  - **Problem class $\mathcal{P}$:** STRIPS tasks.
  - **Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$:** Length h^* of a shortest plan.
  - **Transformation $\mathcal{R}$:** Drop the preconditions and delete lists.
  - **Simpler problem class $\mathcal{P}'$** is a special case of $\mathcal{P}$, $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS tasks with empty preconditions and delete lists.
  - Perfect heuristic for $\mathcal{P}'$: Shortest plan for only-adds STRIPS task.

SOME RIGHTS RESERVED

18.3 The Delete Relaxation

SOME RIGHTS RESERVED

How the Delete Relaxation Changes the World (I)

- Relaxation mapping $\mathcal{R}$ saying that:

"When the world changes, its previous state remains true as well."

Real world: (before)

How the Delete Relaxation Changes the World (I)

- Relaxation mapping $\mathcal{R}$ saying that:

  "When the world changes, its previous state remains true as well."

  Real world: (after)

How the Delete Relaxation Changes the World (I)

- Relaxation mapping $\mathcal{R}$ saying that:

"When the world changes, its previous state remains true as well."

Relaxed world: (before)

How the Delete Relaxation Changes the World (I)

- Relaxation mapping $\mathcal{R}$ saying that:

  “When the world changes, its previous state remains true as well.”

  Relaxed world: (after)

SOME RIGHTS RESERVED

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world: (before)

SOME RIGHTS RESERVED

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world: (after)

SOME RIGHTS RESERVED

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world: (before)

SOME RIGHTS RESERVED

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world: (after)

SOME RIGHTS RESERVED

How the Delete Relaxation Changes the World (III)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world:

How the Delete Relaxation Changes the World (III)

- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world:

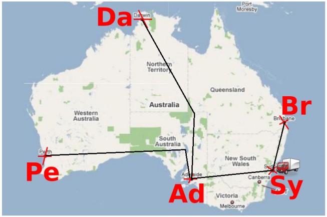
SOME RIGHTS RESERVED

The Delete Relaxation

- **Definition 3.1 (Delete Relaxation).** Let $\Pi := \langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply} \rangle$ be a STRIPS task. The delete relaxation of Π is the task $\Pi^+ = \langle \mathcal{F}, \mathcal{A}^+, \mathcal{I}, \mathcal{G} \rangle$ where $A^+ := \{a^+ \mid a \in A\}$ with $\mathrm{pre}_{a^+} := \mathrm{pre}_a$, $\mathrm{add}_{a^+} := \mathrm{add}_a$, and $\mathrm{del}_{a^+} := \emptyset$.
- In other words, the class of simpler problems $\mathcal{P}'$ is the set of all STRIPS tasks with empty delete lists, and the relaxation mapping $\mathcal{R}$ drops the delete lists.
- **Definition 3.2 (Relaxed Plan).** Let $\Pi := \langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply} \rangle$ be a STRIPS task, and let s be a searchprob/state. A relaxed plan for s is a plan for $\langle \mathcal{F}, \mathcal{A}, s, \mathcal{G} \rangle^+$. A relaxed plan for $\mathcal{I}$ is called a relaxed plan for Π.
- A relaxed plan for s is an searchprob/action sequence that solves s when pretending that all delete lists are empty.
- Also called delete-relaxed plans: "relaxation" is often used to mean delete relaxation by default.

SOME RIGHTS RESERVED

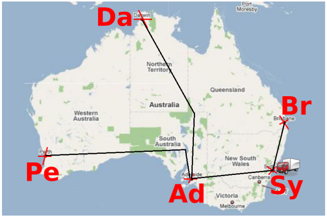
A Relaxed Plan for "TSP" in Australia



1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

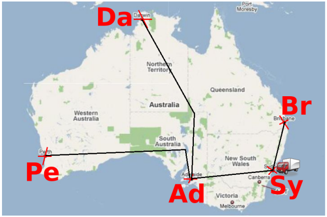
SOME RIGHTS RESERVED

A Relaxed Plan for "TSP" in Australia



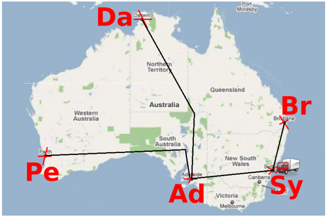
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^{+}$**:** $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for “TSP” in Australia



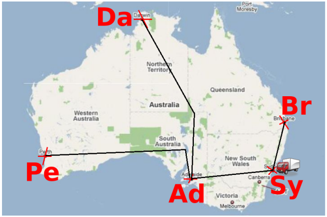
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^{+}$**:** $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
3. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Ad})^{+}$**:** $\{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for “TSP” in Australia



1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^{+}$**:** $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
3. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Ad})^{+}$**:** $\{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
4. $\mathrm{drv}(\mathrm{Ad}, \mathrm{Pe})^{+}$**:**
   $\{\mathrm{at}(\mathrm{Pe}), \mathrm{vis}(\mathrm{Pe}), \mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

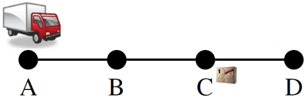
A Relaxed Plan for “TSP” in Australia



1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^{+}$: $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
3. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Ad})^{+}$: $\{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
4. $\mathrm{drv}(\mathrm{Ad}, \mathrm{Pe})^{+}$: $\{\mathrm{at}(\mathrm{Pe}), \mathrm{vis}(\mathrm{Pe}), \mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
5. $\mathrm{drv}(\mathrm{Ad}, \mathrm{Da})^{+}$: $\{\mathrm{at}(\mathrm{Da}), \mathrm{vis}(\mathrm{Da}), \mathrm{at}(\mathrm{Pe}), \mathrm{vis}(\mathrm{Pe}), \mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

SOME RIGHTS RESERVED

A Relaxed Plan for "Logistics"



- Facts P: $\{\mathrm{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\mathrm{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.
- Initial state I: $\{\mathrm{truck}(A), \mathrm{pack}(C)\}$.
- Goal G: $\{\mathrm{truck}(A), \mathrm{pack}(D)\}$.
- Relaxed searchprob/actions A^+: (Notated as "precondition $\Rightarrow$ adds")
  - $\mathrm{drive}(x, y)^+$: "$\mathrm{truck}(x) \Rightarrow \mathrm{truck}(y)$".
  - $\mathrm{load}(x)^+$: "$\mathrm{truck}(x), \mathrm{pack}(x) \Rightarrow \mathrm{pack}(T)$".
  - $\mathrm{unload}(x)^+$: "$\mathrm{truck}(x), \mathrm{pack}(T) \Rightarrow \mathrm{pack}(x)$".

  Relaxed plan:

$$[\mathrm{drive}(A, B)^+, \mathrm{drive}(B, C)^+, \mathrm{load}(C)^+, \mathrm{drive}(C, D)^+, \mathrm{unload}(D)^+]$$

- We don't need to drive the truck back, because "it is still at A".

PlanEx^+

- **Definition 3.3 (Relaxed Plan Existence Problem).** By PlanEx^+, we denote the problem of deciding, given a STRIPS task $\Pi := \langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply} \rangle$, whether or not there exists a relaxed plan for Π.
- This is easier than PlanEx for general STRIPS!
- PlanEx^+ is in P.
- *Proof:* The following algorithm decides PlanEx^+

1.

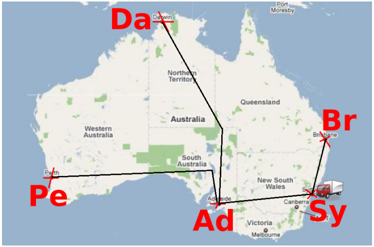
```
var $F := I$
while $G \not\subseteq F$ do
    $F' := F \cup \bigcup_{a \in A : \mathrm{pre}_a \subseteq F} \mathrm{add}_a$
    if $F' = F$ then return ``unsolvable'' endif                (*)
    $F := F'$
endwhile
return ``solvable''
```

2. The algorithm terminates after at most $|\mathcal{F}|$ iterations, and thus runs in polynomial time.
3. Correctness: See slide 635

☐

SOME RIGHTS RESERVED

Deciding PlanEx$^+$ in “TSP” in Australia



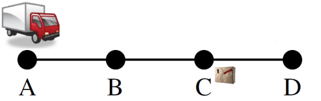
Iterations on F:

1. $\{\mathrm{at(Sy)}, \mathrm{vis(Sy)}\}$
2. $\cup\ \{\mathrm{at(Ad)}, \mathrm{vis(Ad)}, \mathrm{at(Br)}, \mathrm{vis(Br)}\}$
3. $\cup\ \{\mathrm{at(Da)}, \mathrm{vis(Da)}, \mathrm{at(Pe)}, \mathrm{vis(Pe)}\}$

SOME RIGHTS RESERVED

Deciding PlanEx^{+} in "Logistics"

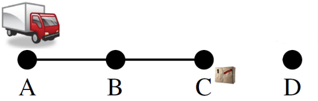
- **Example 3.4 (The solvable Case).**



Iterations on F:

1. $\{\text{truck}(A), \text{pack}(C)\}$
2. $\cup\{\text{truck}(B)\}$
3. $\cup\{\text{truck}(C)\}$
4. $\cup\{\text{truck}(D), \text{pack}(T)\}$
5. $\cup\{\text{pack}(A), \text{pack}(B), \text{pack}(D)\}$

- **Example 3.5 (The unsolvable Case).**



Iterations on F:

1. $\{\text{truck}(A), \text{pack}(C)\}$
2. $\cup\{\text{truck}(B)\}$
3. $\cup\{\text{truck}(C)\}$
4. $\cup\{\text{pack}(T)\}$
5. $\cup\{\text{pack}(A), \text{pack}(B)\}$
6. $\cup\emptyset$

SOME RIGHTS RESERVED

PlanEx$^+$ Algorithm: Proof

Proof: To show: The algorithm returns "solvable" iff there is a relaxed plan for Π.

1. Denote by F_i the content of F after the ith iteration of the while-loop,
2. All $a \in A_0$ are applicable in I, all $a \in A_1$ are applicable in $\mathrm{apply}(A_0^+, I)$, and so forth.
3. Thus $F_i = \mathrm{apply}([A_0^+, \ldots, A_{i-1}^+], I)$. (Within each A_j^+, we can sequence the searchprob/actions in any order.)
4. Direction "$\Rightarrow$"
   If "solvable" is returned after iteration n then $G \subseteq F_n = \mathrm{apply}([A_0^+, \ldots, A_{n-1}^+], I)$ so $[A_0^+, \ldots, A_{n-1}^+]$ can be sequenced to a relaxed plan which shows the claim.
6. Direction "$\Leftarrow$"
   - 6.1. Let $[a_0^+, \ldots, a_{n-1}^+]$ be a relaxed plan, hence $G \subseteq \mathrm{apply}(\langle a_0^+, \ldots, a_{n-1}^+ \rangle, I)$.
   - 6.2. Assume, for the moment, that we drop line (*) from the algorithm. It is then easy to see that $a_i \in A_i$ and $\mathrm{apply}(\langle a_0^+, \ldots, a_{i-1}^+ \rangle, I) \subseteq F_i$, for all i.
   - 6.3. We get $G \subseteq \mathrm{apply}(\langle a_0^+, \ldots, a_{n-1}^+ \rangle, I) \subseteq F_n$, and the algorithm returns "solvable" as desired.
   - 6.4. Assume to the contrary of the claim that, in an iteration $i < n$, (*) fires. Then $G \not\subseteq F$ and $F = F'$. But, with $F = F'$, $F = F_j$ for all $j > i$, and we get $G \not\subseteq F_n$ in contradiction.

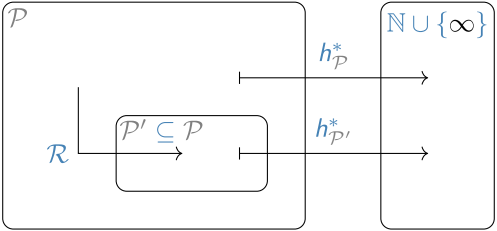
☐

SOME RIGHTS RESERVED

18.4 The h^+Heuristic

SOME RIGHTS RESERVED

Hold on a Sec – Where are we?



- $\mathcal{P}$: STRIPS tasks; $h^*_{\mathcal{P}}$: Length h^* of a shortest plan.
- $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS tasks with empty delete lists.
- $\mathcal{R}$: Drop the delete lists.
- Heuristic function: Length of a shortest relaxed plan ($h^* \circ \mathcal{R}$).
- PlanEx$^+$ is not actually what we're looking for. PlanEx$^+$ $\widehat{=}$ relaxed plan *existence*; we want relaxed plan *length* $h^* \circ \mathcal{R}$.

h^+: The Ideal Delete Relaxation Heuristic

- **Definition 4.1 (Optimal Relaxed Plan).** Let $\langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply}\rangle$ be a STRIPS task, and let s be a searchprob/state. A optimal relaxed plan for s is an optimal plan for $\langle \mathcal{F}, \mathcal{A}, \{s\}, \mathcal{G}\rangle^+$.
- Same as slide 629, just adding the word "optimal".
- Here's what we're looking for:
- **Definition 4.2.** Let $\Pi := \langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply}\rangle$ be a STRIPS task with searchprob/states S. The ideal delete relaxation heuristic h^+ for Π is the function $h^+ \colon S \to \mathbb{N} \cup \{\infty\}$ where $h^+(s)$ is the length of an optimal relaxed plan for s if a relaxed plan for s exists, and $h^+(s) = \infty$ otherwise.
- In other words, $h^+ = h^* \circ \mathcal{R}$, cf. previous slide.

SOME RIGHTS RESERVED

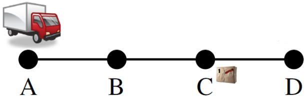
h^+ is Admissible

- **Lemma 4.3.** *Let* $\Pi := \langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply} \rangle$ *be a STRIPS task, and let s be a state. If* $[a_1, \ldots, a_n]$ *is a plan for* $\Pi_s := \langle \mathcal{F}, \mathcal{A}, \{s\}, \mathcal{G} \rangle$, *then* $[a_1^+, \ldots, a_n^+]$ *is a plan for* Π^+.
- *Proof sketch:* Show by induction over $0 \leq i \leq n$ that $\mathrm{apply}([a_1, \ldots, a_i], s) \subseteq \mathrm{apply}([a_1^+, \ldots, a_i^+], s)$.
- If we ignore deletes, the states along the plan can only get bigger.
- **Theorem 4.4.** *h^+ is Admissible.*
- *Proof:*
  1. Let $\Pi := \langle \mathrm{pre}, \mathrm{add}, \mathrm{del}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}, \mathrm{successorfunction}, \mathrm{successorstate}, \mathrm{apply} \rangle$ be a STRIPS task with states $\mathcal{F}$, and let $s \in \mathcal{F}$.
  2. $h^+(s)$ is defined as optimal plan length in Π_s^+.
  3. With the lemma above, any plan for Π also constitutes a plan for Π_s^+.
  4. Thus optimal plan length in Π_s^+ can only be shorter than that in $\Pi_s i$, and the claim follows.

☐

SOME RIGHTS RESERVED

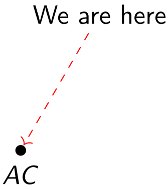
How to Relax During Search: Ignoring Deletes



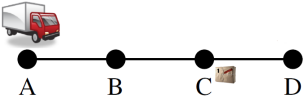
Real problem:

- Initial state I: AC; goal G: AD.
- Searchprob/actions A: pre, add, del.
- $drXY, loX, ulX$.

Greedy best-first search: (tie-breaking: alphabetic)



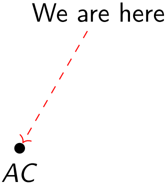
How to Relax During Search: Ignoring Deletes



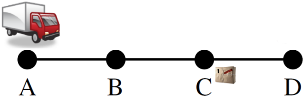
Relaxed problem:

- State s: AC; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^{+}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

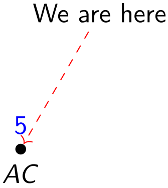


Relaxed problem:

- State s: AC; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^+(s) = 5$: e.g. $[drAB, drBC, drCD, loC, ulD]$.

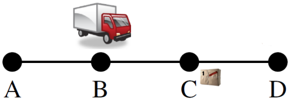
(tie-breaking: alphabetic)

Greedy best-first search:



SOME RIGHTS RESERVED

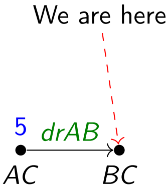
How to Relax During Search: Ignoring Deletes



Real problem:

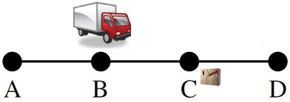
- State s: BC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

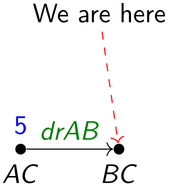
How to Relax During Search: Ignoring Deletes



Relaxed problem:

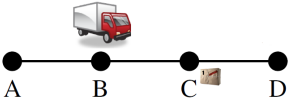
- State s: BC; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

How to Relax During Search: Ignoring Deletes

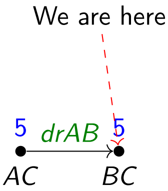


Relaxed problem:

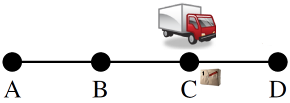
- State s: BC; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^{+}(s) = 5$: e.g. $[drBA, drBC, drCD, loC, ulD]$.

(tie-breaking: alphabetic)

Greedy best-first search:



How to Relax During Search: Ignoring Deletes

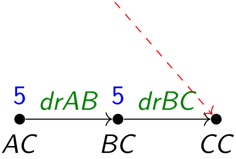


Real problem:

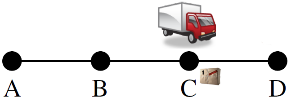
- State s: CC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here



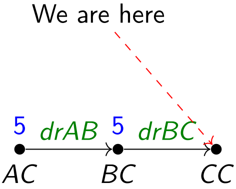
How to Relax During Search: Ignoring Deletes



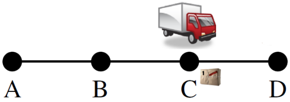
Relaxed problem:

- State s: CC; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^{+}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

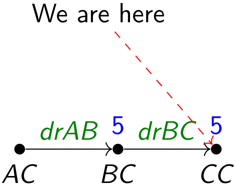


Relaxed problem:

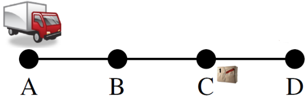
- State s: CC; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^+(s)$ =5: e.g. $[drCB, drBA, drCD, loC, ulD]$.

(tie-breaking: alphabetic)

Greedy best-first search:



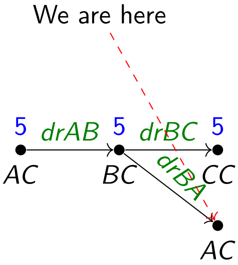
How to Relax During Search: Ignoring Deletes



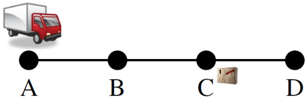
Real problem:

- State s: AC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $BC \xrightarrow{drBA} AC$.

Greedy best-first search: (tie-breaking: alphabetic)



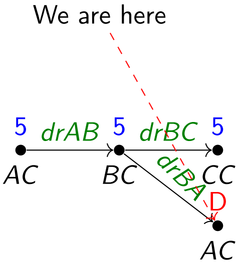
How to Relax During Search: Ignoring Deletes



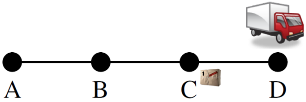
Real problem:

- State s: AC; goal G: AD.
- Searchprob/actions A: pre, add, del.
- Duplicate searchprob/state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

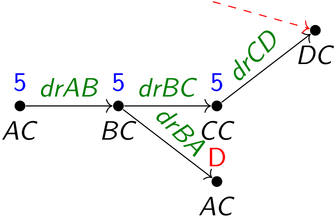


Real problem:

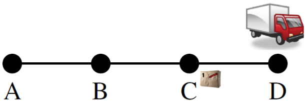
- State s: DC; goal G: AD.
- Searchprob/actions A: pre, add, del.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here



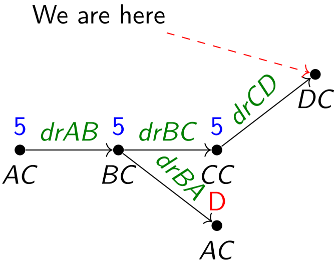
How to Relax During Search: Ignoring Deletes



Relaxed problem:

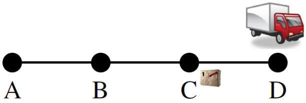
- State s: DC; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^{+}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

How to Relax During Search: Ignoring Deletes

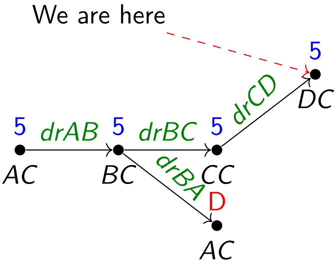


Relaxed problem:

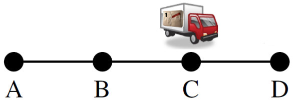
- State s: DC; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^{+}(s)$ =5: e.g. $[drDC, drCB, drBA, loC, ulD]$.

(tie-breaking: alphabetic)

Greedy best-first search:



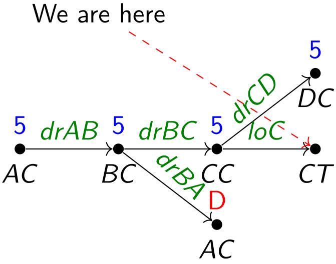
How to Relax During Search: Ignoring Deletes



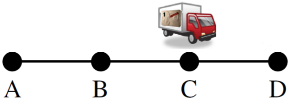
Real problem:

- State s: CT; goal G: AD.
- Searchprob/actions A: pre, add, del.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search: (tie-breaking: alphabetic)



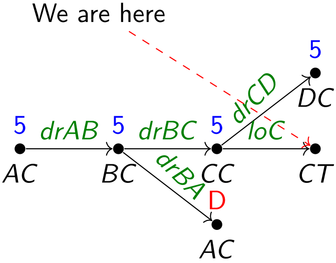
How to Relax During Search: Ignoring Deletes



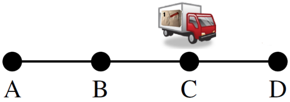
Relaxed problem:

- State s: CT; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^{+}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

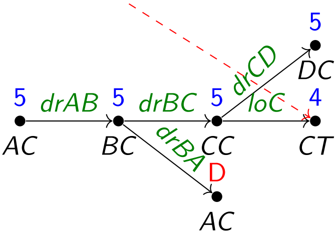


Relaxed problem:

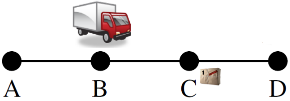
- State s: CT; goal G: AD.
- Searchprob/actions A: pre, add.
- $h^{+}(s)$ =4: e.g. $[drCB, drBA, drCD, ulD]$.

(tie-breaking: alphabetic)

Greedy best-first search:


We are here

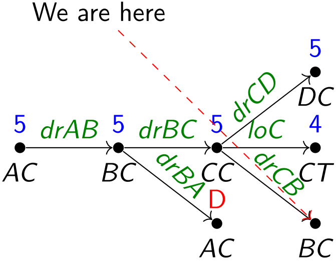
How to Relax During Search: Ignoring Deletes



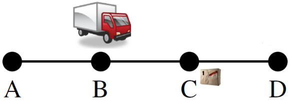
Real problem:

- State s: BC; goal G: AD.
- Searchprob/actions A: pre, add, del.
- $CC \xrightarrow{drCB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



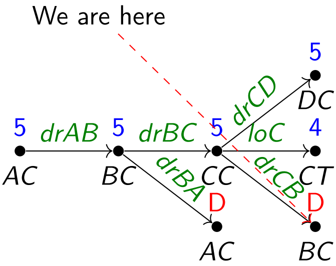
How to Relax During Search: Ignoring Deletes



Real problem:

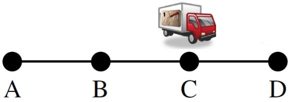
- State s: BC; goal G: AD.
- Searchprob/actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

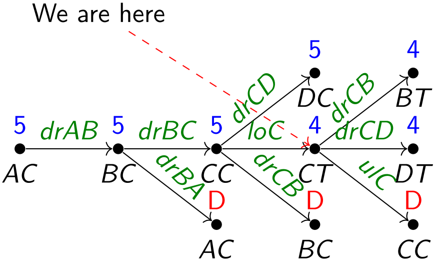
How to Relax During Search: Ignoring Deletes



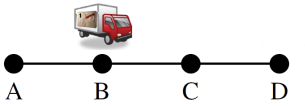
Real problem:

- State s: CT; goal G: AD.
- Searchprob/actions A: pre, add, del.
- Successors: BT, DT, CC.

Greedy best-first search: (tie-breaking: alphabetic)



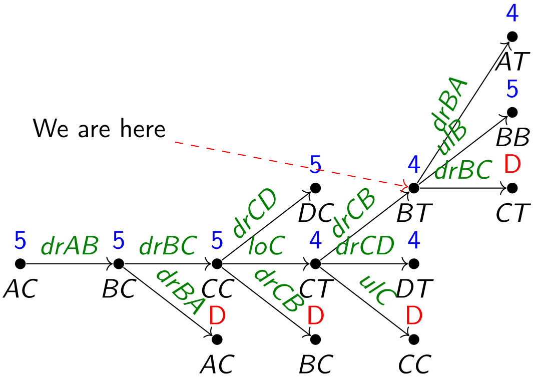
How to Relax During Search: Ignoring Deletes



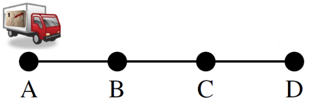
Real problem:

- State s: BT; goal G: AD.
- Searchprob/actions A: pre, add, del.
- Successors: AT, BB, CT.

Greedy best-first search: (tie-breaking: alphabetic)


We are here

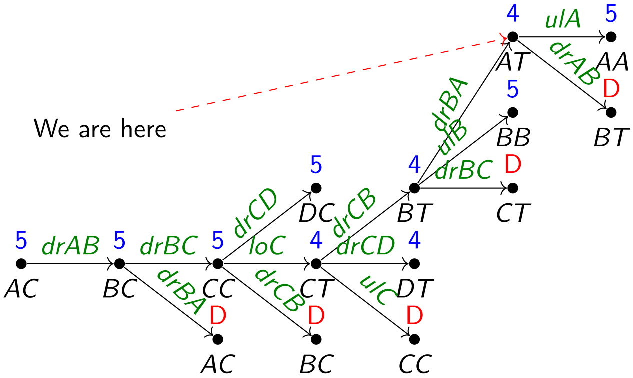
How to Relax During Search: Ignoring Deletes



Real problem:

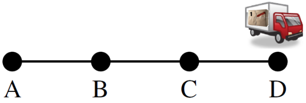
- State s: AT; goal G: AD.
- Searchprob/actions A: pre, add, del.
- Successors: AA, BT.

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

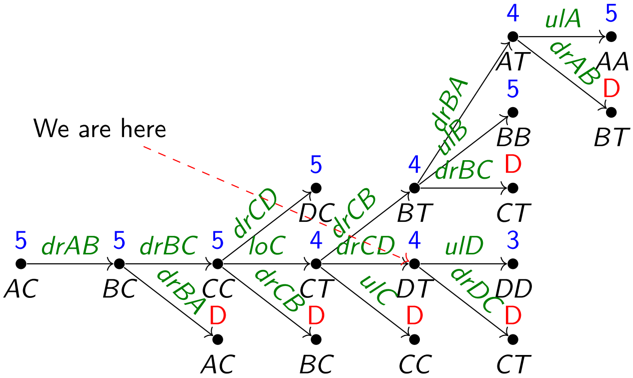
How to Relax During Search: Ignoring Deletes



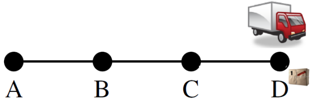
Real problem:

- State s: DT; goal G: AD.
- Searchprob/actions A: $\text{pre}, \text{add}, \text{del}$.
- Successors: DD, CT.

Greedy best-first search: (tie-breaking: alphabetic)


We are here

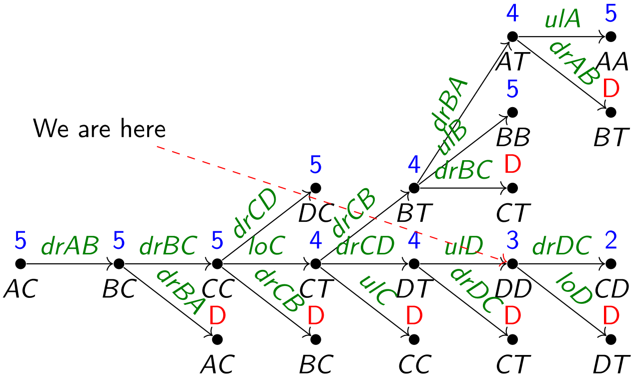
How to Relax During Search: Ignoring Deletes



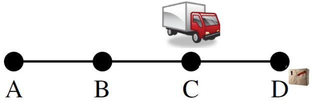
Real problem:

- State *s*: *DD*; goal *G*: *AD*.
- Searchprob/actions *A*: pre, add, del.
- Successors: *CD*, *DT*.

Greedy best-first search: (tie-breaking: alphabetic)



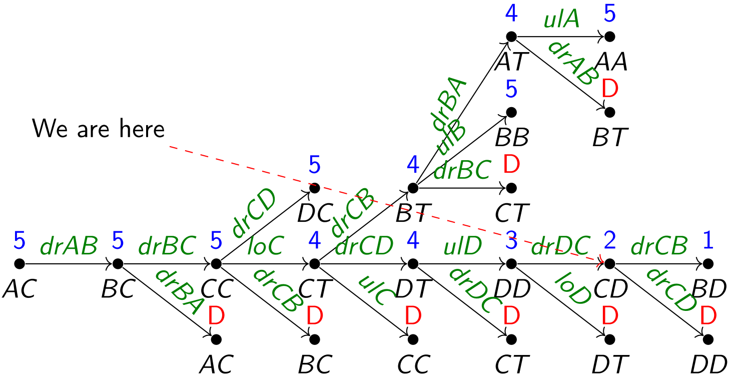
How to Relax During Search: Ignoring Deletes



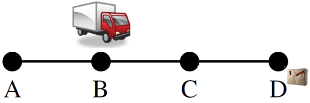
Real problem:

- State *s*: *CD*; goal *G*: *AD*.
- Searchprob/actions *A*: pre, add, del.
- Successors: *BD*, *DD*.

Greedy best-first search: (tie-breaking: alphabetic)


We are here

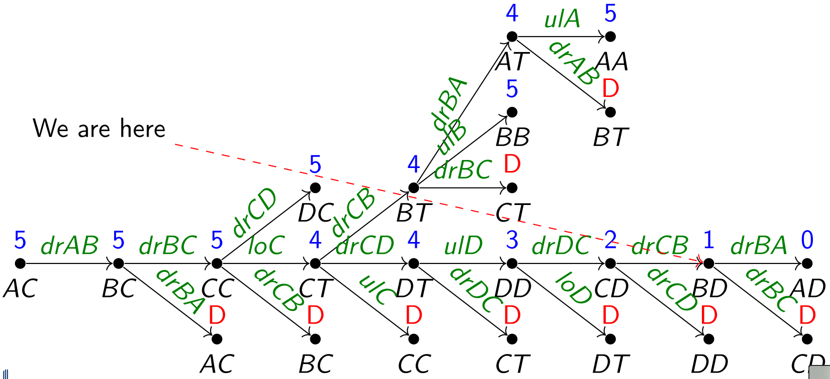
How to Relax During Search: Ignoring Deletes



Real problem:

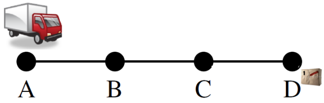
- State s: BD; goal G: AD.
- Searchprob/actions A: pre, add, del.
- Successors: AD, CD.

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

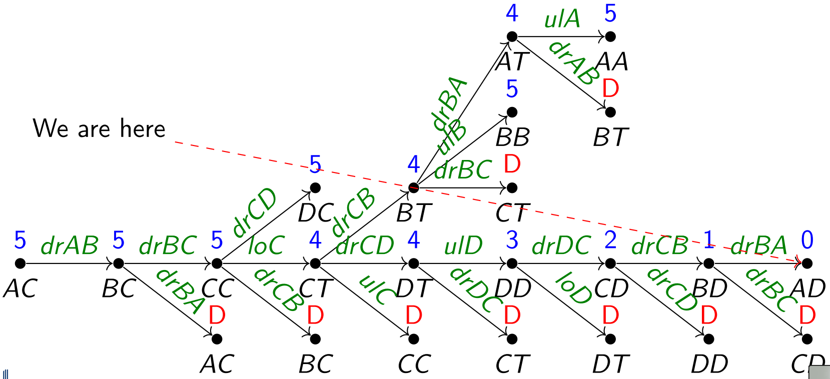
How to Relax During Search: Ignoring Deletes



Real problem:

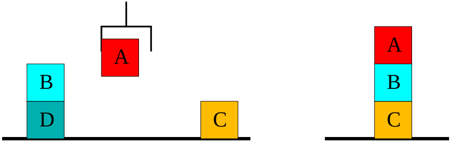
- State *s*: *AD*; goal *G*: *AD*.
- Searchprob/actions *A*: pre, add, del.
- Goal state!

Greedy best-first search: (tie-breaking: alphabetic)



SOME RIGHTS RESERVED

h^+ in the Blocksworld

- 

Initial State Goal State

- **Optimal plan**: $[\mathrm{putdown}(A), \mathrm{unstack}(B, D), \mathrm{stack}(B, C), \mathrm{pickup}(A), \mathrm{stack}(A, B)]$.
- **Optimal relaxed plan**: $[\mathrm{stack}(A, B), \mathrm{unstack}(B, D), \mathrm{stack}(B, C)]$.
- **Observation:** What can we say about the "search space surface" at the initial state here?
- The searchprob/initial state lies on a local minimum under h^+, together with the searchprob/successor state s where we stacked A onto B. All direct other neighbors of these two searchprob/states have a strictly higher h^+ value.

SOME RIGHTS RESERVED

18.5 Conclusion

Summary

- Heuristic search on classical search problems relies on a function h mapping searchprob/states s to an estimate $h(s)$ of their searchprob/goal state distance. Such functions h are derived by solving relaxed problems.
- In planning, the relaxed problems are generated and solved automatically. There are four known families of suitable relaxation methods: *abstractions*, *landmarks*, *critical paths*, and *ignoring deletes* (aka delete relaxation).
- The delete relaxation consists in dropping the deletes from STRIPS tasks. A relaxed plan is a plan for such a relaxed task. $h^+(s)$ is the length of an optimal relaxed plan for searchprob/state s. h^+ is NP-hard to compute.
- h^{FF} approximates h^+ by computing some, not necessarily optimal, relaxed plan. That is done by a forward pass (building a *relaxed planning graph*), followed by a backward pass (*extracting a relaxed plan*).

SOME RIGHTS RESERVED

Topics We Didn't Cover Here

- **Abstractions, Landmarks, Critical-Path Heuristics, Cost Partitions, Compilability between Heuristic Functions, Planning Competitions:**
- **Tractable fragments:** Planning sub-classes that can be solved in polynomial time. Often identified by properties of the "causal graph" and "domain transition graphs".
- **Planning as SAT:** Compile length-k bounded plan existence into satisfiability of a CNF formula φ. Extensive literature on how to obtain small φ, how to schedule different values of k, how to modify the underlying SAT solver.
- **Compilations:** Formal framework for determining whether planning formalism X is (or is not) at least as expressive as planning formalism Y.
- **Admissible pruning/decomposition methods:** Partial-order reduction, symmetry reduction, simulation-based dominance pruning, factored planning, decoupled search.
- **Hand-tailored planning:** Automatic planning is the extreme case where the computer is given no domain knowledge other than "physics". We can instead allow the user to provide search control knowledge, trading off modeling effort against search performance.
- **Numeric planning**, **temporal planning**, **planning under uncertainty** . . .

SOME RIGHTS RESERVED

Chapter 19
Searching, Planning, and Acting in the Real World

SOME RIGHTS RESERVED

Outline

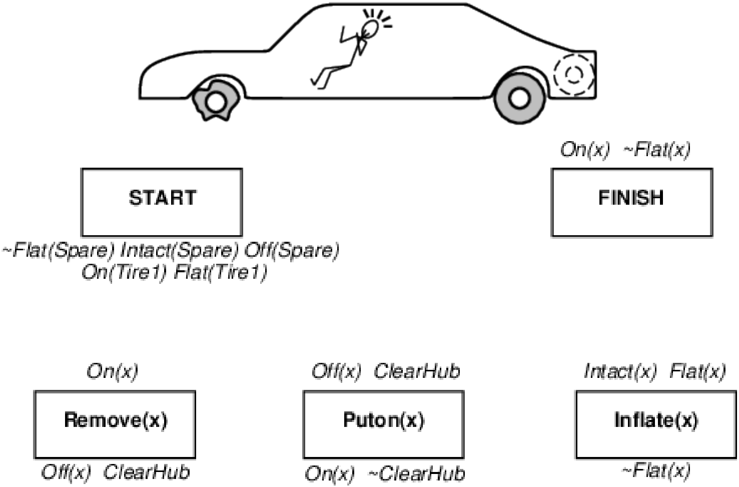
- **So Far:** we made idealizing/simplifying assumptions:
  The environment is fully observable and deterministic.
- **Outline:** In this chapter we will lift some of them
  - The real world (things go wrong)
  - Agents and Belief States
  - Conditional planning
  - Monitoring and replanning
- **Note:** The considerations in this chapter apply to both search and planning.

19.1 Introduction

SOME RIGHTS RESERVED

The real world

- **Example 1.1.** We have a flat tire – what to do?



SOME RIGHTS RESERVED

Generally: Things go wrong (in the real world)

- **Example 1.2 (Incomplete Information).**
  - Unknown preconditions, e.g., $\mathit{Intact}(\mathit{Spare})$?
  - Disjunctive effects, e.g., $\mathit{Inflate}(x)$ causes $\mathit{Inflated}(x) \vee \mathit{SlowHiss}(x) \vee \mathit{Burst}(x) \vee \mathit{BrokenPump} \vee \ldots$
- **Example 1.3 (Incorrect Information).**
  - Current state incorrect, e.g., spare NOT intact
  - Missing/incorrect effects in actions.
- **Definition 1.4.** The qualification problem in planning is that we can never finish listing all the required preconditions and possible conditional effects of actions.
- **Root Cause:** The environment is partially observable and/or non-deterministic.
- **Technical Problem:** We cannot know the "current state of the world", but search/planning algorithms are based on this assumption.
- **Idea:** Adapt search/planning algorithms to work with "sets of possible states".

SOME RIGHTS RESERVED

What can we do if things (can) go wrong?

- **One Solution:** Sensorless planning: plans that work regardless of state/outcome.
- **Problem:** Such plans may not exist! (but they often do in practice)
- **Another Solution:** Conditional plans:
  - Plan to obtain information, (observation actions)
  - Subplan for each contingency.
- **Example 1.5 (A conditional Plan).** (AAA $\widehat{=}$ ADAC)
  $[\mathit{Check}(T1), \mathrm{if}\ \mathit{Intact}(T1)\ \mathrm{then}\ \mathit{Inflate}(T1)\ \mathrm{else}\ \mathit{CallAAA}\ \mathrm{fi}]$
- **Problem:** Expensive because it plans for many unlikely cases.
- **Still another Solution:** Execution monitoring/replanning
  - Assume normal states/outcomes, check progress *during execution*, replan if necessary.
- **Problem:** Unanticipated outcomes may lead to failure. (e.g., no AAA card)
- **Observation 1.6.** *We really need a combination; plan for likely/serious eventualities, deal with others when they arise, as they must eventually.*

19.2 The Furniture Coloring Example

SOME RIGHTS RESERVED

The Furniture-Coloring Example: Specification

- **Example 2.1 (Coloring Furniture).**

Paint a chair and a table in matching colors.

- The initial state is:
  - we have two cans of paint of unknown color,
  - the color of the furniture is unknown as well,
  - only the table is in the agent's field of view.
- Searchprob/actions:
  - remove lid from can
  - paint object with paint from open can.

SOME RIGHTS RESERVED

The Furniture-Coloring Example: PDDL

- **Example 2.2 (Formalization in PDDL).**
  - The PDDL domain file is as expected (searchprob/actions below)

```
(define (domain furniture−coloring)
  (:predicates (object ?x) (can ?x) (inview ?x) (color ?x ?y))
  ...)
```

SOME RIGHTS RESERVED

The Furniture-Coloring Example: PDDL

- **Example 2.3 (Formalization in PDDL).**
  - The PDDL domain file is as expected (searchprob/actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.

```
(define (problem tc−coloring)
  (:domain furniture−objects)
  (:objects table chair c1 c2)
  (:init (object table) (object chair) (can c1) (can c2) (inview table))
  (:goal (color chair ?c) (color table ?c)))
```

SOME RIGHTS RESERVED

The Furniture-Coloring Example: PDDL

- **Example 2.4 (Formalization in PDDL).**
  - The PDDL domain file is as expected (searchprob/actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.
  - Two action schemata: "*remove can lid to open*" and "*paint with open can*"

```
(:action remove−lid
          :parameters (?x)
          :precondition (can ?x)
          :effect (open can))
(:action paint
          :parameters (?x ?y)
          :precondition (and (object ?x) (can ?y) (color ?y ?c) (open ?y))
          :effect (color ?x ?c))
```

    has a universal variable ?c for the paint action ⇝ we cannot just give paint a color argument in a partially observable environment.

  - **Sensorless Plan**: Open one can, paint chair and table in its color.

SOME RIGHTS RESERVED

The Furniture-Coloring Example: PDDL

- **Example 2.5 (Formalization in PDDL).**
  - The PDDL domain file is as expected (searchprob/actions below)
  - The PDDL problem file has a “free” variable ?c for the (undetermined) joint color.
  - Two action schemata: “*remove can lid to open*” and “*paint with open can*” has a universal variable ?c for the paint action ⇝ we cannot just give paint a color argument in a partially observable environment.
  - **Sensorless Plan**: Open one can, paint chair and table in its color.
  - **Note**: Contingent planning can create better plans, but needs perception
  - Two percept schemata: “*color of an object*” and “*color in a can*”

```
(:percept color
          :parameters (?x ?c)
          :precondition (and (object ?x) (inview ?x)))
(:percept can−color
          :parameters (?x ?c)
          :precondition (and (can ?x) (inview ?x) (open ?x)))
```

To perceive the color of an object, it must be in view, a can must also be open.
Note: In a fully observable world, the percepts would not have preconditions.

SOME RIGHTS RESERVED

The Furniture-Coloring Example: PDDL

- **Example 2.6 (Formalization in PDDL).**
  - The PDDL domain file is as expected (searchprob/actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.
  - Two action schemata: "*remove can lid to open*" and "*paint with open can*" has a universal variable ?c for the paint action ⇝ we cannot just give paint a color argument in a partially observable environment.
  - **Sensorless Plan**: Open one can, paint chair and table in its color.
  - **Note**: Contingent planning can create better plans, but needs perception
  - Two percept schemata: "*color of an object*" and "*color in a can*"
  - An action schema: "*look at an object*" that causes it to come into view.

```
(:action lookat
        :parameters (?x)
        :precond: (and (inview ?y) and (notequal ?x ?y))
        :effect (and (inview ?x) (not (inview ?y))))
```

SOME RIGHTS RESERVED

The Furniture-Coloring Example: PDDL

- **Example 2.7 (Formalization in PDDL).**
  - The PDDL domain file is as expected (searchprob/actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.
  - Two action schemata: "*remove can lid to open*" and "*paint with open can*" has a universal variable ?c for the paint action ⇜ we cannot just give paint a color argument in a partially observable environment.
  - **Sensorless Plan**: Open one can, paint chair and table in its color.
  - **Note**: Contingent planning can create better plans, but needs perception
  - Two percept schemata: "*color of an object*" and "*color in a can*"
  - An action schema: "*look at an object*" that causes it to come into view.
  - **Contingent Plan**:
    1. look at furniture to determine color, if same ⇝ done.
    2. else, look at open and look at paint in cans
    3. if paint in one can is the same as an object, paint the other with this color
    4. else paint both in any color

SOME RIGHTS RESERVED

19.3 Searching/Planning with Non-Deterministic Actions

SOME RIGHTS RESERVED

Conditional Plans

- **Definition 3.1.** Conditional plans extend the possible actions in plans by conditional steps that execute sub plans conditionally whether $K+P\models C$, where $K+P$ is the current knowledge base + the percepts.
- **Definition 3.2.** Conditional plans can contain
  - conditional step: $[\ldots,\mathrm{if}\ C\ \mathrm{then}\ Plan_A\ \mathrm{else}\ Plan_B\ \mathrm{fi},\ldots]$,
  - while step: $[\ldots,\mathrm{while}\ C\ \mathrm{do}\ Plan\ \mathrm{done},\ldots]$, and
  - the empty plan $\emptyset$ to make modeling easier.
- **Definition 3.3.** If the possible percepts are limited to determining the current state in a conditional plan, then we speak of a contingency plan.
- **Note:** Need *some plan* for *every possible percept*! Compare to

  game playing: *some response* for *every opponent* move.
  backchaining: *some rule* such that *every premise* satisfied.
- **Idea:** Use an AND–OR tree search (very similar to backward chaining algorithm)

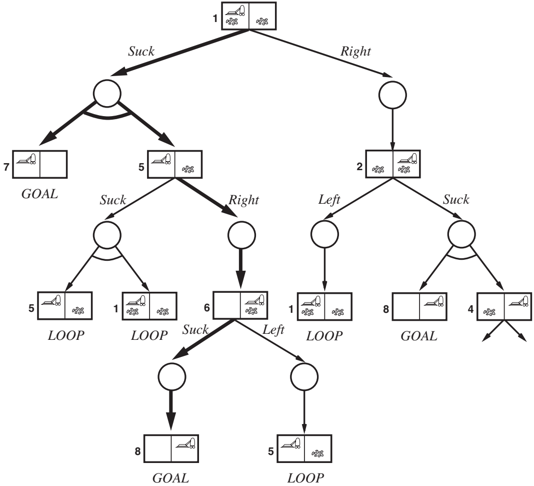
SOME RIGHTS RESERVED

Contingency Planning: The Erratic Vacuum Cleaner

- **Example 3.4 (Erratic vacuum world).**

A variant *suck* action:
if square is

- *dirty*: clean the square, sometimes remove dirt in adjacent square.
- *clean*: sometimes deposits dirt on the carpet.



Solution: [$suck$, if $State = 5$ then [$right, suck$] else [] fi]

SOME RIGHTS RESERVED

Conditional AND-OR Search (Data Structure)

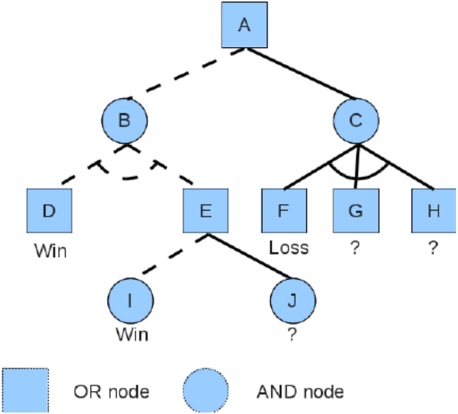
- **Idea:** Use AND-OR trees as data structures for representing problems (or goals) that can be reduced to to conjunctions and disjunctions of subproblems (or subgoals).
- **Definition 3.5.** An AND-OR graph is a is a graph whose non-terminal nodes are partitioned into AND nodes and OR nodes. A valuation of an AND-OR graph T is an assignment of T or F to the nodes of T. A valuation of the terminal nodes of T can be extended by all nodes recursively: Assign T to an
  - OR node, iff at least one of its children is T.
  - AND node, iff all of its children are T.

  A solution for T is a valuation that assigns T to the initial nodes of T.
- **Idea:** A planning task with non deterministic actions generates a AND-OR graph T. A solution that assigns T to a terminal node, iff it is a goal node. Corresponds to a conditional plan.

SOME RIGHTS RESERVED

Conditional AND-OR Search (Example)

- **Definition 3.6.** An AND-OR tree is a AND-OR graph that is also a tree.
  Notation: AND nodes are written with arcs connecting the child edges.
- **Example 3.7 (An AND-OR-tree).**



SOME RIGHTS RESERVED

Conditional AND-OR Search (Algorithm)

- **Definition 3.8.** AND-OR search is an algorithm for searching AND–OR graphs generated by nondeterministic environments.

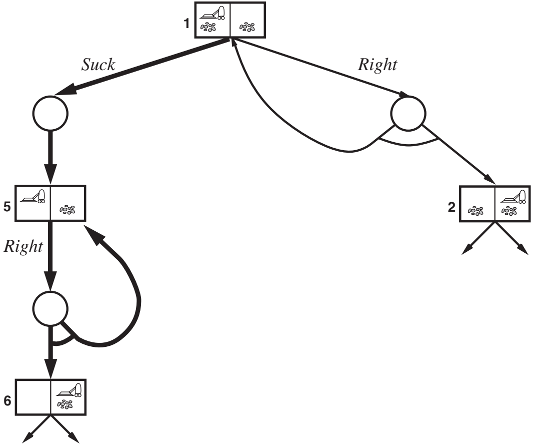
```
function AND/OR−GRAPH−SEARCH($prob$) returns a conditional plan, or fail
  OR−SEARCH($prob$.INITIAL−STATE, $prob$, [])
function OR−SEARCH($state$,$prob$,$path$) returns a conditional plan, or fail
  if $prob$.GOAL−TEST($state$) then return the empty plan
  if $state$ is on $path$ then return fail
  for each $action$ in $prob$.ACTIONS($state$) do
    $plan$ := AND−SEARCH(RESULTS($state$,$action$),$prob$,[$state$ | $path$])
    if $plan \neq$ fail then return [$action$ | $plan$]
  return fail
function AND−SEARCH($states$,$prob$,$path$) returns a conditional plan, or fail
  for each $s_i$ in $states$ do
    $p_i$ := OR−SEARCH($s_i$,$prob$,$path$)
    if $p_i$ = fail then return fail
    return [if $s_1$ then $p_1$ else if $s_2$ then $p_2$ else ... if $s_{n-1}$ then $p_{n-1}$ else $p_n$]
```

- **Cycle Handling:** If a state has been seen before $\leadsto$ **fail**
  - **fail** does not mean "*there is no solution*", but
  - "*if there is a non-cyclic solution, then it is reachable by an earlier incarnation*"!

SOME RIGHTS RESERVED

The Slippery Vacuum Cleaner (try, try, try, . . . try again)

- **Example 3.9 (Slippery Vacuum World).**



Moving sometimes fails
⇝ AND-OR graph

Two possible solutions (depending on what our plan language allows)

- $[L_1:\ \mathit{left}, \text{if } \mathit{AtR} \text{ then } L_1 \text{ else } [\text{if } \mathit{CleanL} \text{ then } \emptyset \text{ else } \mathit{suck} \text{ fi}] \text{ fi}]$ or
- $[\text{while } \mathit{AtR} \text{ do } [\mathit{left}] \text{ done}, \text{if } \mathit{CleanL} \text{ then } \emptyset \text{ else } \mathit{suck} \text{ fi}]$

- We have an infinite loop but plan eventually works unless action always fails.

19.4 Agent Architectures based on Belief States

SOME RIGHTS RESERVED

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!

SOME RIGHTS RESERVED

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 4.2.** A model-based agent has a world model consisting of
  - a belief state that has information about the possible states the world may be in,
  - a sensor model that updates the belief state based on sensor information, and
  - a transition model that updates the belief state based on actions.

SOME RIGHTS RESERVED

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 4.3.** A model-based agent has a world model consisting of
  - a belief state that has information about the possible states the world may be in,
  - a sensor model that updates the belief state based on sensor information, and
  - a transition model that updates the belief state based on actions.
- **Idea:** The agent environment determines what the world model can be.

SOME RIGHTS RESERVED

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 4.4.** A model-based agent has a world model consisting of
    - a belief state that has information about the possible states the world may be in,
    - a sensor model that updates the belief state based on sensor information, and
    - a transition model that updates the belief state based on actions.
- **Idea:** The agent environment determines what the world model can be.
- In a fully observable, deterministic environment,
    - we can observe the initial state and subsequent states are given by the actions alone.
    - Thus the belief state is a singleton (we call its sole member the world state) and the transition model is a function from states and actions to states: a transition function.

SOME RIGHTS RESERVED

World Models by Agent Type in AI-1

- **Search-based Agents:** In a fully observable, deterministic environment
  - goal-based agent with world state $\widehat{=}$ "current state"
  - no inference. (goal $\widehat{=}$ goal state from search problem)
- **CSP-based Agents:** In a fully observable, deterministic environment
  - goal-based agent withworld state $\widehat{=}$ constraint network,
  - inference $\widehat{=}$ constraint propagation. (goal $\widehat{=}$ satisfying assignment)
- **Logic-based Agents:** In a fully observable, deterministic environment
  - model-based agent with world state $\widehat{=}$ logical formula
  - inference $\widehat{=}$ e.g. DPLL or resolution.
- **Planning Agents:** In a fully observable, deterministic, environment
  - goal-based agent with world state $\widehat{=}$ PL0, transition model $\widehat{=}$ STRIPS,
  - inference $\widehat{=}$ state/plan space search. (goal: complete plan/execution)

SOME RIGHTS RESERVED

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\leadsto$ generalize the transition function to a transition relation.

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\rightsquigarrow$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\leadsto$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)
- In a deterministic, but partially observable environment,
  - the belief state must deal with a set of possible states.
  - we can use transition functions.
  - We need a sensor model, which predicts the influence of percepts on the belief state – during update.

SOME RIGHTS RESERVED

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\leadsto$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)
- In a deterministic, but partially observable environment,
  - the belief state must deal with a set of possible states.
  - we can use transition functions.
  - We need a sensor model, which predicts the influence of percepts on the belief state – during update.
- In a stochastic, partially observable environment,
  - mix the ideas from the last two. (sensor model + transition relation)

SOME RIGHTS RESERVED

Preview: New World Models (Belief) $\rightsquigarrow$ new Agent Types

- **Probabilistic Agents:** In a partially observable environment
  - belief state $\widehat{=}$ Bayesian networks,
  - inference $\widehat{=}$ probabilistic inference.

SOME RIGHTS RESERVED

Preview: New World Models (Belief) $\rightsquigarrow$ new Agent Types

- **Probabilistic Agents:** In a partially observable environment
  - belief state $\widehat{=}$ Bayesian networks,
  - inference $\widehat{=}$ probabilistic inference.
- **Decision-Theoretic Agents:** In a partially observable, stochastic environment
  - belief state + transition model $\widehat{=}$ decision networks,
  - inference $\widehat{=}$ maximizing expected utility.
- We will study them in detail this semester.

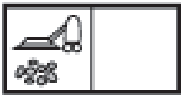
SOME RIGHTS RESERVED

19.5 Searching/Planning without Observations

SOME RIGHTS RESERVED

Conformant/Sensorless Planning

- **Definition 5.1.** Conformant or sensorless planning tries to find plans that work without any sensing. (not even the initial state)



- **Example 5.2 (Sensorless Vacuum Cleaner World).**

Searchprob/states	integer dirt and robot locations
Searchprob/actions	*left*, *right*, *suck*, *noOp*
Searchprob/goal states	*notdirty*?

- **Observation 5.3.** *In a sensorless world we do not know the initial state.(or any state after)*
- **Observation 5.4.** *Sensorless planning must search in the space of belief states (sets of possible actual states).*
- **Example 5.5 (Searching the Belief State Space).**
  - Start in $\{1,2,3,4,5,6,7,8\}$
  - Solution: $[right, suck, left, suck]$

$$\begin{array}{ll} right & \rightarrow \{2,4,6,8\} \\ suck & \rightarrow \{4,8\} \\ left & \rightarrow \{3,7\} \\ suck & \rightarrow \{7\} \end{array}$$

SOME RIGHTS RESERVED

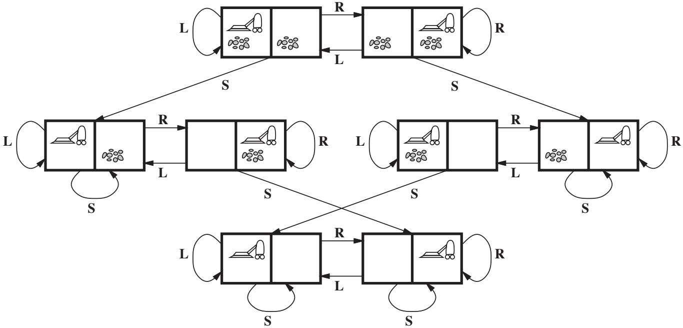
Search in the Belief State Space: Let's Do the Math

- **Recap:** We describe an search problem $\Pi := \langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$ via its states $\mathcal{S}$, actions $\mathcal{A}$, and transition model $\mathcal{T}\colon\mathcal{A}\times\mathcal{S}\to\mathcal{P}(\mathcal{A})$, goal states $\mathcal{G}$, and initial state $\mathcal{I}$.
- **Problem:** What is the corresponding sensorless problem?
- **Let's think:** Let $\Pi := \langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$ be a (physical) problem
  - States $\mathcal{S}^b$: The belief states are the $2^{|\mathcal{S}|}$ subsets of $\mathcal{S}$.
  - The initial state $\mathcal{I}^b$ is just $\mathcal{S}$ (no information)
  - Goal states $\mathcal{G}^b := \{S\in\mathcal{S}^b \mid S\subseteq\mathcal{G}\}$ (all possible states must be physical goal states)
  - Actions $\mathcal{A}^b$: we just take $\mathcal{A}$. (that's the point!)
  - Transition model $\mathcal{T}^b\colon\mathcal{A}^b\times\mathcal{S}^b\to\mathcal{P}(\mathcal{A}^b)$: i.e. what is $\mathcal{T}^b(a,S)$ for $a\in\mathcal{A}$ and $S\subseteq\mathcal{S}$? This is slightly tricky as a need not be applicable to all $s\in S$.
    1. if actions are harmless to the environment, take $\mathcal{T}^b(a,S) := \bigcup_{s\in S}\mathcal{T}(a,s)$.
    2. if not, better take $\mathcal{T}^b(a,S) := \bigcap_{s\in S}\mathcal{T}(a,s)$. (the safe bet)
- **Observation 5.6.** *In belief-state space the problem is always fully observable!*

SOME RIGHTS RESERVED

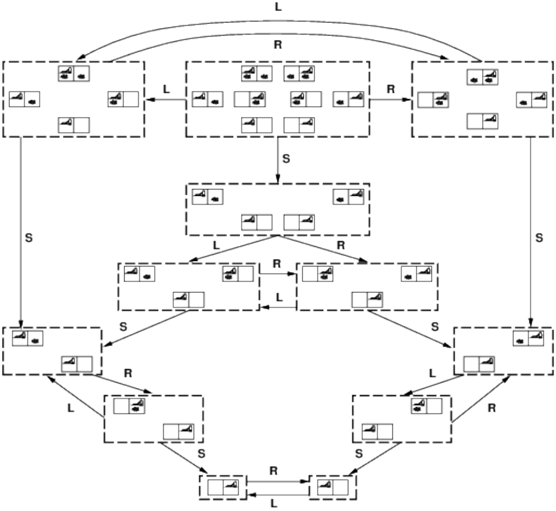
State Space vs. Belief State Space

- **Example 5.7 (State/Belief State Space in the Vacuum World).** In the vacuum world all actions are always applicable (1./2. equal)



State Space vs. Belief State Space

- **Example 5.8 (State/Belief State Space in the Vacuum World).** In the vacuum world all actions are always applicable (1./2. equal)



SOME RIGHTS RESERVED

Evaluating Conformant Planning

- **Upshot:** We can build belief-space problem formulations automatically,
  - but they are exponentially bigger in theory, in practice they are often similar;
  - e.g. 12 reachable belief states out of $2^8 = 256$ for vacuum example.
- **Problem:** Belief states are HUGE; e.g. initial belief state for the 10×10 vacuum world contains $100 \cdot 2^{100} \approx 10^{32}$ physical states
- **Idea:** Use planning techniques: compact descriptions for
  - belief states; e.g. "*all*" for initial state or "*not leftmost column*" after *left*.
  - actions as belief state to belief state operations.
- **This actually works:** Therefore we talk about conformant planning!

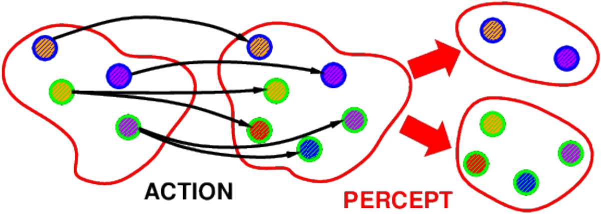
SOME RIGHTS RESERVED

19.6 Searching/Planning with Observation

SOME RIGHTS RESERVED

Conditional planning (Motivation)

- **Note:** So far, we have never used the agent's sensors.
  - In , since the environment was observable and deterministic we could just use offline planning.
  - In because we chose to.
- **Note:** If the world is nondeterministic or partially observable then percepts usually provide information, i.e., split up the belief state



- **Idea:** This can systematically be used in search/planning via belief-state search, but we need to rethink/specialize the Transition model.

SOME RIGHTS RESERVED

A Transition Model for Belief-State Search

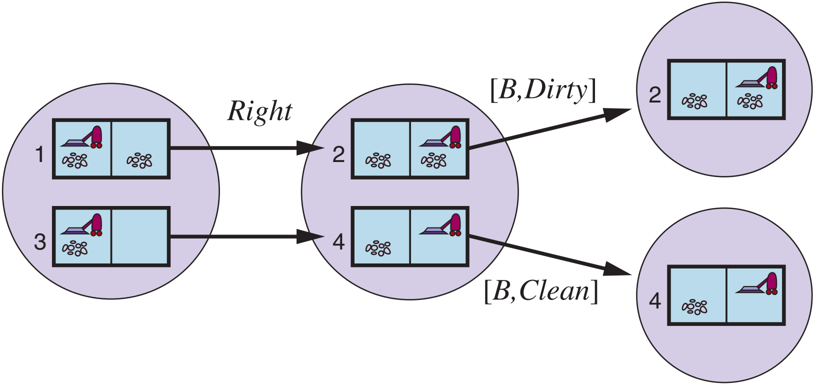
- We extend the ideas from slide 660 to include partial observability.
- **Definition 6.1.** Given a (physical) search problem $\Pi := \langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, we define the belief state search problem induced by Π to be $\langle\mathcal{P}(\mathcal{S}),\mathcal{A},\mathcal{T}^b,\mathcal{S},\{S\in\mathcal{S}^b\,|\,S\subseteq\mathcal{G}\}\rangle$, where the transition model $\mathcal{T}^b$ is constructed in three stages:
  - The prediction stage: given a belief state b and an action a we define $\widehat{b} := \mathrm{PRED}(b,a)$ for some function $\mathrm{PRED}\colon \mathcal{P}(\mathcal{S})\times\mathcal{A}\to\mathcal{P}(\mathcal{S})$.
  - The observation prediction stage determines the set of possible percepts that could be observed in the predicted belief state: $\mathrm{PossPERC}(\widehat{b}) = \{\mathrm{PERC}(s)\,|\,s\in\widehat{b}\}$.
  - The update stage determines, for each possible percept, the resulting belief state: $\mathrm{UPDATE}(\widehat{b},o) := \{s\,|\,o=\mathrm{PERC}(s) \text{ and } s\in\widehat{b}\}$

  The functions PRED and PERC are the main parameters of this model. We define $\mathrm{RESULT}(b,a) := \{\mathrm{UPDATE}(\mathrm{PRED}(b,a),o)\,|\,\mathrm{PossPERC}(\mathrm{PRED}(b,a))\}$
- **Observation 6.2.** *We always have* $\mathrm{UPDATE}(\widehat{b},o)\subseteq\widehat{b}$.
- **Observation 6.3.** *If sensing is deterministic, belief states for different possible percepts are disjoint, forming a partition of the original predicted belief state.*

SOME RIGHTS RESERVED

Example: Local Sensing Vacuum Worlds

- **Example 6.4 (Transitions in the Vacuum World).** Deterministic World:

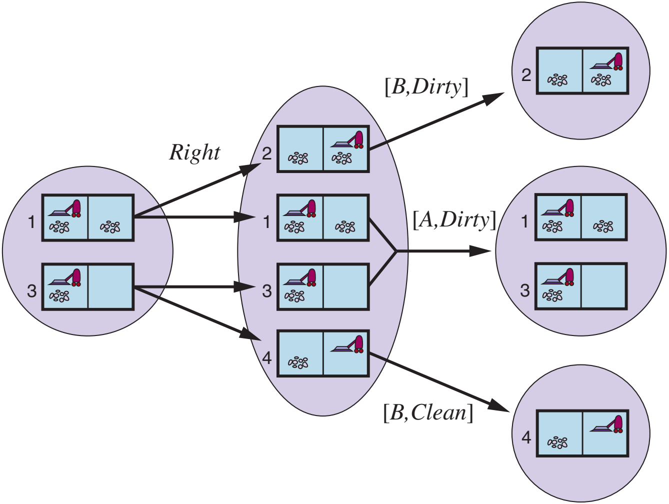


The action *Right* is deterministic, sensing disambiguates to singletons

SOME RIGHTS RESERVED

Example: Local Sensing Vacuum Worlds

- **Example 6.5 (Transitions in the Vacuum World).** Slippery World:

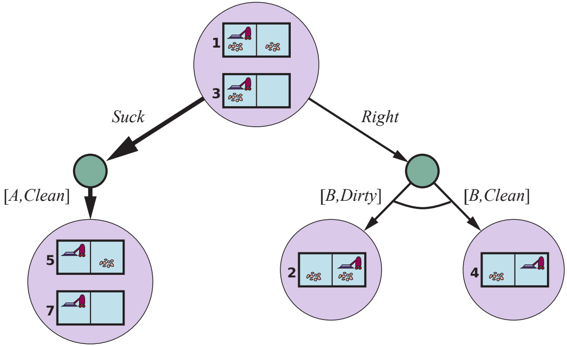


The action *Right* is non-deterministic, sensing disambiguates somewhat

SOME RIGHTS RESERVED

Belief-State Search with Percepts

- **Observation:** The belief-state transition model induces an AND-OR graph.
- **Idea:** Use AND-OR search in non deterministic environments.
- **Example 6.6.** AND-OR graph for initial percept $[A, Dirty]$.



Solution: $[Suck, Right, \text{if } Bstate = \{6\} \text{ then } Suck \text{ else } [] \text{ fi}]$

- **Note:** Belief-state-problem ⇝ conditional step tests on belief-state percept (plan would not be executable in a partially observable environment otherwise)

SOME RIGHTS RESERVED

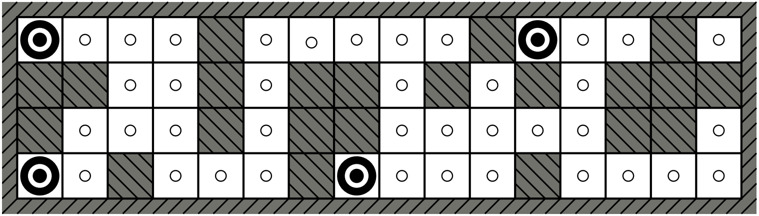
Example: Agent Localization

- **Example 6.7.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\rightsquigarrow \widehat{b}_1$ all possible locations.

SOME RIGHTS RESERVED

Example: Agent Localization

- **Example 6.8.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\rightsquigarrow \widehat{b_1}$ all possible locations.
  2. Initial percept: *NWS* (walls north, west, and south) $\rightsquigarrow \widehat{b_2} = \mathrm{UPDATE}(\widehat{b_1}, NWS)$



SOME RIGHTS RESERVED

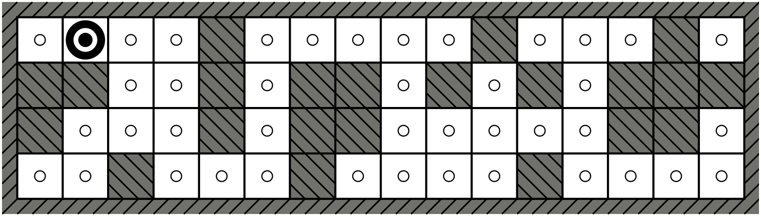
Example: Agent Localization

- **Example 6.9.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\rightsquigarrow \widehat{b}_1$ all possible locations.
  2. Initial percept: *NWS* (walls north, west, and south) $\rightsquigarrow \widehat{b}_2 = \mathrm{UPDATE}(\widehat{b}_1, NWS)$
  3. Agent executes *Move* $\rightsquigarrow \widehat{b}_3 = \mathrm{PRED}(\widehat{b}_2, Move) =$ "*one step away from these*".

SOME RIGHTS RESERVED

Example: Agent Localization

- **Example 6.10.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\rightsquigarrow$ $\widehat{b_1}$ all possible locations.
  2. Initial percept: *NWS* (walls north, west, and south) $\rightsquigarrow$ $\widehat{b_2} = \mathrm{UPDATE}(\widehat{b_1}, NWS)$
  3. Agent executes *Move* $\rightsquigarrow$ $\widehat{b_3} = \mathrm{PRED}(\widehat{b_2}, Move)$ = "*one step away from these*".
  4. Next percept: *NS* $\rightsquigarrow$ $\widehat{b_4} = \mathrm{UPDATE}(\widehat{b_3}, NS)$



SOME RIGHTS RESERVED

Example: Agent Localization

- **Example 6.11.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\leadsto \widehat{b}_1$ all possible locations.
  2. Initial percept: *NWS* (walls north, west, and south) $\leadsto \widehat{b}_2 = \mathrm{UPDATE}(\widehat{b}_1, \mathit{NWS})$
  3. Agent executes *Move* $\leadsto \widehat{b}_3 = \mathrm{PRED}(\widehat{b}_2, \mathit{Move})$ = "*one step away from these*".
  4. Next percept: *NS* $\leadsto \widehat{b}_4 = \mathrm{UPDATE}(\widehat{b}_3, \mathit{NS})$

  All in all, $\widehat{b}_4 = \mathrm{UPDATE}(\mathrm{PRED}(\mathrm{UPDATE}(\widehat{b}_1, \mathit{NWS}), \mathit{Move}), \mathit{NS})$ localizes the agent.
- **Observation:** PRED enlarges the belief state, while UPDATE shrinks it again.

SOME RIGHTS RESERVED

Contingent Planning

- **Definition 6.12.** The generation of plan with conditional branching based on percepts is called contingent planning, solutions are called contingent plans.
- Appropriate for partially observable or non-deterministic environments.
- **Example 6.13.** Continuing 2.1.

  One of the possible contingent plan is

```
((lookat table) (lookat chair)
  (if (and (color table c) (color chair c)) (noop)
    ((removelid c1) (lookat c1) (removelid c2) (lookat c2)
     (if (and (color table c) (color can c)) ((paint chair can))
       (if (and (color chair c) (color can c)) ((paint table can))
         ((paint chair c1) (paint table c1)))))))
```

- **Note:** Variables in this plan are existential; e.g. in
  - line 2: If there is come joint color c of the table and chair ⇝ done.
  - line 4/5: Condition can be satisfied by $[c_1/can]$ or $[c_2/can]$ ⇝ instantiate accordingly.
- **Definition 6.14.** During plan execution the agent maintains the belief state b, chooses the branch depending on whether $b \vDash c$ for the condition c.
- **Note:** The planner must make sure $b \vDash c$ can always be decided.

SOME RIGHTS RESERVED

Contingent Planning: Calculating the Belief State

- **Problem:** How do we compute the belief state?
- **Recall:** Given a belief state b, the new belief state $\widehat{b}$ is computed based on prediction with the action a and the refinement with the percept p.
- **Here:**
  Given an action a and percepts $p = p_1 \wedge \ldots \wedge p_n$, we have
  - $\widehat{b} = b \backslash \mathrm{del}_a \cup \mathrm{add}_a$ (as for the sensorless agent)
  - If $n = 1$ and (:percept p_1 :precondition c) is the only percept axiom, also add p and c to $\widehat{b}$. (add c as otherwise p impossible)
  - If $n > 1$ and (:percept p_i :precondition c_i) are the percept axioms, also add p and $c_1 \vee \ldots \vee c_n$ to $\widehat{b}$. (belief state no longer conjunction of literals ☹)
- **Idea:** Given such a mechanism for generating (exact or approximate) updated belief states, we can generate contingent plans with an extension of AND-OR search over belief states.
- **Extension:** This also works for non-deterministic searchprob/actions: we extend the representation of effects to disjunctions.

SOME RIGHTS RESERVED

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)
- Works on all common devices (mobile phone, notebook, etc.)
- Is in English; takes about 10 - 20 min
  depending on proficiency in english and using ALeA

SOME RIGHTS RESERVED

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)
- Works on all common devices (mobile phone, notebook, etc.)
- Is in English; takes about 10 - 20 min
  depending on proficiency in english and using ALeA
- Questions about how ALeA is used, what it is like usig ALeA, and questions about demography

SOME RIGHTS RESERVED

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)
- Works on all common devices (mobile phone, notebook, etc.)
- Is in English; takes about 10 - 20 min
  depending on proficiency in english and using ALeA
- Questions about how ALeA is used, what it is like usig ALeA, and questions about demography
- Token is generated at the end of the survey (SAVE THIS CODE!)
  - Completed survey count as a successfull prepquiz in AI1!
  - Look for Quiz 15 in the usual place (single question)
  - just submit the token to get full points
  - The token can also be used to exercise the rights of the GDPR.

SOME RIGHTS RESERVED

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)
- Works on all common devices (mobile phone, notebook, etc.)
- Is in English; takes about 10 - 20 min depending on proficiency in english and using ALeA
- Questions about how ALeA is used, what it is like usig ALeA, and questions about demography
- Token is generated at the end of the survey (SAVE THIS CODE!)
  - Completed survey count as a successfull prepquiz in AI1!
  - Look for Quiz 15 in the usual place (single question)
  - just submit the token to get full points
  - The token can also be used to exercise the rights of the GDPR.
- Survey has no timelimit and is free, anonymous, can be paused and continued later on and can be cancelled.

SOME RIGHTS RESERVED

Find the Survey Here

https://ddi-survey.cs.fau.de/limesurvey/index.php/667123?lang=en

This URL will also be posted on the forum tonight.

SOME RIGHTS RESERVED

19.7 Online Search

Online Search and Replanning

- **Note:** So far we have concentrated on offline problem solving, where the agent only acts (plan execution) after search/planning terminates.
- **Recall:** In online problem solving an agent interleaves computation and action: it computes one action at a time based on incoming perceptions.
- Online problem solving is helpful in
    - dynamic or semidynamic environments. (long computation times can be harmful)
    - stochastic environments. (solve contingencies only when they arise)
- Online problem solving is necessary in unknown environments $\leadsto$ exploration problem.

SOME RIGHTS RESERVED

Online Search Problems

- **Observation:** Online problem solving even makes sense in deterministic, fully observable environments.
- **Definition 7.1.** A online search problem consists of a set S of states, and
  - a function $\mathrm{Actions}(s)$ that returns a list of actions allowed in state s.
  - the step cost function c, where $c(s,a,s')$ is the cost of executing action a in state s with outcome s'. (cost unknown before executing a)
  - a goal test $\mathrm{Goal\ Test}$.
- **Note:** We can only determine $\mathrm{RESULT}(s,a)$ by being in s and executing a.
- **Definition 7.2.** The competitive ratio of an online problem solving agent is the quotient of
  - offline performance, i.e. cost of optimal solutions with full information and
  - online performance, i.e. the actual cost induced by online problem solving.

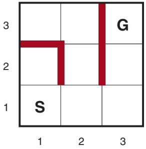
SOME RIGHTS RESERVED

Online Search Problems (Example)

- **Example 7.3 (A simple maze problem).**

  The agent starts at S and must reach G but knows nothing of the environment. In particular not that

  - $\mathrm{Up}(1,1)$ results in $(1{,}2)$ and
  - $\mathrm{Down}(1,1)$ results in $(1{,}1)$ (i.e. back)



SOME RIGHTS RESERVED

Online Search Obstacles (Dead Ends)

- **Definition 7.4.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.

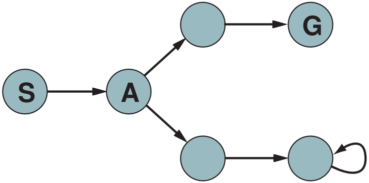
Online Search Obstacles (Dead Ends)

- **Definition 7.10.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.
- **Observation 7.11.** *No online algorithm can avoid dead ends in all state spaces.*

SOME RIGHTS RESERVED

Online Search Obstacles (Dead Ends)

- **Definition 7.16.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.
- **Observation 7.17.** *No online algorithm can avoid dead ends in all state spaces.*
- **Example 7.18.** Two state spaces that lead an online agent into dead ends:



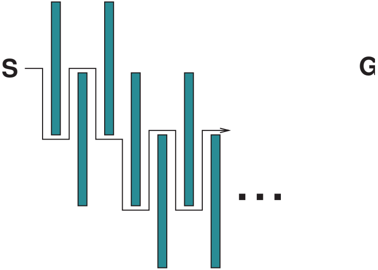
  Any agent will fail in at least one of the spaces.
- **Definition 7.19.** We call 7.6 an adversary argument.

SOME RIGHTS RESERVED

Online Search Obstacles (Dead Ends)

- **Definition 7.22.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.
- **Observation 7.23.** *No online algorithm can avoid dead ends in all state spaces.*
- **Example 7.24.** Two state spaces that lead an online agent into dead ends: Any agent will fail in at least one of the spaces.
- **Definition 7.25.** We call 7.6 an adversary argument.
- **Example 7.26.** Forcing an online agent into an arbitrarily inefficient route:

Whichever choice the agent makes the adversary can block with a long, thin wall



Online Search Obstacles (Dead Ends)

- **Definition 7.28.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.
- **Observation 7.29.** *No online algorithm can avoid dead ends in all state spaces.*
- **Example 7.30.** Two state spaces that lead an online agent into dead ends: Any agent will fail in at least one of the spaces.
- **Definition 7.31.** We call 7.6 an adversary argument.
- **Example 7.32.** Forcing an online agent into an arbitrarily inefficient route:
- **Observation:** Dead ends are a real problem for robots: ramps, stairs, cliffs, . . .
- **Definition 7.33.** A state space is called safely explorable, iff a goal state is reachable from every reachable state.
- We will always assume this in the following.

SOME RIGHTS RESERVED

Online Search Agents

- **Observation:** Online and offline search algorithms differ considerably:
  - For an offline agent, the environment is visible a priori.
  - An online agent builds a "map" of the environment from percepts in visited states.

  Therefore, e.g. A^* can expand any node in the fringe, but an online agent must go there to explore it.
- **Intuition:** It seems best to expand nodes in "local order" to avoid spurious travel.
- **Idea:** Depth first search seems a good fit. (must only travel for backtracking)

SOME RIGHTS RESERVED

Online DFS Search Agent

- **Definition 7.34.** The online depth first search algorithm:

```
function ONLINE−DFS−AGENT($s'$) returns an action
  inputs: $s'$, a percept that identifies the current state
  persistent: $result$, a table mapping $(s, a)$ to $s'$, initially empty
      $untried$, a table mapping $s$ to a list of untried actions
      $unbacktracked$, a table mapping $s$ to a list backtracks not tried
      $s$, $a$, the previous state and action, initially null
  if $\mathrm{Goal\ Test}(s')$ then return stop
  if $s' \notin untried$ then $untried[s'] := \mathrm{Actions}(s')$
  if s is not null then
       $result[s, a] := s'$
       add $s$ to the front of $unbacktracked[s']$
  if $untried[s']$ is empty then
       if $unbacktracked[s']$ is empty then return stop
       else $a :=$ an action $b$ such that $result[s', b] = pop(unbacktracked[s'])$
   else $a := pop(untried[s'])$
   $s := s'$
   return $a$
```

- **Note:** *result* is the "environment map" constructed as the agent explores.

SOME RIGHTS RESERVED

19.8 Replanning and Execution Monitoring

SOME RIGHTS RESERVED

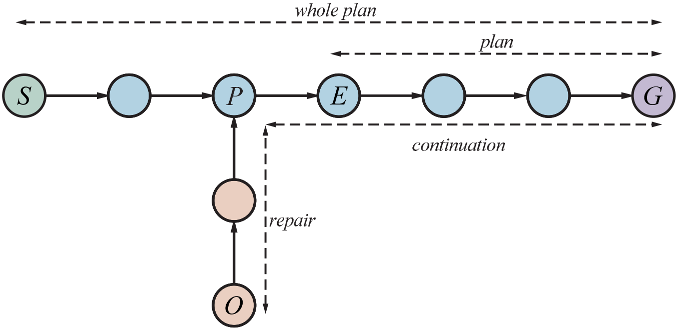
Replanning (Ideas)

- **Idea:** We can turn a planner P into an online problem solver by adding an action $\mathrm{RePlan}(g)$ without preconditions that re-starts P in the current state with goal g.
- **Observation:** Replanning induces a tradeoff between pre-planning and re-planning.
- **Example 8.1.** The plan $[\mathrm{RePlan}(g)]$ is a (trivially) complete plan for any goal g. (not helpful)
- **Example 8.2.** A plan with sub-plans for every contingency (e.g. what to do if a meteor strikes) may be too costly/large. (wasted effort)
- **Example 8.3.** But when a tire blows while driving into the desert, we want to have water pre-planned. (due diligence against catastrophies)
- **Observation:** In stochastic or partially observable environments we also need some form of execution monitoring to determine the need for replanning (plan repair).

SOME RIGHTS RESERVED

Replanning for Plan Repair

- **Generally:** Replanning when the agent's model of the world is incorrect.
- **Example 8.4 (Plan Repair by Replanning).** Given a plan from S to G.



  - The agent executes *wholeplan* step by step, monitoring the rest (*plan*).
  - After a few steps the agent expects to be in E, but observes state O.
  - Replanning: by calling the planner recursively
    - find state P in *wholeplan* and a plan *repair* from O to P. (P may be G)
    - minimize the cost of *repair* + *continuation*

SOME RIGHTS RESERVED

Factors in World Model Failure $\rightsquigarrow$ Monitoring

- **Generally:** The agent's world model can be incorrect, because
  - an action has a missing precondition (need a screwdriver for remove—lid)
  - an action misses an effect (painting a table gets paint on the floor)
  - it is missing a state variable (amount of paint in a can: no paint $\rightsquigarrow$ no color)
  - no provisions for exogenous events (someone knocks over a paint can)
- **Observation:** Without a way for monitoring for these, planning is very brittle.
- **Definition 8.5.** There are three levels of execution monitoring: before executing an action
  - action monitoring checks whether all preconditions still hold.
  - plan monitoring checks that the remaining plan will still succeed.
  - goal monitoring checks whether there is a better set of goals it could try to achieve.
- **Note:** 8.4 was a case of action monitoring leading to replanning.

SOME RIGHTS RESERVED

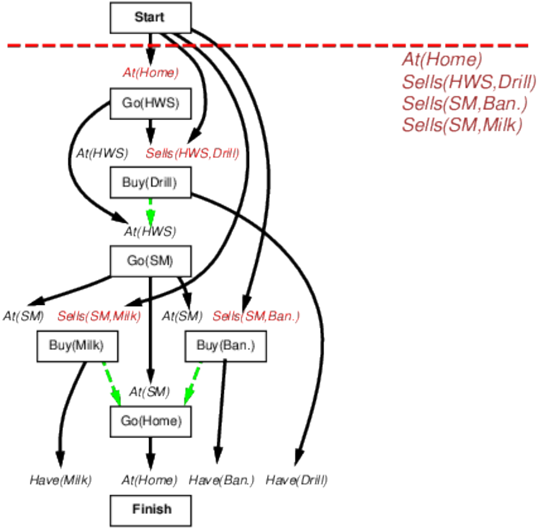
Integrated Execution Monitoring and Planning

- **Problem:** Need to upgrade planing data structures by bookkeeping for execution monitoring.
- **Observation:** With their causal links, partially ordered plans already have most of the infrastructure for action monitoring:
  Preconditions of remaining plan
  $\widehat{=}$ all preconditions of remaining steps not achieved by remaining steps
  $\widehat{=}$ all causal link "crossing current time point"
- **Idea:** On failure, resume planning (e.g. by POP) to achieve open conditions from current state.
- **Definition 8.6.** IPEM (Integrated Planning, Execution, and Monitoring):
  - keep updating *Start* to match current state
  - links from searchprob/actions replaced by links from *Start* when done

SOME RIGHTS RESERVED

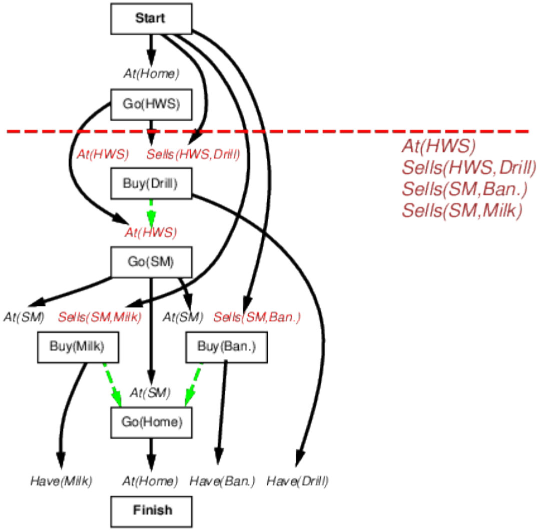
Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



Execution Monitoring Example

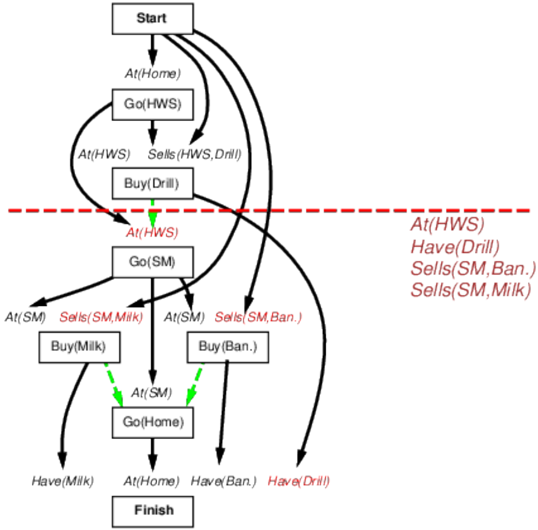
- **Example 8.8 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



SOME RIGHTS RESERVED

Execution Monitoring Example

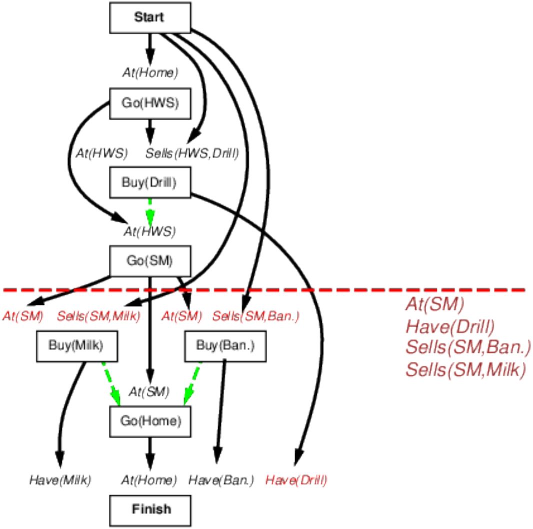
- **Example 8.9 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



SOME RIGHTS RESERVED

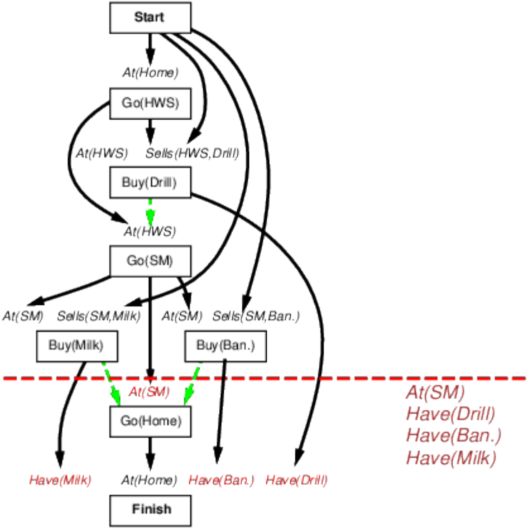
Execution Monitoring Example

- **Example 8.10 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



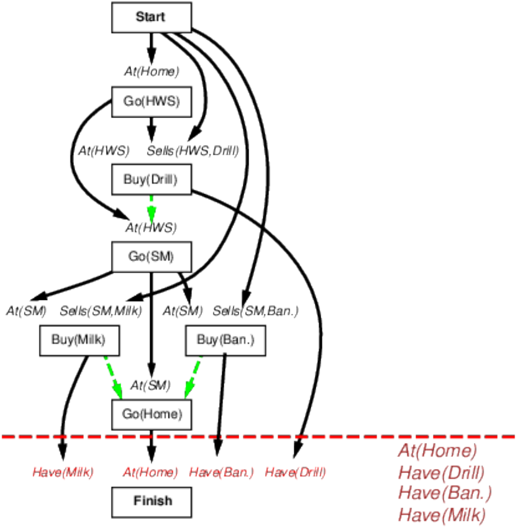
Execution Monitoring Example

- **Example 8.11 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



Execution Monitoring Example

- **Example 8.12 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



SOME RIGHTS RESERVED

Chapter 20
Semester Change-Over

SOME RIGHTS RESERVED

20.1 What did we learn in AI 1?

SOME RIGHTS RESERVED

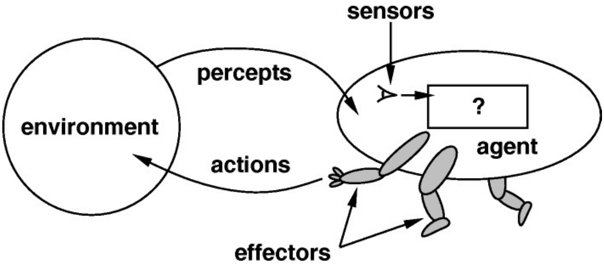
Topics of AI-1 (Winter Semester)

- Getting Started
  - What is artificial intelligence? (situating ourselves)
  - Logic programming in Prolog (An influential paradigm)
  - Intelligent Agents (a unifying framework)
- Problem Solving
  - Problem Solving and search (Black Box World States and Actions)
  - Adversarial search (Game playing) (A nice application of search)
  - constraint satisfaction problems (Factored World States)
- Knowledge and Reasoning
  - Formal Logic as the mathematics of Meaning
  - Propositional logic and satisfiability (Atomic Propositions)
  - First-order logic and theorem proving (Quantification)
  - Logic programming (Logic + Search $\rightsquigarrow$ Programming)
  - Description logics and semantic web
- Planning
  - Planning Frameworks
  - Planning Algorithms
  - Planning and Acting in the real world

SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

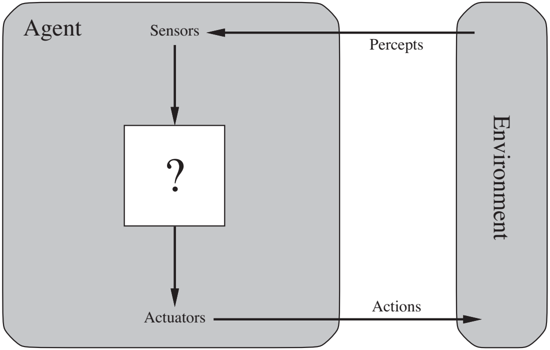
- Agents interact with the environment



SOME RIGHTS RESERVED

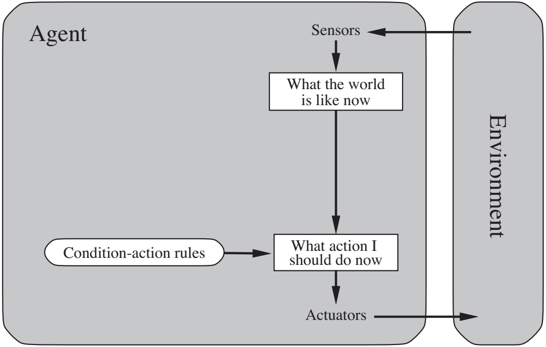
Rational Agents as an Evaluation Framework for AI

- General agent schema



Rational Agents as an Evaluation Framework for AI

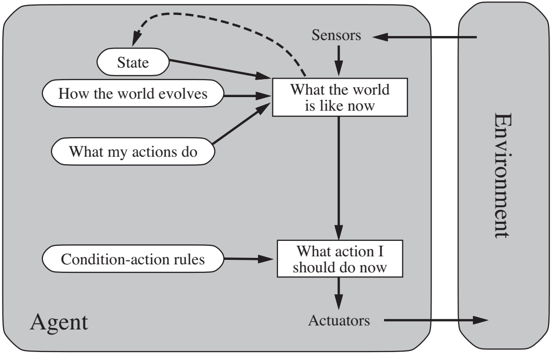
- Reflex Agents



SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

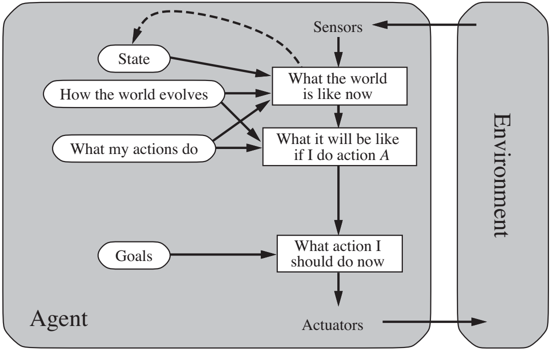
- Reflex Agents with State



SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

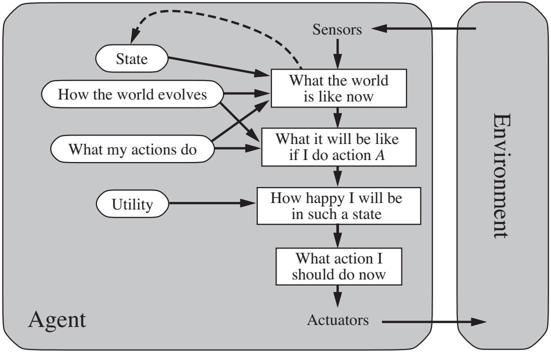
- Goal-Based Agents


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action *A*
Goals
What action I should do now
Agent
Actuators
Environment

SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

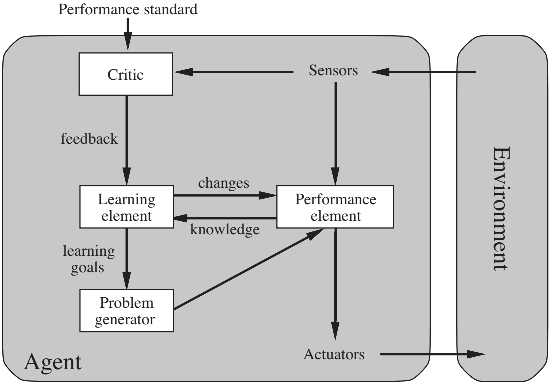
- Utility-Based Agent


Sensors
State
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action *A*
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

- Learning Agents



Rational Agent

- **Idea:** Try to design agents that are successful (do the right thing)
- **Definition 1.1.** An agent is called rational, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date. This is called the MEU principle.
- **Note:** A rational agent need not be perfect
  - only needs to maximize expected value (rational $\neq$ omniscient)
    - need not predict e.g. very unlikely but catastrophic events in the future
  - percepts may not supply all relevant information (Rational $\neq$ clairvoyant)
    - if we cannot perceive things we do not need to react to them.
    - but we may need to try to find out about hidden dangers (exploration)
  - action outcomes may not be as expected (rational $\neq$ successful)
    - but we may need to take action to ensure that they do (more often) (learning)
- Rational $\rightsquigarrow$ exploration, learning, autonomy

SOME RIGHTS RESERVED

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)

SOME RIGHTS RESERVED

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)

SOME RIGHTS RESERVED

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)
- Describing world states by formal language (and drawing inferences)
  - Propositional logic and DPLL (deciding entailment efficiently)
  - First-order logic and ATP (reasoning about infinite domains)
  - **Digression**: Logic programming (logic + search)
  - Description logics as moderately expressive, but decidable logics

SOME RIGHTS RESERVED

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)
- Describing world states by formal language (and drawing inferences)
  - Propositional logic and DPLL (deciding entailment efficiently)
  - First-order logic and ATP (reasoning about infinite domains)
  - **Digression**: Logic programming (logic + search)
  - Description logics as moderately expressive, but decidable logics
- Planning: Problem Solving using white-box world/action descriptions
  - **Framework**: describing world states in logic as sets of propositions and actions by preconditions and add/delete lists
  - **Algorithms**: e.g heuristic search by problem relaxations

SOME RIGHTS RESERVED

Topics of AI-2 (Summer Semester)

- Uncertain Knowledge and Reasoning
  - Uncertainty
  - Probabilistic reasoning
  - Making Decisions in Episodic Environments
  - Problem Solving in Sequential Environments
- Foundations of machine learning
  - Learning from Observations
  - Knowledge in Learning
  - Statistical Learning Methods
- Communication (If there is time)
  - Natural Language Processing
  - Natural Language for Communication

SOME RIGHTS RESERVED

2025-05-14

20.2 Administrative Ground Rules

SOME RIGHTS RESERVED

Prerequisites

- **Remember:** **AI-1** dealt with situations with “complete information” and strictly computable, “perfect” solutions to problems. (i.e. tree search, logical inference, planning, etc.)
- **AI-2** will focus on *probabilistic* scenarios by introducing uncertain situations, and *approximate* solutions to problems. (Bayesian networks, Markov models, machine learning, etc.)

SOME RIGHTS RESERVED

Prerequisites

- **Remember:** **AI-1** dealt with situations with "complete information" and strictly computable, "perfect" solutions to problems. (i.e. tree search, logical inference, planning, etc.)
- **AI-2** will focus on *probabilistic* scenarios by introducing uncertain situations, and *approximate* solutions to problems. (Bayesian networks, Markov models, machine learning, etc.)
- **Weak Prerequisites for AI-2:** (if you do not have them, study up as needed)
  - AI-1 (in particular: PEAS, propositional logic/first-order logic (mostly the syntax), some logic programming)
  - (very) elementary complexity theory. (big Oh and friends)

SOME RIGHTS RESERVED

Prerequisites

- **Remember:** **AI-1** dealt with situations with “complete information” and strictly computable, “perfect” solutions to problems. (i.e. tree search, logical inference, planning, etc.)
- **AI-2** will focus on *probabilistic* scenarios by introducing uncertain situations, and *approximate* solutions to problems. (Bayesian networks, Markov models, machine learning, etc.)
- **Weak Prerequisites for AI-2:** (if you do not have them, study up as needed)
  - AI-1 (in particular: PEAS, propositional logic/first-order logic (mostly the syntax), some logic programming)
  - (very) elementary complexity theory. (big Oh and friends)
  - rudimentary probability theory (e.g. from stochastics)

SOME RIGHTS RESERVED

Prerequisites

- **Remember:** **AI-1** dealt with situations with "complete information" and strictly computable, "perfect" solutions to problems. (i.e. tree search, logical inference, planning, etc.)
- **AI-2** will focus on *probabilistic* scenarios by introducing uncertain situations, and *approximate* solutions to problems. (Bayesian networks, Markov models, machine learning, etc.)
- **Weak Prerequisites for AI-2:** (if you do not have them, study up as needed)
    - AI-1 (in particular: PEAS, propositional logic/first-order logic (mostly the syntax), some logic programming)
    - (very) elementary complexity theory. (big Oh and friends)
    - rudimentary probability theory (e.g. from stochastics)
    - basic linear algebra (vectors, matrices,...)

SOME RIGHTS RESERVED

Prerequisites

- **Remember:** **AI-1** dealt with situations with “complete information” and strictly computable, “perfect” solutions to problems. (i.e. tree search, logical inference, planning, etc.)
- **AI-2** will focus on *probabilistic* scenarios by introducing uncertain situations, and *approximate* solutions to problems. (Bayesian networks, Markov models, machine learning, etc.)
- **Weak Prerequisites for AI-2:** (if you do not have them, study up as needed)
  - AI-1 (in particular: PEAS, propositional logic/first-order logic (mostly the syntax), some logic programming)
  - (very) elementary complexity theory. (big Oh and friends)
  - rudimentary probability theory (e.g. from stochastics)
  - basic linear algebra (vectors, matrices,...)
  - basic real analysis (aka. calculus) (primarily: (partial) derivatives)

SOME RIGHTS RESERVED

Prerequisites

- **Remember:** **AI-1** dealt with situations with "complete information" and strictly computable, "perfect" solutions to problems. (i.e. tree search, logical inference, planning, etc.)
- **AI-2** will focus on *probabilistic* scenarios by introducing uncertain situations, and *approximate* solutions to problems. (Bayesian networks, Markov models, machine learning, etc.)
- **Weak Prerequisites for AI-2:** (if you do not have them, study up as needed)
  - AI-1 (in particular: PEAS, propositional logic/first-order logic (mostly the syntax), some logic programming)
  - (very) elementary complexity theory. (big Oh and friends)
  - rudimentary probability theory (e.g. from stochastics)
  - basic linear algebra (vectors, matrices,...)
  - basic real analysis (aka. calculus) (primarily: (partial) derivatives)
- **Meaning:** I will *assume* you know these things, but some of them we will recap, and what you don't know will make things slightly harder for you, but by no means prohibitively difficult.

SOME RIGHTS RESERVED

“Strict” Prerequisites

- **Most crucially – Mathematical Literacy:** Mathematics is the language that computer scientists express their ideas in! (“*A search problem is a tuple* $(N, S, G, ...)$ *such that...*”)
- **Note:** This is a skill that can be *learned*, and more importantly, *practiced!* Not having/honing this skill *will* make things more difficult for you. Be aware of this and, if necessary, work on it – it will pay off, not only in this course.

SOME RIGHTS RESERVED

"Strict" Prerequisites

- **Most crucially – Mathematical Literacy:** Mathematics is the language that computer scientists express their ideas in! ("*A search problem is a tuple* $(N, S, G, ...)$ *such that...*")
- **Note:** This is a skill that can be *learned*, and more importantly, *practiced!* Not having/honing this skill *will* make things more difficult for you. Be aware of this and, if necessary, work on it – it will pay off, not only in this course.
- **But also:** Motivation, interest, curiosity, hard work. (AI-2 is non-trivial)

SOME RIGHTS RESERVED

“Strict” Prerequisites

- **Most crucially – Mathematical Literacy:** Mathematics is the language that computer scientists express their ideas in! (“*A search problem is a tuple* $(N, S, G, \ldots)$ *such that...*”)
- **Note:** This is a skill that can be *learned*, and more importantly, *practiced!* Not having/honing this skill *will* make things more difficult for you. Be aware of this and, if necessary, work on it – it will pay off, not only in this course.
- **But also:** Motivation, interest, curiosity, hard work. (AI-2 is non-trivial)
- **Note:** Grades correlate significantly with invested effort; including, but not limited to:
  - time spent on exercises, (learning is 80% perspiration, only 20% inspiration)
  - being here in presence, (humans are social animals ⇜ mirror neurons)
  - asking questions, (Q/A dialogues activate brains)

SOME RIGHTS RESERVED

“Strict” Prerequisites

- **Most crucially – Mathematical Literacy:** Mathematics is the language that computer scientists express their ideas in! (“*A search problem is a tuple* $(N, S, G, \ldots)$ *such that...*”)
- **Note:** This is a skill that can be *learned*, and more importantly, *practiced!* Not having/honing this skill *will* make things more difficult for you. Be aware of this and, if necessary, work on it – it will pay off, not only in this course.
- **But also:** Motivation, interest, curiosity, hard work. (AI-2 is non-trivial)
- **Note:** Grades correlate significantly with invested effort; including, but not limited to:
  - time spent on exercises, (learning is 80% perspiration, only 20% inspiration)
  - being here in presence, (humans are social animals ⇝ mirror neurons)
  - asking questions, (Q/A dialogues activate brains)
  - talking to your peers, (pool your insights, share your triumphs/frustrations). . .

  All of these we try to support with the ALeA system. (which also gives us the data to prove this)

SOME RIGHTS RESERVED

Assessment, Grades

- **Overall (Module) Grade:**
  - Grade via the exam (Klausur) $\rightsquigarrow$ 100% of the grade.
  - Up to 10% bonus on-top for an exam with $\geq$ 50% points. (< 50% $\rightsquigarrow$ no bonus)
  - Bonus points $\widehat{=}$ percentage sum of the best 10 prepquizzes divided by 100.

SOME RIGHTS RESERVED

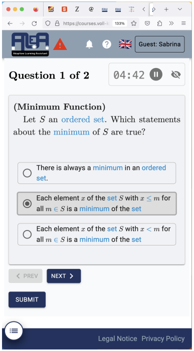
Assessment, Grades

- **Overall (Module) Grade:**
  - Grade via the exam (Klausur) $\leadsto$ 100% of the grade.
  - Up to 10% bonus on-top for an exam with $\geq 50\%$ points. ($< 50\% \leadsto$ no bonus)
  - Bonus points $\widehat{=}$ percentage sum of the best 10 prepquizzes divided by 100.
- **Exam:** exam conducted in presence on paper! ($\sim$ Oct. 10. 2025)
- **Retake Exam:** 90 minutes exam six months later. ($\sim$ April 10. 2026)
- ⚠ You have to register for exams in `https://campo.fau.de` in the first month of classes.
- **Note:** You can de-register from an exam on `https://campo.fau.de` up to three working days before exam. (do not miss that if you are not prepared)

Preparedness Quizzes

- **PrepQuizzes:** Before every lecture we offer a 10 min online quiz – the PrepQuiz – about the material from the previous week. (16:15-16:25; starts in week 2)
- **Motivations:** We do this to
  - keep you prepared and working continuously. (primary)
  - bonus points if the exam has $\geq 50\%$ points (potential part of your grade)
  - update the ALeA learner model. (fringe benefit)
- The prepquizes will be given in the ALeA system

- `https://courses.voll-ki.fau.de/quiz-dash/ai-2`
- You have to be logged into ALeA! (via FAU IDM)
- You can take the prepquiz on your laptop or phone, …
- …in the lecture or at home …
- …via WLAN or 4G Network. (do not overload)
- Prepquizzes will only be available 16:15-16:25!



Question 1 of 2

(Minimum Function)
Let S an ordered set. Which statements about the minimum of S are true?

○ There is always a minimum in an ordered set.

● Each element x of the set S with $x \leq m$ for all $m \in S$ is a minimum of the set

○ Each element x of the set S with $x < m$ for all $m \in S$ is a minimum of the set

SOME RIGHTS RESERVED

⚠ Special Admin Conditions ⚠

- Some degree programs do not “import” the course Artificial Intelligence 1, and thus you may not be able to register for the exam via `https://campo.fau.de`.
  - Just send me an e-mail and come to the exam, (we do the necessary admin)
  - Tell your program coordinator about AI-1/2 so that they remedy this situation
- In “Wirtschafts-Informatik” you can only take AI-1 and AI-2 together in the “Wahlpflichtbereich”.
  - ECTS credits need to be divisible by five ⇝ $7.5 + 7.5 = 15$.

SOME RIGHTS RESERVED

20.3 Overview over AI and Topics of AI-II

SOME RIGHTS RESERVED

20.3.1 What is Artificial Intelligence?

What is Artificial Intelligence? Definition

- **Definition 3.1 (According to Wikipedia).** Artificial Intelligence (AI) is intelligence exhibited by machines
- **Definition 3.2 (also).** Artificial Intelligence (AI) is a sub-field of CS that is concerned with the automation of intelligent behavior.
- **BUT:** it is already difficult to define intelligence precisely.
- **Definition 3.3 (Elaine Rich).** artificial intelligence (AI) studies how we can make the computer do things that humans can still do better at the moment.

SOME RIGHTS RESERVED

What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

the ability to learn

SOME RIGHTS RESERVED

What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

Inference

SOME RIGHTS RESERVED

What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

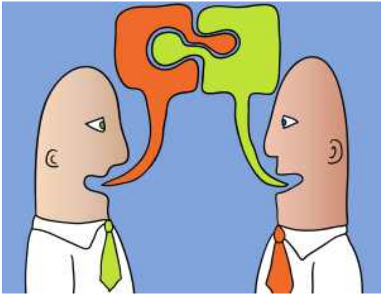
Perception

SOME RIGHTS RESERVED

What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

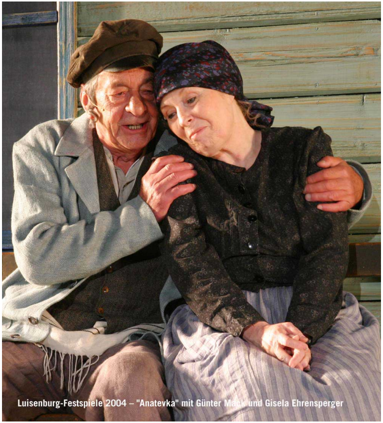
Language understanding



What is Artificial Intelligence? Components

- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

Emotion



Luisenburg-Festspiele 2004 – "Anatevka" mit Günter Mack und Gisela Ehrensperger

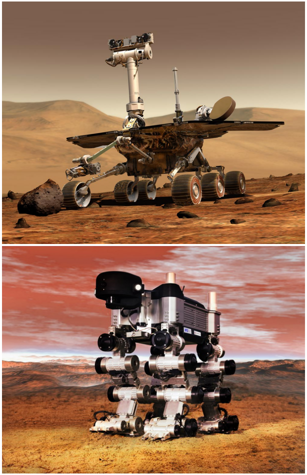
SOME RIGHTS RESERVED

20.3.2 Artificial Intelligence is here today!

SOME RIGHTS RESERVED

Artificial Intelligence is here today!

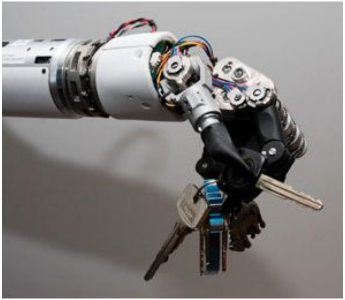
- in outer space
  - in outer space systems need autonomous control:
  - remote control impossible due to time lag
- in artificial limbs
- in household appliances
- in hospitals
- for safety/security



SOME RIGHTS RESERVED

Artificial Intelligence is here today!

- in outer space
- in artificial limbs
  - the user controls the prosthesis via existing nerves, can e.g. grip a sheet of paper.
- in household appliances
- in hospitals
- for safety/security



SOME RIGHTS RESERVED

Artificial Intelligence is here today!

- in outer space
- in artificial limbs
- in household appliances
    - The iRobot Roomba vacuums, mops, and sweeps in corners, . . . , parks, charges, and discharges.
    - general robotic household help is on the horizon.
- in hospitals
- for safety/security

SOME RIGHTS RESERVED

Artificial Intelligence is here today!

- in outer space
- in artificial limbs
- in household appliances
- in hospitals
  - in the USA 90% of the prostate operations are carried out by RoboDoc
  - Paro is a cuddly robot that eases solitude in nursing homes.
- for safety/security

SOME RIGHTS RESERVED

Artificial Intelligence is here today!

- in outer space
- in artificial limbs
- in household appliances
- in hospitals
- for safety/security
    - e.g. Intel verifies correctness of all chips after the “Pentium 5 disaster”

© 1999 Randy Glasbergen. www.glasbergen.com

"It's the latest innovation in office safety. When your computer crashes, an air bag is activated so you won't bang your head in frustration."

SOME RIGHTS RESERVED

The AI Conundrum

- **Observation:** Reserving the term “artificial intelligence” has been quite a land grab!
- **But:** researchers at the Dartmouth Conference (1956) really thought they would solve/reach AI in two/three decades.
- **Consequence:** AI still asks the big questions. (and still promises answers soon)
- **Another Consequence:** AI as a field is an incubator for many innovative technologies.
- **AI Conundrum:** Once AI solves a subfield it is called “CS”. (becomes a separate subfield of CS)
- **Example 3.4.** Functional/Logic Programming, automated theorem proving, Planning, machine learning, Knowledge Representation, ...
- **Still Consequence:** AI research was alternatingly flooded with money and cut off brutally.

SOME RIGHTS RESERVED

The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.

SOME RIGHTS RESERVED

The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.
- Funding levels are tied to public perception of success (especially for AI)

SOME RIGHTS RESERVED

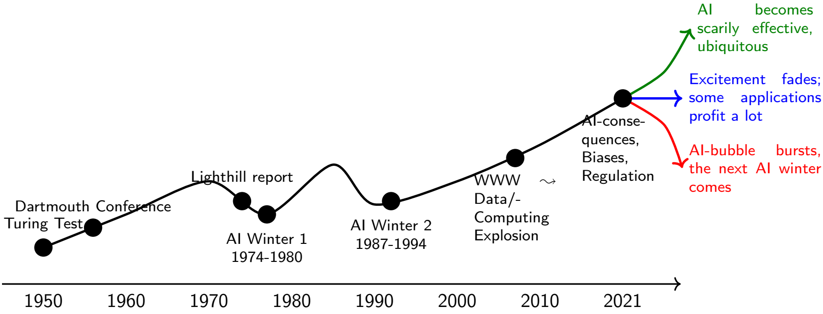
The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.
- Funding levels are tied to public perception of success (especially for AI)
- **Definition 3.7.** An AI winter is a time period of low public perception and funding for AI,
  mostly because AI has failed to deliver on its – sometimes overblown – promises
  An AI summer is a time period of high public perception and funding for AI

SOME RIGHTS RESERVED

The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.
- Funding levels are tied to public perception of success (especially for AI)
- **Definition 3.8.** An AI winter is a time period of low public perception and funding for AI,
  mostly because AI has failed to deliver on its – sometimes overblown – promises
  An AI summer is a time period of high public perception and funding for AI
- A potted history of AI (AI summers and summers)


Dartmouth Conference
Turing Test
Lighthill report
AI Winter 1
1974-1980
AI Winter 2
1987-1994
WWW ⇝ Data/-Computing Explosion
AI-consequences, Biases, Regulation
AI becomes scarily effective, ubiquitous
Excitement fades; some applications profit a lot
AI-bubble bursts, the next AI winter comes
1950
1960
1970
1980
1990
2000
2010
2021

SOME RIGHTS RESERVED

20.3.3 Ways to Attack the AI Problem

SOME RIGHTS RESERVED

Four Main Approaches to Artificial Intelligence

- **Definition 3.9.** Symbolic AI is a subfield of AI based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into meaning-carrying structures (expressions) and manipulating them (using processes) to produce new expressions.

SOME RIGHTS RESERVED

Four Main Approaches to Artificial Intelligence

- **Definition 3.13.** Symbolic AI is a subfield of AI based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into meaning-carrying structures (expressions) and manipulating them (using processes) to produce new expressions.
- **Definition 3.14.** Statistical AI remedies the two shortcomings of symbolic AI approaches: that all concepts represented by symbols are crisply defined, and that all aspects of the world are knowable/representable in principle. Statistical AI adopts sophisticated mathematical models of uncertainty and uses them to create more accurate world models and reason about them.

SOME RIGHTS RESERVED

Four Main Approaches to Artificial Intelligence

- **Definition 3.17.** Symbolic AI is a subfield of AI based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into meaning-carrying structures (expressions) and manipulating them (using processes) to produce new expressions.
- **Definition 3.18.** Statistical AI remedies the two shortcomings of symbolic AI approaches: that all concepts represented by symbols are crisply defined, and that all aspects of the world are knowable/representable in principle. Statistical AI adopts sophisticated mathematical models of uncertainty and uses them to create more accurate world models and reason about them.
- **Definition 3.19.** Subsymbolic AI (also called connectionism or neural AI) is a subfield of AI that posits that intelligence is inherently tied to brains, where information is represented by a simple sequence pulses that are processed in parallel via simple calculations realized by neurons, and thus concentrates on neural computing.

SOME RIGHTS RESERVED

Four Main Approaches to Artificial Intelligence

- **Definition 3.21.** Symbolic AI is a subfield of AI based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into meaning-carrying structures (expressions) and manipulating them (using processes) to produce new expressions.
- **Definition 3.22.** Statistical AI remedies the two shortcomings of symbolic AI approaches: that all concepts represented by symbols are crisply defined, and that all aspects of the world are knowable/representable in principle. Statistical AI adopts sophisticated mathematical models of uncertainty and uses them to create more accurate world models and reason about them.
- **Definition 3.23.** Subsymbolic AI (also called connectionism or neural AI) is a subfield of AI that posits that intelligence is inherently tied to brains, where information is represented by a simple sequence pulses that are processed in parallel via simple calculations realized by neurons, and thus concentrates on neural computing.
- **Definition 3.24.** Embodied AI posits that intelligence cannot be achieved by reasoning about the state of the world (symbolically, statistically, or connectivist), but must be embodied i.e. situated in the world, equipped with a "body" that can interact with it via sensors and actuators. Here, the main method for realizing intelligent behavior is by learning from the world.

SOME RIGHTS RESERVED

Two ways of reaching Artificial Intelligence?

- We can classify the AI approaches by their coverage and the analysis depth(they are complementary)

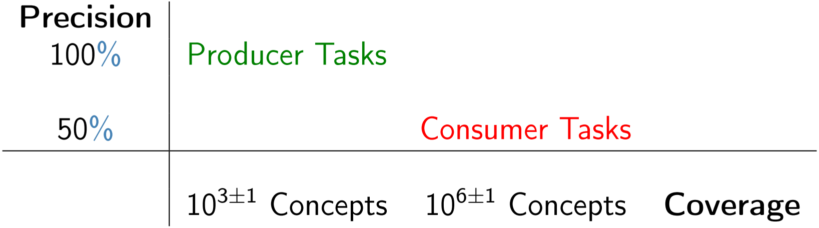
Deep	symbolic AI-1	not there yet cooperation?
Shallow	no-one wants this	statistical/sub symbolic AI-2
Analysis ↑ vs. Coverage →	Narrow	Wide

- **This semester** we will cover foundational aspects of symbolic AI (deep/narrow processing)
- **next semester** concentrate on statistical/subsymbolic AI. (shallow/wide-coverage)

SOME RIGHTS RESERVED

Environmental Niches for both Approaches to AI

- **Observation:** There are two kinds of applications/tasks in AI
  - Consumer tasks: consumer grade applications have tasks that must be fully generic and wide coverage. (e.g. machine translation like Google Translate)
  - Producer tasks: producer grade applications must be high-precision, but can be domain-specific (e.g. multilingual documentation, machinery-control, program verification, medical technology)



after Aarne Ranta [**Ranta:atcp17**].

- **General Rule:** Subsymbolic AI is well suited for consumer tasks, while symbolic AI is better suited for producer tasks.
- A domain of producer tasks I am interested in: mathematical/technical documents.

SOME RIGHTS RESERVED

20.3.4 AI in the KWARC Group

The KWARC Research Group

- **Observation:** The ability to represent knowledge about the world and to draw logical inferences is one of the central components of intelligent behavior.
- **Thus:** reasoning components of some form are at the heart of many AI systems.
- **KWARC Angle:** Scaling up (web-coverage) without dumbing down (too much)
  - Content markup instead of full formalization (too tedious)
  - User support and quality control instead of “The Truth” (elusive anyway)
  - use Mathematics as a test tube (⚠ Mathematics $\hat{=}$ Anything Formal ⚠)
  - care more about applications than about philosophy (we cannot help getting this right anyway as logicians)
- The KWARC group was established at Jacobs Univ. in 2004, moved to FAU Erlangen in 2016
- See `http://kwarc.info` for projects, publications, and links

SOME RIGHTS RESERVED

Overview: KWARC Research and Projects

Applications: eMath 3.0, Active Documents, Active Learning, Semantic Spreadsheets/CAD/CAM, Change Mangagement, Global Digital Math Library, Math Search Systems, SMGloM: Semantic Multilingual Math Glossary, Serious Games, . . .

Foundations of Math:

- MathML, *OpenMath*
- advanced Type Theories
- MMT: Meta Meta Theory
- Logic Morphisms/Atlas
- Theorem Prover/CAS Interoperability
- Mathematical Models/Simulation

KM & Interaction:

- Semantic Interpretation (aka. Framing)
- math-literate interaction
- MathHub: math archives & active docs
- Active documents: embedded semantic services
- Model-based Education

Semantization:

- LaTeXML: LaTeX ⇝ XML
- sTeX: Semantic LaTeX
- invasive editors
- Context-Aware IDEs
- Mathematical Corpora
- Linguistics of Math
- ML for Math Semantics Extraction

Foundations: Computational Logic, Web Technologies, OMDoc/MMT

SOME RIGHTS RESERVED

Research Topics in the KWARC Group

- We are always looking for bright, motivated KWARCies.
- We have topics in for all levels! (Enthusiast, Bachelor, Master, Ph.D.)
- List of current topics: `https://gl.kwarc.info/kwarc/thesis-projects/`
  - Automated Reasoning: Maths Representation in the Large
  - Logics development, $(\text{Meta})^n$-Frameworks
  - Math Corpus Linguistics: Semantics Extraction
  - Serious Games, Cognitive Engineering, Math Information Retrieval, Legal Reasoning, . . .
  - . . . last but not least: KWARC is the home of ALeA!
- We always try to find a topic at the intersection of your and our interests.
- We also sometimes have positions!. (HiWi, Ph.D.: $\frac{1}{2}$ E-13, PostDoc: full E-13)

SOME RIGHTS RESERVED

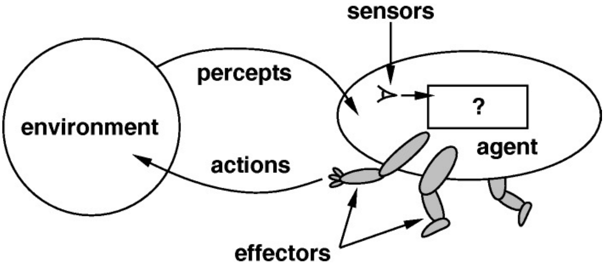
20.3.5 Agents and Environments in AI2

20.3.5.1 Recap: Rational Agents as a Conceptual Framework

Agents and Environments

- **Definition 3.25.** An agent is anything that
  - perceives its environment via sensors (a means of sensing the environment)
  - acts on it with actuators (means of changing the environment).

  Any recognizable, coherent employment of the actuators of an agent is called an action.

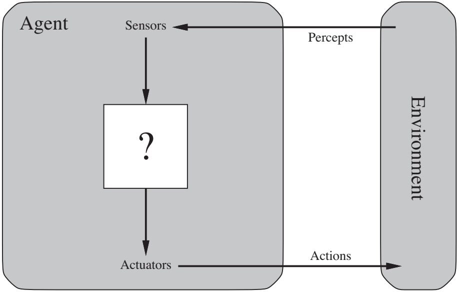


- **Example 3.26.** Agents include humans, robots, softbots, thermostats, etc.
- **Remark:** The notion of an agent and its environment is intentionally designed to be inclusive. We will classify and discuss subclasses of both later.

SOME RIGHTS RESERVED

Agent Schema: Visualizing the Internal Agent Structure

- **Agent Schema:** We will use the following kind of agent schema to visualize the internal structure of an agent:



Different agents differ on the contents of the white box in the center.

Rationality

- **Idea:** Try to design agents that are successful! (aka. "do the right thing")
- **Problem:** What do we mean by "successful", how do we measure "success"?
- **Definition 3.27.** A performance measure is a function that evaluates a sequence of environments.
- **Example 3.28.** A performance measure for a vacuum cleaner could
  - award one point per "square" cleaned up in time T?
  - award one point per clean "square" per time step, minus one per move?
  - penalize for $> k$ dirty squares?
- **Definition 3.29.** An agent is called rational, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date.
- **Critical Observation:** We only need to maximize the expected value, not the actual value of the performance measure!
- **Question:** Why is rationality a good quality to aim for?

SOME RIGHTS RESERVED

Consequences of Rationality: Exploration, Learning, Autonomy

- **Note:** A rational agent need not be perfect:
  - It only needs to maximize expected value (rational $\neq$ omniscient)
    - need not predict e.g. very unlikely but catastrophic events in the future
  - Percepts may not supply all relevant information (rational $\neq$ clairvoyant)
    - if we cannot perceive things we do not need to react to them.
    - but we may need to try to find out about hidden dangers (exploration)
  - Action outcomes may not be as expected (rational $\neq$ successful)
    - but we may need to take action to ensure that they do (more often) (learning)
- **Note:** Rationality may entail exploration, learning, autonomy (depending on the environment / task)
- **Definition 3.30.** An agent is called autonomous, if it does not rely on the prior knowledge about the environment of the designer.
- Autonomy avoids fixed behaviors that can become unsuccessful in a changing environment. (anything else would be irrational)
- The agent may have to learn all relevant traits, invariants, properties of the environment and actions.

SOME RIGHTS RESERVED

PEAS: Describing the Task Environment

- **Observation:** To design a rational agent, we must specify the task environment in terms of performance measure, environment, actuators, and sensors, together called the PEAS components.
- **Example 3.31.** When designing an automated taxi:
  - **Performance measure:** safety, destination, profits, legality, comfort, . . .
  - **Environment:** US streets/freeways, traffic, pedestrians, weather, . . .
  - **Actuators:** steering, accelerator, brake, horn, speaker/display, . . .
  - **Sensors:** video, accelerometers, gauges, engine sensors, keyboard, GPS, . . .
- **Example 3.32 (Internet Shopping Agent).** The task environment:
  - Performance measure: price, quality, appropriateness, efficiency
  - Environment: current and future WWW sites, vendors, shippers
  - Actuators: display to user, follow URL, fill in form
  - Sensors: HTML pages (text, graphics, scripts)

SOME RIGHTS RESERVED

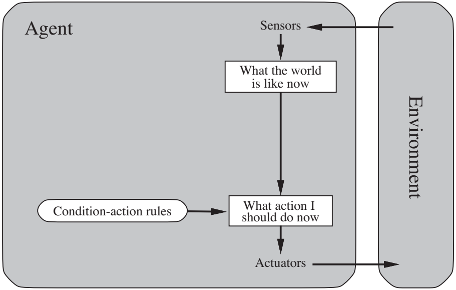
Environment types

- **Observation 3.33.** *Agent design is largely determined by the type of environment it is intended for.*
- **Problem:** There is a vast number of possible kinds of environments in AI.
- **Solution:** Classify along a few "dimensions". (independent characteristics)
- **Definition 3.34.** For an agent *a* we classify the environment *e* of *a* by its type, which is one of the following. We call *e*
  1. fully observable, iff the *a*'s sensors give it access to the complete state of the environment at any point in time, else partially observable.
  2. deterministic, iff the next state of the environment is completely determined by the current state and *a*'s action, else stochastic.
  3. episodic, iff *a*'s experience is divided into atomic episodes, where it perceives and then performs a single action. Crucially, the next episode does not depend on previous ones. Non-episodic environments are called sequential.
  4. dynamic, iff the environment can change without an action performed by *a*, else static. If the environment does not change but *a*'s performance measure does, we call *e* semidynamic.
  5. discrete, iff the sets of *e*'s state and *a*'s actions are countable, else continuous.
  6. single-agent, iff only *a* acts on *e*; else multi-agent(when must we count parts of *e* as agents?)

SOME RIGHTS RESERVED

Reflex Agents

- **Definition 3.35.** An agent $\langle\mathcal{P},\mathcal{A},f\rangle$ is called a reflex agent, iff it only takes the last percept into account when choosing an action, i .e. $f(p_1,\ldots,p_k)=f(p_k)$ for all $p_1,\ldots,p_k\in\mathcal{P}$.
- **Agent Schema:**

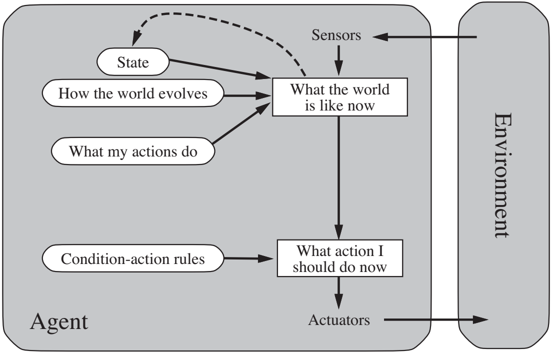


- **Example 3.36 (Agent Program).**

```
procedure Reflex−Vacuum−Agent [location,status] returns an action
  if status = Dirty then . . .
```

Model-based Reflex Agents: Idea

- **Idea:** Keep track of the state of the world we cannot see in an internal model.
- **Agent Schema:**



SOME RIGHTS RESERVED

Model-based Reflex Agents: Definition

- **Definition 3.37.** A model-based agent $\langle\mathcal{P},\mathcal{A},\mathcal{S},\mathcal{T},s_0,S,a\rangle$ is an agent $\langle\mathcal{P},\mathcal{A},f\rangle$ whose actions depend on
  1. a world model: a set $\mathcal{S}$ of possible states, and a start state $s_0\in\mathcal{S}$.
  2. a transition model $\mathcal{T}$, that predicts a new state $\mathcal{T}(s,a)$ from a state s and an action a.
  3. a sensor model S that given a state s and a percept p determine a new state $S(s,p)$.
  4. an action function $a\colon\mathcal{S}\to\mathcal{A}$ that given a state selects the next action.

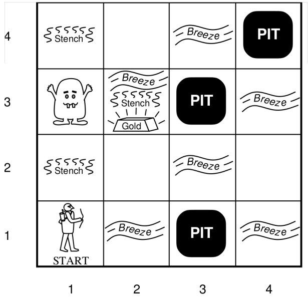
  If the world model of a model-based agent A is in state s and A has last taken action a, and now perceives p, then A will transition to state $s'=S(p,\mathcal{T}(s,a))$ and take action $a'=a(s')$.
  So, given a sequence $p_1,\ldots,p_n$ of percepts, we recursively define states $s_n=S(\mathcal{T}(s_{n-1},a(s_{n-1})),p_n)$ with $s_1=S(s_0,p_1)$. Then $f(p_1,\ldots,p_n)=a(s_n)$.
- **Note:** As different percept sequences lead to different states, so the agent function $f()\colon\mathcal{P}^*\to\mathcal{A}$ no longer depends only on the last percept.
- **Example 3.38 (Tail Lights Again).** Model-based agents can do the ??? if the states include a concept of tail light brightness.

20.3.5.2 Sources of Uncertainty

SOME RIGHTS RESERVED

Sources of Uncertainty in Decision-Making

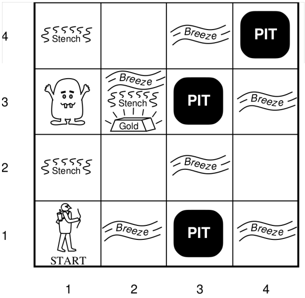
Where's that d. . . Wumpus?
And where am I, anyway??



- **Non-deterministic actions:**
  - "When I try to go forward in this dark cave, I might actually go forward-left or forward-right."

Sources of Uncertainty in Decision-Making

Where's that d. . . Wumpus?
And where am I, anyway??

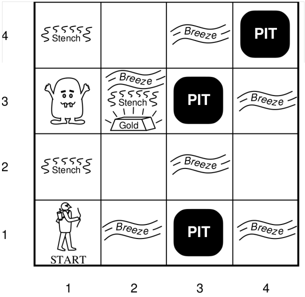


- **Non-deterministic actions:**
  - "When I try to go forward in this dark cave, I might actually go forward-left or forward-right."
- **Partial observability with unreliable sensors:**
  - "Did I feel a breeze right now?";
  - "I think I might smell a Wumpus here, but I got a cold and my nose is blocked."
  - "According to the heat scanner, the Wumpus is probably in cell [2,3]."

SOME RIGHTS RESERVED

Sources of Uncertainty in Decision-Making

Where's that d. . . Wumpus?
And where am I, anyway??



- **Non-deterministic actions:**
  - "When I try to go forward in this dark cave, I might actually go forward-left or forward-right."
- **Partial observability with unreliable sensors:**
  - "Did I feel a breeze right now?";
  - "I think I might smell a Wumpus here, but I got a cold and my nose is blocked."
  - "According to the heat scanner, the Wumpus is probably in cell [2,3]."
- **Uncertainty about the domain behavior:**
  - "Are you *sure* the Wumpus never moves?"

SOME RIGHTS RESERVED

Unreliable Sensors

- **Robot Localization:** Suppose we want to support localization using landmarks to narrow down the area.
- **Example 3.39.** "*If you see the Eiffel tower, then you're in Paris.*"

Unreliable Sensors

- **Robot Localization:** Suppose we want to support localization using landmarks to narrow down the area.
- **Example 3.40.** "*If you see the Eiffel tower, then you're in Paris.*"
- **Difficulty:** Sensors can be imprecise.
    - Even if a landmark is perceived, we cannot conclude with certainty that the robot is at that location.
    - "*This is the half-scale Las Vegas copy, you dummy.*"
    - Even if a landmark is *not* perceived, we cannot conclude with certainty that the robot is *not* at that location.
    - "*Top of Eiffel tower hidden in the clouds.*"
- Only the probability of being at a location increases or decreases.

SOME RIGHTS RESERVED

20.3.5.3 Agent Architectures based on Belief States

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 3.42.** A model-based agent has a world model consisting of
  - a belief state that has information about the possible states the world may be in,
  - a sensor model that updates the belief state based on sensor information, and
  - a transition model that updates the belief state based on actions.

SOME RIGHTS RESERVED

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 3.43.** A model-based agent has a world model consisting of
  - a belief state that has information about the possible states the world may be in,
  - a sensor model that updates the belief state based on sensor information, and
  - a transition model that updates the belief state based on actions.
- **Idea:** The agent environment determines what the world model can be.

SOME RIGHTS RESERVED

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 3.44.** A model-based agent has a world model consisting of
  - a belief state that has information about the possible states the world may be in,
  - a sensor model that updates the belief state based on sensor information, and
  - a transition model that updates the belief state based on actions.
- **Idea:** The agent environment determines what the world model can be.
- In a fully observable, deterministic environment,
  - we can observe the initial state and subsequent states are given by the actions alone.
  - Thus the belief state is a singleton (we call its sole member the world state) and the transition model is a function from states and actions to states: a transition function.

SOME RIGHTS RESERVED

World Models by Agent Type in AI-1

- **Search-based Agents:** In a fully observable, deterministic environment
  - goal-based agent with world state $\widehat{=}$ "current state"
  - no inference. (goal $\widehat{=}$ goal state from search problem)
- **CSP-based Agents:** In a fully observable, deterministic environment
  - goal-based agent withworld state $\widehat{=}$ constraint network,
  - inference $\widehat{=}$ constraint propagation. (goal $\widehat{=}$ satisfying assignment)
- **Logic-based Agents:** In a fully observable, deterministic environment
  - model-based agent with world state $\widehat{=}$ logical formula
  - inference $\widehat{=}$ e.g. DPLL or resolution.
- **Planning Agents:** In a fully observable, deterministic, environment
  - goal-based agent with world state $\widehat{=}$ PL0, transition model $\widehat{=}$ STRIPS,
  - inference $\widehat{=}$ state/plan space search. (goal: complete plan/execution)

SOME RIGHTS RESERVED

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\rightsquigarrow$ generalize the transition function to a transition relation.

SOME RIGHTS RESERVED

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\leadsto$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)

SOME RIGHTS RESERVED

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\leadsto$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)
- In a deterministic, but partially observable environment,
  - the belief state must deal with a set of possible states.
  - we can use transition functions.
  - We need a sensor model, which predicts the influence of percepts on the belief state – during update.

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\leadsto$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)
- In a deterministic, but partially observable environment,
  - the belief state must deal with a set of possible states.
  - we can use transition functions.
  - We need a sensor model, which predicts the influence of percepts on the belief state – during update.
- In a stochastic, partially observable environment,
  - mix the ideas from the last two. (sensor model + transition relation)

SOME RIGHTS RESERVED

Preview: New World Models (Belief) $\rightsquigarrow$ new Agent Types

- **Probabilistic Agents:** In a partially observable environment
  - belief state $\widehat{=}$ Bayesian networks,
  - inference $\widehat{=}$ probabilistic inference.

Preview: New World Models (Belief) $\leadsto$ new Agent Types

- **Probabilistic Agents:** In a partially observable environment
  - belief state $\widehat{=}$ Bayesian networks,
  - inference $\widehat{=}$ probabilistic inference.
- **Decision-Theoretic Agents:** In a partially observable, stochastic environment
  - belief state + transition model $\widehat{=}$ decision networks,
  - inference $\widehat{=}$ maximizing expected utility.
- We will study them in detail this semester.

SOME RIGHTS RESERVED

Overview: AI2

- Basics of probability theory (probability spaces, random variables, conditional probabilities, independence,...)

Overview: AI2

- Basics of probability theory (probability spaces, random variables, conditional probabilities, independence,...)
- Probabilistic reasoning: Computing the *a posteriori* probabilities of events given evidence, causal reasoning (Representing distributions efficiently, Bayesian networks,...)

SOME RIGHTS RESERVED

Overview: AI2

- Basics of probability theory (probability spaces, random variables, conditional probabilities, independence,...)
- Probabilistic reasoning: Computing the *a posteriori* probabilities of events given evidence, causal reasoning (Representing distributions efficiently, Bayesian networks,...)
- Probabilistic Reasoning over time (Markov chains, Hidden Markov models,...)

⇒ We can update our world model episodically based on observations (i.e. sensor data)

SOME RIGHTS RESERVED

Overview: AI2

- Basics of probability theory (probability spaces, random variables, conditional probabilities, independence,...)
- Probabilistic reasoning: Computing the *a posteriori* probabilities of events given evidence, causal reasoning (Representing distributions efficiently, Bayesian networks,...)
- Probabilistic Reasoning over time (Markov chains, Hidden Markov models,...)

⇒ We can update our world model episodically based on observations (i.e. sensor data)

- Decision theory: Making decisions under uncertainty (Preferences, Utilities, Decision networks, Markov Decision Procedures,...)

⇒ We can choose the right action based on our world model and the likely outcomes of our actions

SOME RIGHTS RESERVED

Overview: AI2

- Basics of probability theory (probability spaces, random variables, conditional probabilities, independence,...)
- Probabilistic reasoning: Computing the *a posteriori* probabilities of events given evidence, causal reasoning (Representing distributions efficiently, Bayesian networks,...)
- Probabilistic Reasoning over time (Markov chains, Hidden Markov models,...)

⇒ We can update our world model episodically based on observations (i.e. sensor data)

- Decision theory: Making decisions under uncertainty (Preferences, Utilities, Decision networks, Markov Decision Procedures,...)

⇒ We can choose the right action based on our world model and the likely outcomes of our actions

- Machine learning: Learning from data (Decision Trees, Classifiers, Neural Networks,...)

SOME RIGHTS RESERVED

Part 5
Reasoning with Uncertain Knowledge

SOME RIGHTS RESERVED

Chapter 21
Quantifying Uncertainty

21.1 Probability Theory

SOME RIGHTS RESERVED

21.1.1 Prior and Posterior Probabilities

SOME RIGHTS RESERVED

Probabilistic Models

- **Definition 1.1 (Mathematically (slightly simplified)).** A probability space or (probability model) is a pair $\langle\Omega, P\rangle$ such that:
  - Ω is a set of outcomes (called the sample space),
  - P is a function $\mathcal{P}(\Omega) \to [0{,}1]$, such that:
    - $P(\Omega) = 1$ and
    - $P(\bigcup_i A_i) = \sum_i P(A_i)$ for all pairwise disjoint $A_i \in \mathcal{P}(\Omega)$.

  P is called a probability measure.

  These properties are called the Kolmogorov axioms.
- **Intuition:** We run some experiment, the outcome of which is any $\omega \in \Omega$.
  - For $X \subseteq \Omega$, $P(X)$ is the probability that the result of the experiment is *any one* of the outcomes in X.
  - Naturally, the probability that *any* outcome occurs is 1 (hence $P(\Omega) = 1$).
  - The probability of pairwise disjoint sets of outcomes should just be the sum of their probabilities.
- **Example 1.2 (Dice throws).** Assume we throw a (fair) die two times. Then the sample space Ω is $\{(i,j) \mid 1 \leq i,j \leq 6\}$. We define P by letting $P(\{A\}) = \frac{1}{36}$ for every $A \in \Omega$.
  Since the probability of any outcome is the same, we say P is uniformly distributed.

SOME RIGHTS RESERVED

Random Variables

- In practice, we are rarely interested in the *specific* outcome of an experiment, but rather in some *property* of the outcome. This is especially true in the very common situation where we don't even *know* the precise probabilities of the individual outcomes.
- **Example 1.3.** The probability that the *sum* of our two dice throws is 7 is $P(\{(i,j)\in\Omega \mid i+j=7\}) = P(\{(6,1),(1,6),(5,2),(2,5),(4,3),(3,4)\}) = \frac{6}{36} = \frac{1}{6}$.
- **Definition 1.4 (Again, slightly simplified).** Let D be a set. A random variable is a function $X\colon\Omega\to D$. We call D (somewhat confusingly) the domain of X, denoted $\mathrm{dom}(X)$.
  For $x\in D$, we define the probability of x as
  $P(X=x) := P(\{\omega\in\Omega \mid X(\omega)=x\})$.
- **Definition 1.5.** We say that a random variable X is finite domain, iff its domain $\mathrm{dom}(X)$ is finite and Boolean, iff $\mathrm{dom}(X)=\{\mathsf{T},\mathsf{F}\}$.
  For a Boolean random variable, we will simply write $P(X)$ for $P(X=\mathsf{T})$ and $P(\neg X)$ for $P(X=\mathsf{F})$.

SOME RIGHTS RESERVED

Some Examples

- **Example 1.6.** Summing up our two dice throws is a random variable $S\colon \Omega \to [2,12]$ with $S((i,j)) = i + j$. The probability that they sum up to 7 is written as $P(S = 7) = \frac{1}{6}$.
- **Example 1.7.** The first and second of our two dice throws are random variables $\mathrm{First}, \mathrm{Second}\colon \Omega \to [1,6]$ with $\mathrm{First}((i,j)) = i$ and $\mathrm{Second}((i,j)) = j$.
- *Remark 1.8.* Note, that the *identity* $\Omega \to \Omega$ is a random variable as well.
- **Example 1.9.** We can model toothache, cavity and gingivitis as Boolean random variables, with the underlying probability space being...?? ¯_(ツ)_/¯
- **Example 1.10.** We can model tomorrow's weather as a random variable with domain $\{\texttt{sunny}, \texttt{rainy}, \texttt{foggy}, \texttt{warm}, \texttt{cloudy}, \texttt{humid}, \ldots\}$, with the underlying probability space being...?? ¯_(ツ)_/¯

⇒ This is why *probabilistic reasoning* is necessary: We can rarely reduce probabilistic scenarios down to clearly defined, fully known probability spaces and derive all the interesting things from there.

But: The definitions here allow us to *reason* about probabilities and random variables in a *mathematically* rigorous way, e.g. to make our intuitions and assumptions precise, and prove our methods to be *sound*.

SOME RIGHTS RESERVED

Propositions

- This is nice and all, but in practice we are interested in "compound" probabilities like:

  "What is the probability that the sum of our two dice throws is 7, but neither of the two dice is a 3?"

- **Idea:** Reuse the syntax of propositional logic and define the logical connectives for random variables!

- **Example 1.11.** We can express the above as: $P(\neg(\mathrm{First}=3)\wedge\neg(\mathrm{Second}=3)\wedge(S=7))$

- **Definition 1.12.** Let X_1, X_2 be random variables, $x_1\in\mathrm{dom}(X_1)$ and $x_2\in\mathrm{dom}(X_2)$. We define:

  1. $P(X_1\neq x_1){:=}P(\neg(X_1=x_1)) := P(\{\omega\in\Omega\,|\,X_1(\omega)\neq x_1\}){=}1-P(X_1=x_1)$.
  2. $P((X_1=x_1)\wedge(X_2=x_2)) := P(\{\omega\in\Omega\,|\,(X_1(\omega)=x_1)\wedge(X_2(\omega)=x_2)\})$ $=P(\{\omega\in\Omega\,|\,X_1(\omega)=x_1\}\cap\{\omega\in\Omega\,|\,X_2(\omega)=x_2\})$.
  3. $P((X_1=x_1)\vee(X_2=x_2)) := P(\{\omega\in\Omega\,|\,(X_1(\omega)=x_1)\vee(X_2(\omega)=x_2)\})$ $=P(\{\omega\in\Omega\,|\,X_1(\omega)=x_1\}\cup\{\omega\in\Omega\,|\,X_2(\omega)=x_2\})$.

  It is also common to write $P(A,B)$ for $P(A\wedge B)$

- **Example 1.13.** $P((\mathrm{First}\neq 3)\wedge(\mathrm{Second}\neq 3)\wedge(S=7)) = P(\{(1,6),(6,1),(2,5),(5,2)\}) = \frac{1}{9}$

SOME RIGHTS RESERVED

Events

- **Definition 1.14 (Again slightly simplified).** Let $\langle\Omega, P\rangle$ be a probability space. An event is a subset of Ω.
- **Definition 1.15 (Convention).** We call an event (by extension) anything that *represents* a subset of Ω: any statement formed from the logical connectives and values of random variables, on which $P(\cdot)$ is defined.
- **Problem 1.1**
  Remember: We can define $A \vee B := \neg(\neg A \wedge \neg B)$, $\mathsf{T} := A \vee \neg A$ and $\mathsf{F} := \neg\mathsf{T}$ – is this compatible with the definition of probabilities on propositional formulae? And why is $P(X_1 \neq x_1) = 1 - P(X_1 = x_1)$?
- **Problem 1.2 (Inclusion-Exclusion-Principle)**
  Show that $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$.
- **Problem 1.3**
  Show that $P(A) = P(A \wedge B) + P(A \wedge \neg B)$

SOME RIGHTS RESERVED

Conditional Probabilities

- **Observation:** As we gather new information, our beliefs (*should*) change, and thus our probabilities!
- **Example 1.16.** Your "probability of missing the connection train" increases when you are informed that your current train has 30 minutes delay.
- **Example 1.17.** The "probability of cavity" increases when the doctor is informed that the patient has a toothache.
- **Example 1.18.** The probability that $S=3$ is clearly higher if I know that $\mathrm{First}=1$ than otherwise – or if I know that $\mathrm{First}=6$!
- **Definition 1.19.** Let A and B be events where $P(B)\neq 0$. The conditional probability of A given B is defined as:

  $$P(A \mid B) := \frac{P(A \wedge B)}{P(B)}$$

  We also call $P(A)$ the prior probability of A, and $P(A \mid B)$ the posterior probability.
- **Intuition:** If we *assume* B to hold, then we are only interested in the "part" of Ω where A is true *relative to* B.
- **Alternatively:** We restrict our sample space Ω to the subset of outcomes where B holds. We then define a new probability space on this subset by scaling the probability measure so that it sums to 1 – which we do by dividing by

SOME RIGHTS RESERVED

Examples

- **Example 1.20.** If we assume $\mathrm{First} = 1$, then $P(S = 3 \mid (\mathrm{First} = 1))$ should be precisely $P(\mathrm{Second} = 2) = \frac{1}{6}$. We check:

$$P(S = 3 \mid (\mathrm{First} = 1)) = \frac{P((S = 3) \wedge (\mathrm{First} = 1))}{P(\mathrm{First} = 1)} = \frac{1/36}{1/6} = \frac{1}{6}$$

- **Example 1.21.** Assume the prior probability $P(\mathrm{cavity})$ is 0.122. The probability that a patient has both a cavity and a toothache is $P(\mathrm{cavity} \wedge \mathrm{toothache}) = 0.067$. The probability that a patient has a toothache is $P(\mathrm{toothache}) = 0.15$.

  If the patient complains about a toothache, we can update our estimation by computing the posterior probability:

$$P(\mathrm{cavity} \mid \mathrm{toothache}) = \frac{P(\mathrm{cavity} \wedge \mathrm{toothache})}{P(\mathrm{toothache})} = \frac{0.067}{0.15} = 0.45.$$

- **Note:** We just computed the probability of some underlying *disease* based on the presence of a *symptom*!
- **More Generally:** We computed the probability of a *cause* from observing its *effect*.

SOME RIGHTS RESERVED

Some Rules

- Equations on unconditional probabilities have direct analogues for conditional probabilities.
- **Problem 1.4**
  Convince yourself of the following:
  - $P(A \mid C) = 1 - P(\neg A \mid C)$.
  - $P(A \mid C) = P(A \wedge B \mid C) + P(A \wedge \neg B \mid C)$.
  - $P(A \vee B \mid C) = P(A \mid C) + P(B \mid C) - P(A \wedge B \mid C)$.
- But **not on the right hand side!**
- **Problem 1.5**
  Find *counterexamples* for the following (**false**) claims:
  - $P(A \mid C) = 1 - P(A \mid \neg C)$
  - $P(A \mid C) = P(A \mid (B \wedge C)) + P(A \mid (B \wedge \neg C))$.
  - $P(A \mid (B \vee C)) = P(A \mid B) + P(A \mid C) - P(A \mid (B \wedge C))$.

SOME RIGHTS RESERVED

Bayes' Rule

- **Note:** By definition, $P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$. In practice, we often know the conditional probability already, and use it to compute the probability of the conjunction instead: $P(A \wedge B) = P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A)$.

SOME RIGHTS RESERVED

Bayes' Rule

- **Note:** By definition, $P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$. In practice, we often know the conditional probability already, and use it to compute the probability of the conjunction instead: $P(A \wedge B) = P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A)$.

- **Theorem 1.23 (Bayes' Theorem).** *Given propositions A and B where $P(A) \neq 0$ and $P(B) \neq 0$, we have:*

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

- *Proof:*
  1. $P(A \mid B) = \frac{P(A \wedge B)}{P(B)} = \frac{P(B \mid A) \cdot P(A)}{P(B)}$

☐

...okay, that was straightforward... what's the big deal?

SOME RIGHTS RESERVED

Bayes' Rule

- **Note:** By definition, $P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$. In practice, we often know the conditional probability already, and use it to compute the probability of the conjunction instead: $P(A \wedge B) = P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A)$.

- **Theorem 1.24 (Bayes' Theorem).** *Given propositions A and B where* $P(A) \neq 0$ *and* $P(B) \neq 0$*, we have:*

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

- *Proof:*
  1. $P(A \mid B) = \frac{P(A \wedge B)}{P(B)} = \frac{P(B \mid A) \cdot P(A)}{P(B)}$

☐

...okay, that was straightforward... what's the big deal?

- **(Somewhat Dubious) Claim:** Bayes' Rule is the entire scientific method condensed into a single equation!
- This is an extreme overstatement, but there is a grain of truth in it.

SOME RIGHTS RESERVED

Bayes' Theorem - Why the Hype?

- Say we have a *hypothesis* H about the world. (e.g. "The universe had a beginning")
- We have *some prior belief* $P(H)$.
- We gather *evidence* E. (e.g. "We observe a cosmic microwave background at 2.7K everywhere")
- Bayes' Rule tells us how to *update our belief* in H based on H's ability to *predict* E (the *likelihood* $P(E \mid H)$) – *and*, importantly, the ability of *competing hypotheses* to predict the *same* evidence. (This is actually how scientific hypotheses should be evaluated)

$$\underbrace{P(H \mid E)}_{\text{posterior}} = \frac{P(E \mid H) \cdot P(H)}{P(E)} = \frac{\overbrace{P(E \mid H)}^{\text{likelihood}} \cdot \overbrace{P(H)}^{\text{prior}}}{\underbrace{P(E \mid H)}_{\text{likelihood}}\underbrace{P(H)}_{\text{prior}} + \underbrace{P(E \mid \neg H)P(\neg H)}_{\text{competition}}}$$

…if I keep gathering evidence and update, ultimately the impact of the prior belief will diminish.

"You're entitled to your own priors, but not your own likelihoods"

SOME RIGHTS RESERVED

21.1.2 Independence

SOME RIGHTS RESERVED

Independence

- **Question:** What is the probability that $S = 7$ *and* the patient has a toothache?
  Or less contrived: What is the probability that the patient has a gingivitis *and* a cavity?
- **Definition 1.25.** Two events A and B are called independent, iff $P(A \wedge B) = P(A) \cdot P(B)$.
  Two random variables X_1, X_2 are called independent, iff for all $x_1 \in \mathrm{dom}(X_1)$ and $x_2 \in \mathrm{dom}(X_2)$, the events $X_1 = x_1$ and $X_2 = x_2$ are independent. We write $A \perp B$ or $X_1 \perp X_2$, respectively.
- **Theorem 1.26.** *Equivalently: Given events A and B with $P(B) \neq 0$, then A and B are independent iff $P(A \mid B) = P(A)$ (equivalently: $P(B \mid A) = P(B)$).*
- *Proof:*
  1. $\Rightarrow$
     By definition, $P(A \mid B) = \frac{P(A \wedge B)}{P(B)} = \frac{P(A) \cdot P(B)}{P(B)} = P(A)$,
  3. $\Leftarrow$
     Assume $P(A \mid B) = P(A)$.
     Then $P(A \wedge B) = P(A \mid B) \cdot P(B) = P(A) \cdot P(B)$.

  $\square$
- **Note:** Independence asserts that two events are *"not related"* – the probability of one does not depend on the other.

SOME RIGHTS RESERVED

Independence (Examples)

- **Example 1.27.**
  - $\mathrm{First} = 2$ and $\mathrm{Second} = 3$ are independent – more generally, First and Second are independent (The outcome of the first die does not affect the outcome of the second die)
    Quick check:
    $P((\mathrm{First} = a) \wedge (\mathrm{Second} = b)) = \frac{1}{36} = P(\mathrm{First} = a) \cdot P(\mathrm{Second} = b)$ ✓
  - First and S are **not** independent. (The outcome of the first die affects the sum of the two dice.) Counterexample:
    $P((\mathrm{First} = 1) \wedge (S = 4)) = \frac{1}{36} \neq P(\mathrm{First} = 1) \cdot P(S = 4) = \frac{1}{6} \cdot \frac{1}{2} = \frac{1}{72}$

SOME RIGHTS RESERVED

Independence (Examples)

- **Example 1.29.**
  - $\mathrm{First} = 2$ and $\mathrm{Second} = 3$ are independent – more generally, First and Second are independent (The outcome of the first die does not affect the outcome of the second die)
    Quick check:
    $P((\mathrm{First} = a) \wedge (\mathrm{Second} = b)) = \frac{1}{36} = P(\mathrm{First} = a) \cdot P(\mathrm{Second} = b)$ ✓
  - First and S are **not** independent. (The outcome of the first die affects the sum of the two dice.) Counterexample:
    $P((\mathrm{First} = 1) \wedge (S = 4)) = \frac{1}{36} \neq P(\mathrm{First} = 1) \cdot P(S = 4) = \frac{1}{6} \cdot \frac{1}{2} = \frac{1}{72}$
  - **But:** $P((\mathrm{First} = a) \wedge (S = 7)) = \frac{1}{36} = \frac{1}{6} \cdot \frac{1}{6} = P(\mathrm{First} = a) \cdot P(S = 7)$ – so the events $\mathrm{First} = a$ and $S = 7$ *are* independent. (Why?)

SOME RIGHTS RESERVED

Independence (Examples)

- **Example 1.31.**
  - $\mathrm{First}=2$ and $\mathrm{Second}=3$ are independent – more generally, First and Second are independent (The outcome of the first die does not affect the outcome of the second die)
    Quick check:
    $P((\mathrm{First}=a)\wedge(\mathrm{Second}=b))=\frac{1}{36}=P(\mathrm{First}=a)\cdot P(\mathrm{Second}=b)$ ✓
  - First and S are **not** independent. (The outcome of the first die affects the sum of the two dice.) Counterexample:
    $P((\mathrm{First}=1)\wedge(S=4))=\frac{1}{36}\neq P(\mathrm{First}=1)\cdot P(S=4)=\frac{1}{6}\cdot\frac{1}{2}=\frac{1}{72}$
  - **But:** $P((\mathrm{First}=a)\wedge(S=7))=\frac{1}{36}=\frac{1}{6}\cdot\frac{1}{6}=P(\mathrm{First}=a)\cdot P(S=7)$ – so the events $\mathrm{First}=a$ and $S=7$ *are* independent. (Why?)
- **Example 1.32.**
  - Are cavity and $\mathrm{toothache}$ independent?

SOME RIGHTS RESERVED

Independence (Examples)

- **Example 1.33.**
  - $\mathrm{First}=2$ and $\mathrm{Second}=3$ are independent – more generally, First and Second are independent (The outcome of the first die does not affect the outcome of the second die)
    Quick check:
    $P((\mathrm{First}=a)\wedge(\mathrm{Second}=b))=\frac{1}{36}=P(\mathrm{First}=a)\cdot P(\mathrm{Second}=b)$ ✓
  - First and S are **not** independent. (The outcome of the first die affects the sum of the two dice.) Counterexample:
    $P((\mathrm{First}=1)\wedge(S=4))=\frac{1}{36}\neq P(\mathrm{First}=1)\cdot P(S=4)=\frac{1}{6}\cdot\frac{1}{2}=\frac{1}{72}$
  - **But:** $P((\mathrm{First}=a)\wedge(S=7))=\frac{1}{36}=\frac{1}{6}\cdot\frac{1}{6}=P(\mathrm{First}=a)\cdot P(S=7)$ – so the events $\mathrm{First}=a$ and $S=7$ *are* independent. (Why?)
- **Example 1.34.**
  - Are cavity and toothache independent?
    ...since cavities can cause a toothache, that would probably be a bad design decision ...

SOME RIGHTS RESERVED

Independence (Examples)

- **Example 1.35.**
  - $\mathrm{First} = 2$ and $\mathrm{Second} = 3$ are independent – more generally, First and Second are independent (The outcome of the first die does not affect the outcome of the second die)

    Quick check:

    $P((\mathrm{First} = a) \wedge (\mathrm{Second} = b)) = \frac{1}{36} = P(\mathrm{First} = a) \cdot P(\mathrm{Second} = b)$ ✓
  - First and S are **not** independent. (The outcome of the first die affects the sum of the two dice.) Counterexample:

    $P((\mathrm{First} = 1) \wedge (S = 4)) = \frac{1}{36} \neq P(\mathrm{First} = 1) \cdot P(S = 4) = \frac{1}{6} \cdot \frac{1}{2} = \frac{1}{72}$
  - **But:** $P((\mathrm{First} = a) \wedge (S = 7)) = \frac{1}{36} = \frac{1}{6} \cdot \frac{1}{6} = P(\mathrm{First} = a) \cdot P(S = 7)$ – so the events $\mathrm{First} = a$ and $S = 7$ *are* independent. (Why?)
- **Example 1.36.**
  - Are cavity and $\mathrm{toothache}$ independent?

    . . . since cavities can cause a toothache, that would probably be a bad design decision . . .
  - Are cavity and $\mathrm{gingivitis}$ independent? Cavities do not cause gingivitis, and gingivitis does not cause cavities, so... yes... right? (...as far as I know. I'm not a dentist.)

SOME RIGHTS RESERVED

Independence (Examples)

- **Example 1.37.**
  - $\mathrm{First} = 2$ and $\mathrm{Second} = 3$ are independent – more generally, First and Second are independent (The outcome of the first die does not affect the outcome of the second die)
    Quick check:
    $P((\mathrm{First} = a) \wedge (\mathrm{Second} = b)) = \frac{1}{36} = P(\mathrm{First} = a) \cdot P(\mathrm{Second} = b)$ ✓
  - First and S are **not** independent. (The outcome of the first die affects the sum of the two dice.) Counterexample:
    $P((\mathrm{First} = 1) \wedge (S = 4)) = \frac{1}{36} \neq P(\mathrm{First} = 1) \cdot P(S = 4) = \frac{1}{6} \cdot \frac{1}{2} = \frac{1}{72}$
  - **But:** $P((\mathrm{First} = a) \wedge (S = 7)) = \frac{1}{36} = \frac{1}{6} \cdot \frac{1}{6} = P(\mathrm{First} = a) \cdot P(S = 7)$ – so the events $\mathrm{First} = a$ and $S = 7$ *are* independent. (Why?)
- **Example 1.38.**
  - Are cavity and toothache independent?
    . . . since cavities can cause a toothache, that would probably be a bad design decision . . .
  - Are cavity and gingivitis independent? Cavities do not cause gingivitis, and gingivitis does not cause cavities, so... yes... right? (...as far as I know. I'm not a dentist.)
  - **Probably not!** A patient who has cavities has probably worse dental hygiene than those who don't, and is thus more likely to have gingivitis as well.
  - ⇝ cavity may be *evidence* that raises the probabilty of gingivitis, even if they are not directly causally related.

SOME RIGHTS RESERVED

g

Conditional Independence – Motivation

- A dentist can diagnose a cavity by using a *probe*, which may (or may not) *catch* in a cavity.
- Say we know from clinical studies that $P(\mathrm{cavity}) = 0.2$, $P(\mathrm{toothache} \mid \mathrm{cavity}) = 0.6$, $P(\mathrm{toothache} \mid \neg\mathrm{cavity}) = 0.1$, $P(\mathrm{catch} \mid \mathrm{cavity}) = 0.9$, and $P(\mathrm{catch} \mid \neg\mathrm{cavity}) = 0.2$.
- Assume the patient complains about a toothache, and our probe indeed catches in the aching tooth. What is the likelihood of having a cavity $P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch}))$?

SOME RIGHTS RESERVED

Conditional Independence – Motivation

- A dentist can diagnose a cavity by using a *probe*, which may (or may not) *catch* in a cavity.
- Say we know from clinical studies that $P(\mathrm{cavity}) = 0.2$, $P(\mathrm{toothache} \mid \mathrm{cavity}) = 0.6$, $P(\mathrm{toothache} \mid \neg\mathrm{cavity}) = 0.1$, $P(\mathrm{catch} \mid \mathrm{cavity}) = 0.9$, and $P(\mathrm{catch} \mid \neg\mathrm{cavity}) = 0.2$.
- Assume the patient complains about a toothache, and our probe indeed catches in the aching tooth. What is the likelihood of having a cavity $P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch}))$?
- **Idea:** Use Bayes' rule:

$$P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch})) = \frac{P(\mathrm{toothache} \wedge \mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity})}{P(\mathrm{toothache} \wedge \mathrm{catch})}$$

- **Note:** $P(\mathrm{toothache} \wedge \mathrm{catch}) = P(\mathrm{toothache} \wedge \mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity}) + P(\mathrm{toothache} \wedge \mathrm{catch} \mid \neg\mathrm{cavity}) \cdot P(\neg\mathrm{cavity})$
- **Problem:** Now we're only missing $P(\mathrm{toothache} \wedge \mathrm{catch} \mid (\mathrm{cavity} = b))$ for $b \in \{\mathsf{T}, \mathsf{F}\}$. …Now what?

Conditional Independence – Motivation

- A dentist can diagnose a cavity by using a *probe*, which may (or may not) *catch* in a cavity.
- Say we know from clinical studies that $P(\mathrm{cavity}) = 0.2$, $P(\mathrm{toothache} \mid \mathrm{cavity}) = 0.6$, $P(\mathrm{toothache} \mid \neg\mathrm{cavity}) = 0.1$, $P(\mathrm{catch} \mid \mathrm{cavity}) = 0.9$, and $P(\mathrm{catch} \mid \neg\mathrm{cavity}) = 0.2$.
- Assume the patient complains about a toothache, and our probe indeed catches in the aching tooth. What is the likelihood of having a cavity $P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch}))$?
- **Idea:** Use Bayes' rule:

$$P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch})) = \frac{P(\mathrm{toothache} \wedge \mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity})}{P(\mathrm{toothache} \wedge \mathrm{catch})}$$

- **Note:** $P(\mathrm{toothache} \wedge \mathrm{catch}) = P(\mathrm{toothache} \wedge \mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity}) + P(\mathrm{toothache} \wedge \mathrm{catch} \mid \neg\mathrm{cavity}) \cdot P(\neg\mathrm{cavity})$
- **Problem:** Now we're only missing $P(\mathrm{toothache} \wedge \mathrm{catch} \mid (\mathrm{cavity} = b))$ for $b \in \{\mathsf{T}, \mathsf{F}\}$. …Now what?
- Are toothache and catch independent, maybe?

Conditional Independence – Motivation

- A dentist can diagnose a cavity by using a *probe*, which may (or may not) *catch* in a cavity.
- Say we know from clinical studies that $P(\mathrm{cavity}) = 0.2$, $P(\mathrm{toothache} \mid \mathrm{cavity}) = 0.6$, $P(\mathrm{toothache} \mid \neg\mathrm{cavity}) = 0.1$, $P(\mathrm{catch} \mid \mathrm{cavity}) = 0.9$, and $P(\mathrm{catch} \mid \neg\mathrm{cavity}) = 0.2$.
- Assume the patient complains about a toothache, and our probe indeed catches in the aching tooth. What is the likelihood of having a cavity $P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch}))$?
- **Idea:** Use Bayes' rule:

$$P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch})) = \frac{P(\mathrm{toothache} \wedge \mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity})}{P(\mathrm{toothache} \wedge \mathrm{catch})}$$

- **Note:** $P(\mathrm{toothache} \wedge \mathrm{catch}) = P(\mathrm{toothache} \wedge \mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity}) + P(\mathrm{toothache} \wedge \mathrm{catch} \mid \neg\mathrm{cavity}) \cdot P(\neg\mathrm{cavity})$
- **Problem:** Now we're only missing $P(\mathrm{toothache} \wedge \mathrm{catch} \mid (\mathrm{cavity} = b))$ for $b \in \{\mathrm{T}, \mathrm{F}\}$. …Now what?
- Are toothache and catch independent, maybe? **No**: Both have a common (possible) cause, cavity.

  Also, there's this pesky $P(\cdot \mid \mathrm{cavity})$ in the way.…..wait a minute...

SOME RIGHTS RESERVED

Conditional Independence – Definition

- *Assuming* the patient has (or does not have) a cavity, the events toothache and catch are independent: Both are caused by a cavity, but they don't influence each other otherwise.
  i.e. cavity "contains all the information" that links toothache and catch in the first place.

SOME RIGHTS RESERVED

Conditional Independence – Definition

- *Assuming* the patient has (or does not have) a cavity, the events toothache and catch are independent: Both are caused by a cavity, but they don't influence each other otherwise.
  i.e. cavity "contains all the information" that links toothache and catch in the first place.

- **Definition 1.41.** Given events A, B, C with $P(C) \neq 0$, then A and B are called conditionally independent given C, iff $P(A \wedge B \mid C) = P(A \mid C) \cdot P(B \mid C)$.
  Equivalently: iff $P(A \mid (B \wedge C)) = P(A \mid C)$, or $P(B \mid (A \wedge C)) = P(B \mid C)$.

  Let Y be a random variable. We call two random variables X_1, X_2 conditionally independent given Y, iff for all $x_1 \in \mathrm{dom}(X_1)$, $x_2 \in \mathrm{dom}(X_2)$ and $y \in \mathrm{dom}(Y)$, the events $X_1 = x_1$ and $X_2 = x_2$ are conditionally independent given $Y = y$.

SOME RIGHTS RESERVED

Conditional Independence – Definition

- *Assuming* the patient has (or does not have) a cavity, the events toothache and catch are independent: Both are caused by a cavity, but they don't influence each other otherwise.
  i.e. cavity "contains all the information" that links toothache and catch in the first place.

- **Definition 1.43.** Given events A, B, C with $P(C) \neq 0$, then A and B are called conditionally independent given C, iff $P(A \wedge B \mid C) = P(A \mid C) \cdot P(B \mid C)$.
  Equivalently: iff $P(A \mid (B \wedge C)) = P(A \mid C)$, or $P(B \mid (A \wedge C)) = P(B \mid C)$.

  Let Y be a random variable. We call two random variables X_1, X_2 conditionally independent given Y, iff for all $x_1 \in \mathrm{dom}(X_1)$, $x_2 \in \mathrm{dom}(X_2)$ and $y \in \mathrm{dom}(Y)$, the events $X_1 = x_1$ and $X_2 = x_2$ are conditionally independent given $Y = y$.

- **Example 1.44.** Let's assume toothache and catch are conditionally independent given cavity/¬cavity. Then we can finally compute:
  $P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch})) = \frac{P(\mathrm{toothache} \wedge \mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity})}{P(\mathrm{toothache} \wedge \mathrm{catch})}$
  $= \frac{P(\mathrm{toothache} \mid \mathrm{cavity}) \cdot P(\mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity})}{P(\mathrm{toothache} \mid \mathrm{cavity}) \cdot P(\mathrm{catch} \mid \mathrm{cavity}) \cdot P(\mathrm{cavity}) + P(\mathrm{toothache} \mid \neg\mathrm{cavity}) \cdot P(\mathrm{catch} \mid \neg\mathrm{cavity}) \cdot P(}$
  $= \frac{0.6 \cdot 0.9 \cdot 0.2}{0.6 \cdot 0.9 \cdot 0.2 + 0.1 \cdot v0.2 \cdot 0.8} = 0.87$

SOME RIGHTS RESERVED

Conditional Independence

- **Lemma 1.45.** *If A and B are conditionally independent given C, then* $P(A \mid (B \wedge C)) = P(A \mid C)$

  Proof:

  $P(A \mid (B \wedge C)) = \frac{P(A \wedge B \wedge C)}{P(B \wedge C)} = \frac{P(A \wedge B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \wedge C)}{P(B \wedge C)} = P(A \mid C)$

  □

- **Question:** If A and B are conditionally independent given C, does this imply that A and B are independent?

SOME RIGHTS RESERVED

Conditional Independence

- **Lemma 1.46.** *If A and B are conditionally independent given C, then* $P(A \mid (B \wedge C)) = P(A \mid C)$
  Proof:
  $P(A \mid (B \wedge C)) = \frac{P(A \wedge B \wedge C)}{P(B \wedge C)} = \frac{P(A \wedge B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \wedge C)}{P(B \wedge C)} = P(A \mid C)$

  ☐

- **Question:** If A and B are conditionally independent given C, does this imply that A and B are independent? **No.** See previous slides for a counterexample.
- **Question:** If A and B are independent, does this imply that A and B are also conditionally independent given C?

SOME RIGHTS RESERVED

Conditional Independence

- **Lemma 1.47.** *If A and B are conditionally independent given C, then* $P(A \mid (B \wedge C)) = P(A \mid C)$

  Proof:
  $P(A \mid (B \wedge C)) = \frac{P(A \wedge B \wedge C)}{P(B \wedge C)} = \frac{P(A \wedge B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \wedge C)}{P(B \wedge C)} = P(A \mid C)$

  ☐

- **Question:** If A and B are conditionally independent given C, does this imply that A and B are independent? **No.** See previous slides for a counterexample.
- **Question:** If A and B are independent, does this imply that A and B are also conditionally independent given C? **No.** For example: First and Second are independent, but not conditionally independent given $S = 4$.

SOME RIGHTS RESERVED

Conditional Independence

- **Lemma 1.48.** *If A and B are conditionally independent given C, then* $P(A\mid (B\wedge C))=P(A\mid C)$

  Proof:

  $$P(A\mid (B\wedge C))=\frac{P(A\wedge B\wedge C)}{P(B\wedge C)}=\frac{P(A\wedge B\mid C)\cdot P(C)}{P(B\wedge C)}=\frac{P(A\mid C)\cdot P(B\mid C)\cdot P(C)}{P(B\wedge C)}=\frac{P(A\mid C)\cdot P(B\wedge C)}{P(B\wedge C)}=P(A\mid C)$$

  ☐

- **Question:** If A and B are conditionally independent given C, does this imply that A and B are independent? **No.** See previous slides for a counterexample.
- **Question:** If A and B are independent, does this imply that A and B are also conditionally independent given C? **No.** For example: First and Second are independent, but not conditionally independent given $S=4$.
- **Question:** Okay, so what if A, B and C are *all* pairwise independent? Are A and B conditionally independent given C *now*?

SOME RIGHTS RESERVED

Conditional Independence

- **Lemma 1.49.** *If A and B are conditionally independent given C, then* $P(A \mid (B \wedge C)) = P(A \mid C)$

  Proof:

  $$P(A \mid (B \wedge C)) = \frac{P(A \wedge B \wedge C)}{P(B \wedge C)} = \frac{P(A \wedge B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \wedge C)}{P(B \wedge C)} = P(A \mid C)$$

  □

- **Question:** If A and B are conditionally independent given C, does this imply that A and B are independent? **No.** See previous slides for a counterexample.
- **Question:** If A and B are independent, does this imply that A and B are also conditionally independent given C? **No.** For example: First and Second are independent, but not conditionally independent given $S = 4$.
- **Question:** Okay, so what if A, B and C are *all* pairwise independent? Are A and B conditionally independent given C *now*? **Still no.** Remember: $\mathrm{First} = a$, $\mathrm{Second} = b$ and $S = 7$ are all independent, but First and Second are not conditionally independent given $S = 7$.
- **Question:** When can we infer conditional independence from a “more general” notion of independence?

SOME RIGHTS RESERVED

Conditional Independence

- **Lemma 1.50.** *If A and B are conditionally independent given C, then* $P(A \mid (B \wedge C)) = P(A \mid C)$

  Proof:

  $$P(A \mid (B \wedge C)) = \frac{P(A \wedge B \wedge C)}{P(B \wedge C)} = \frac{P(A \wedge B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \mid C) \cdot P(C)}{P(B \wedge C)} = \frac{P(A \mid C) \cdot P(B \wedge C)}{P(B \wedge C)} = P(A \mid C)$$

  ☐

- **Question:** If A and B are conditionally independent given C, does this imply that A and B are independent? **No.** See previous slides for a counterexample.
- **Question:** If A and B are independent, does this imply that A and B are also conditionally independent given C? **No.** For example: First and Second are independent, but not conditionally independent given $S = 4$.
- **Question:** Okay, so what if A, B and C are *all* pairwise independent? Are A and B conditionally independent given C *now*? **Still no.** Remember: $\text{First} = a$, $\text{Second} = b$ and $S = 7$ are all independent, but First and Second are not conditionally independent given $S = 7$.
- **Question:** When can we infer conditional independence from a "more general" notion of independence?

  We need *mutual independence*. Roughly: A set of events is called *mutually* independent, if every event is independent from *any conjunction of the oth*

SOME RIGHTS RESERVED

21.1.3 Conclusion

SOME RIGHTS RESERVED

Summary

- Probability spaces serve as a mathematical model (and hence justification) for everything related to probabilities.
- The "atoms" of any statement of probability are the random variables. (Important special cases: Boolean and finite domain)
- We can define probabilities on compund (propositional logical) statements, with (outcomes of) random variables as "propositional variables".
- Conditional probabilities represent *posterior probabilities* given some observed outcomes.
- Independence and conditional independence are strong assumptions that allow us to simplify computations of probabilities
- Bayes' Theorem (can be used between "causal" and "diagnostic" conditional probabilities)

SOME RIGHTS RESERVED

So much about the math...

- We now have a mathematical setup for probabilities.
- **But:** The math does not tell us what probabilities *are*:
- Assume we can mathematically derive this to be the case: *the probability of rain tomorrow is* 0.3. What does this even *mean*?

SOME RIGHTS RESERVED

So much about the math...

- We now have a mathematical setup for probabilities.
- **But:** The math does not tell us what probabilities *are*:
- Assume we can mathematically derive this to be the case: *the probability of rain tomorrow is* 0.3. What does this even *mean*?
- **Frequentist Answer:** The probability of an event is the limit of its relative frequency in a large number of trials.
  In other words: "In 30% of the cases where we have similar weather conditions, it rained the next day."

SOME RIGHTS RESERVED

So much about the math...

- We now have a mathematical setup for probabilities.
- **But:** The math does not tell us what probabilities *are*:
- Assume we can mathematically derive this to be the case: *the probability of rain tomorrow is* 0.3. What does this even *mean*?
- **Frequentist Answer:** The probability of an event is the limit of its relative frequency in a large number of trials.
  In other words: "In 30% of the cases where we have similar weather conditions, it rained the next day."
- **Objection:** Okay, but what about *unique* events? "The probability of me passing the exam is 80%" – does this mean anything, if I only take the exam once? Am I comparable to "similar students"? What counts as sufficiently "similar"?

SOME RIGHTS RESERVED

So much about the math...

- We now have a mathematical setup for probabilities.
- **But:** The math does not tell us what probabilities *are*:
- Assume we can mathematically derive this to be the case: *the probability of rain tomorrow is* 0.3. What does this even *mean*?
- **Frequentist Answer:** The probability of an event is the limit of its relative frequency in a large number of trials.
  In other words: "In 30% of the cases where we have similar weather conditions, it rained the next day."
- **Objection:** Okay, but what about *unique* events? "The probability of me passing the exam is 80%" – does this mean anything, if I only take the exam once? Am I comparable to "similar students"? What counts as sufficiently "similar"?
- **Bayesian Answer:** Probabilities are *degrees of belief*. It means you **should** be 30% confident that it will rain tomorrow.

SOME RIGHTS RESERVED

So much about the math...

- We now have a mathematical setup for probabilities.
- **But:** The math does not tell us what probabilities *are*:
- Assume we can mathematically derive this to be the case: *the probability of rain tomorrow is* 0.3. What does this even *mean*?
- **Frequentist Answer:** The probability of an event is the limit of its relative frequency in a large number of trials.
  In other words: "In 30% of the cases where we have similar weather conditions, it rained the next day."
- **Objection:** Okay, but what about *unique* events? "The probability of me passing the exam is 80%" – does this mean anything, if I only take the exam once? Am I comparable to "similar students"? What counts as sufficiently "similar"?
- **Bayesian Answer:** Probabilities are *degrees of belief*. It means you **should** be 30% confident that it will rain tomorrow.
- **Objection:** And why *should* I? Is this not purely *subjective* then?

SOME RIGHTS RESERVED

Pragmatics

- **Pragmatically** both interpretations amount to the same thing: I should *act as if* I'm 30% confident that it will rain tomorrow. (Whether by fiat, or because in 30% of comparable cases, it rained.)
- **Objection:** Still: why should I? And why should my beliefs follow the seemingly arbitrary Kolmogorov axioms?

SOME RIGHTS RESERVED

Pragmatics

- **Pragmatically** both interpretations amount to the same thing: I should *act as if* I'm 30% confident that it will rain tomorrow. (Whether by fiat, or because in 30% of comparable cases, it rained.)
- **Objection:** Still: why should I? And why should my beliefs follow the seemingly arbitrary Kolmogorov axioms?
- [**deFinetti:sssdp31**]: If an agent has a belief that violates the Kolmogorov axioms, then there exists a combination of "bets" on propositions so that the agent *always* loses money.
- **In other words:** If your beliefs are not consistent with the mathematics, and you *act in accordance with your beliefs*, there is a way to exploit this inconsistency to your disadvantage.

SOME RIGHTS RESERVED

Pragmatics

- **Pragmatically** both interpretations amount to the same thing: I should *act as if* I'm 30% confident that it will rain tomorrow. (Whether by fiat, or because in 30% of comparable cases, it rained.)
- **Objection:** Still: why should I? And why should my beliefs follow the seemingly arbitrary Kolmogorov axioms?
- [**deFinetti:sssdp31**]: If an agent has a belief that violates the Kolmogorov axioms, then there exists a combination of "bets" on propositions so that the agent *always* loses money.
- **In other words:** If your beliefs are not consistent with the mathematics, and you *act in accordance with your beliefs*, there is a way to exploit this inconsistency to your disadvantage.
- . . . and, more importantly, the AI agents you design! ☺
- **I (and my agents) do not bet:** That is not true, in a partially observable or non-deterministic world, every action choice is a necessarily bet: The outcome is not sure.

SOME RIGHTS RESERVED

21.2 Probabilistic Reasoning Techniques

SOME RIGHTS RESERVED

Okay, now how do I implement this?

- This is a CS course. We need to implement this stuff.
- Do we... implement random variables as functions? Is a probability space a... class maybe?

SOME RIGHTS RESERVED

Okay, now how do I implement this?

- This is a CS course. We need to implement this stuff.
- Do we... implement random variables as functions? Is a probability space a... class maybe?
- **No:** As mentioned, we rarely know the probability space entirely. Instead we will use probability distributions, which are just arrays (of arrays of...) of probabilities.
- And then we represent *those* as sparsely as possible, by exploiting independence, conditional independence, . . .

SOME RIGHTS RESERVED

21.2.1 Probability Distributions

Probability Distributions

- **Definition 2.1.** The probability distribution for a random variable X, written $\mathbb{P}(X)$, is the vector of probabilities for the (ordered) domain of X.
- **Note:** The values in a probability distribution are all positive and sum to 1. (Why?)
- **Example 2.2.** $\mathbb{P}(\mathrm{First}) = \mathbb{P}(\mathrm{Second}) = \langle\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\rangle$. (Both First and Second are uniformly distributed)
- **Example 2.3.** The probability distribution $\mathbb{P}(S)$ is $\langle\frac{1}{36}, \frac{1}{18}, \frac{1}{12}, \frac{1}{9}, \frac{5}{36}, \frac{1}{6}, \frac{5}{36}, \frac{1}{9}, \frac{1}{12}, \frac{1}{18}, \frac{1}{36}\rangle$. Note the symmetry, with a "peak" at 7 – the random variable is (*approximately*, because our domain is discrete rather than continuous) normally distributed (or gaussian distributed, or follows a bell-curve,...).
- **Example 2.4.** Probability distributions for Boolean random variables are naturally *pairs* (probabilities for T and F), e.g.:

$$\mathbb{P}(\mathrm{toothache}) = \langle 0.15, 0.85\rangle$$
$$\mathbb{P}(\mathrm{cavity}) = \langle 0.122, 0.878\rangle$$

- More generally:
- **Definition 2.5.** A probability distribution is a vector v of values $v_i \in [0,1]$ such that $\sum_i v_i = 1$.

SOME RIGHTS RESERVED

The Full Joint Probability Distribution

- **Definition 2.6.** Given random variables $X_1, \ldots, X_n$ with domains $D_1, \ldots, D_n$, the full joint probability distribution, denoted $\mathbb{P}(X_1, \ldots, X_n)$, is the n-dimensional array of size $|D_1 \times \ldots \times D_n|$ that lists the probabilities of all conjunctions of values of the random variables.
- **Example 2.7.** $\mathbb{P}(\mathrm{cavity}, \mathrm{toothache}, \mathrm{gingivitis})$ could look something like this:

	toothache		¬toothache	
	gingivitis	¬gingivitis	gingivitis	¬gingivitis
cavity	0.007	0.06	0.005	0.05
¬cavity	0.08	0.003	0.045	0.75

- **Example 2.8.** $\mathbb{P}(\mathrm{First}, S)$

First \ S	2	3	4	5	6	7	8	9	10	11	12
1	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	0	0	0	0
2	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	0	0	0
3	0	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	0	0
4	0	0	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	0
5	0	0	0	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0
6	0	0	0	0	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$

Note that if we know the value of First, the value of S is completely determined by the value of Second.

SOME RIGHTS RESERVED

Conditional Probability Distributions

- **Definition 2.9.** Given random variables X and Y, the conditional probability distribution of X given Y, written $\mathbb{P}(X|Y)$ is the table of all conditional probabilities of values of X given values of Y.
- For sets of variables analogously: $\mathbb{P}(X_1,\ldots,X_n|Y_1,\ldots,Y_m)$.
- **Example 2.10.** $\mathbb{P}(\mathrm{cavity}|\mathrm{toothache})$:

	toothache	¬toothache
cavity	$P(\mathrm{cavity} \mid \mathrm{toothache}) = 0.45$	$P(\mathrm{cavity} \mid \neg\mathrm{toothache}) = 0.065$
¬cavity	$P(\neg\mathrm{cavity} \mid \mathrm{toothache}) = 0.55$	$P(\neg\mathrm{cavity} \mid \neg\mathrm{toothache}) = 0.935$

- **Example 2.11.** $\mathbb{P}(\mathrm{First}|S)$

$\mathrm{First} \setminus S$	2	3	4	5	6	7	8	9	10	11	12
1	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	0	0	0	0	0
2	0	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{5}$	0	0	0	0
3	0	0	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{4}$	0	0	0
4	0	0	0	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	0	0
5	0	0	0	0	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	0
6	0	0	0	0	0	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	1

- **Note:** Every "column" of a conditional probability distribution is itself a probability distribution. (Why?)

Convention

- We now "lift" multiplication and division to the level of whole probability distributions:
- **Definition 2.12.** Whenever we use $\mathbb{P}$ in an equation, we take this to mean a *system of equations*, for each value in the domains of the random variables involved.

Example 2.13.

- $\mathbb{P}(X, Y) = \mathbb{P}(X|Y) \cdot \mathbb{P}(Y)$ represents the system of equations $P(X = x \wedge Y = y) = P(X = x \mid Y = y) \cdot P(Y = y)$ for all x, y in the respective domains.
- $\mathbb{P}(X|Y) := \frac{\mathbb{P}(X,Y)}{\mathbb{P}(Y)}$ represents the system of equations $P(X = x \mid (Y = y)) := \frac{P((X=x) \wedge (Y=y))}{P(Y=y)}$
- Bayes' Theorem: $\mathbb{P}(X|Y) = \frac{\mathbb{P}(Y|X) \cdot \mathbb{P}(X)}{\mathbb{P}(Y)}$ represents the system of equations $P(X = x \mid (Y = y)) = \frac{P(Y=y \mid (X=x)) \cdot P(X=x)}{P(Y=y)}$

SOME RIGHTS RESERVED

So, what's the point?

- Obviously, the probability distribution contains all the information about a specific random variable we need.
- **Observation:** The full joint probability distribution of variables $X_1, \ldots, X_n$ contains *all* the information about the random variables *and their conjunctions* we need.
- **Example 2.14.** We can read off the probability $P(\mathrm{toothache})$ from the full joint probability distribution as $0.007 + 0.06 + 0.08 + 0.003 = 0.15$, and the probability $P(\mathrm{toothache} \wedge \mathrm{cavity})$ as $0.007 + 0.06 = 0.067$
- We can actually implement this! (They're just (nested) arrays)
- **But:** just as we often don't have a fully specified probability space to work in, we often don't have a full joint probability distribution for our random variables either.
- **Also:** Given random variables $X_1, \ldots, X_n$, the full joint probability distribution has $\prod_{i=1}^{n} |\mathrm{dom}(X_i)|$ entries! ($\mathbb{P}(\mathrm{First}, S)$ already has 60 entries!)
- **So:** The rest of this section deals with keeping things small, by *computing* probabilities instead of *storing* them all.

SOME RIGHTS RESERVED

Probabilistic Reasoning

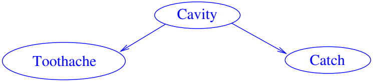
- **Probabilistic reasoning** refers to inferring probabilities of events from the probabilities of other events
  as opposed to determining the probabilities e.g. *empirically*, by gathering (sufficient amounts of *representative*) data and counting.
- **Note:** In practice, we are *primarily* interested in, and have access to, conditional probabilities rather than the unconditional probabilities of conjunctions of events:
  - We don't reason in a vacuum: Usually, we have some evidence and want to infer the posterior probability of some related event. (e.g. infer a plausible *cause* given some *symptom*)
    $\leadsto$ we are interested in the conditional probability $P(\mathrm{hypothesis} \mid \mathrm{observation})$.
  - "*80% of patients with a cavity complain about a toothache*" (i.e. $P(\mathrm{toothache} \mid \mathrm{cavity})$) is more the kind of data people actually collect and publish than "*1.2% of the general population have both a cavity and a toothache*" (i.e. $P(\mathrm{cavity} \wedge \mathrm{toothache})$).
  - Consider the probe catching in a cavity. The probe is a diagnostic tool, which is usually evaluated in terms of its *sensitivity* $P(\mathrm{catch} \mid \mathrm{cavity})$ and *specificity* $P(\neg\mathrm{catch} \mid \neg\mathrm{cavity})$. (You have probably heard these words a lot since 2020...)

SOME RIGHTS RESERVED

21.2.2 Naive Bayes

Naive Bayes Models

- Consider again the dentistry example with random variables cavity, $\mathrm{toothache}$, and catch. We assume cavity **causes** both $\mathrm{toothache}$ and catch, and that $\mathrm{toothache}$ and catch are conditionally independent given cavity:

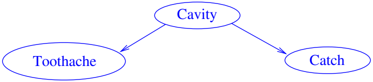


- We likely know the *sensitivity* $P(\mathrm{catch} \mid \mathrm{cavity})$ and *specificity* $P(\neg\mathrm{catch} \mid \neg\mathrm{cavity})$, which jointly give us $\mathbb{P}(\mathrm{catch}|\mathrm{cavity})$, and from medical studies, we should be able to determine $P(\mathrm{cavity})$ (the *prevalence* of cavities in the population) and $\mathbb{P}(\mathrm{toothache}|\mathrm{cavity})$.

- This kind of situation is surprisingly common, and therefore deserves a name.

SOME RIGHTS RESERVED

Naive Bayes Models



- **Definition 2.15.** A naive Bayes model (or, less accurately, Bayesian classifier, or, derogatorily, idiot Bayes model) consists of:
  1. random variables $C, E_1, \ldots, E_n$ such that all the $E_1, \ldots, E_n$ are conditionally independent given C,
  2. the probability distribution $\mathbb{P}(C)$, and
  3. the conditional probability distributions $\mathbb{P}(E_i|C)$.

  We call C the cause and the $E_1, \ldots, E_n$ the effects of the model.
- **Convention:** Whenever we draw a graph of random variables, we take the arrows to connect *causes* to their direct *effects*, and assert that unconnected nodes are conditionally independent given all their ancestors. We will make this more precise later.
- Can we compute the full joint probability distribution $\mathbb{P}(\mathrm{cavity}, \mathrm{toothache}, \mathrm{catch})$ from this information?

SOME RIGHTS RESERVED

Recovering the Full Joint Probability Distribution

- **Lemma 2.16 (Product rule).** $\mathbb{P}(X, Y) = \mathbb{P}(X|Y) \cdot \mathbb{P}(Y)$.
- We can generalize this to more than two variables, by repeatedly applying the product rule:
- **Lemma 2.17 (Chain rule).** *For any sequence of random variables $X_1, \ldots, X_n$:*

$$\mathbb{P}(X_1, \ldots, X_n) = \mathbb{P}(X_1|X_2, \ldots, X_n) \cdot \mathbb{P}(X_2|X_3, \ldots X_n) \cdot \ldots \cdot \mathbb{P}(X_{n-1}|X_n) \cdot P(X_n)$$

  Hence:

- **Theorem 2.18.** *Given a naive Bayes model with effects $E_1, \ldots, E_n$ and cause C, we have*

$$\mathbb{P}(C, E_1, \ldots, E_n) = \mathbb{P}(C) \cdot (\prod_{i=1}^{n} \mathbb{P}(E_i|C)).$$

- *Proof:* Using the chain rule:
  1. $\mathbb{P}(E_1, \ldots, E_n, C) = \mathbb{P}(E_1|E_2, \ldots, E_n, C) \cdot \ldots \cdot \mathbb{P}(E_n|C) \cdot \mathbb{P}(C)$
  2. Since all the E_i are conditionally independent, we can drop them on the right hand sides of the $\mathbb{P}(E_j|..., C)$

☐

SOME RIGHTS RESERVED

Marginalization

- Great, so now we can compute $\mathbb{P}(C|E_1,\ldots,E_n) = \frac{\mathbb{P}(C,E_1,\ldots,E_n)}{\mathbb{P}(E_1,\ldots,E_n)}$...
  ...except that we don't know $\mathbb{P}(E_1,\ldots,E_n)$:-/
  ...except that we can compute the full joint probability distribution, so we can recover it:
- **Lemma 2.19 (Marginalization).** *Given random variables $X_1,\ldots,X_n$ and $Y_1,\ldots,Y_m$, we have* $\mathbb{P}(X_1,\ldots,X_n) = \sum_{y_1\in\mathrm{dom}(Y_1),\ldots,y_m\in\mathrm{dom}(Y_m)} \mathbb{P}(X_1,\ldots,X_n,Y_1=y_1,\ldots,Y_m=y_m)$.
  (This is just a fancy way of saying "we can add the relevant entries of the full joint probability distribution")
- **Example 2.20.** Say we observed $\mathrm{toothache}=\mathsf{T}$ and $\mathrm{catch}=\mathsf{T}$. Using marginalization, we can compute

$$\begin{aligned} P(\mathrm{cavity} \mid (\mathrm{toothache} \wedge \mathrm{catch})) &= \frac{P(\mathrm{cavity} \wedge \mathrm{toothache} \wedge \mathrm{catch})}{P(\mathrm{toothache} \wedge \mathrm{catch})} \\ &= \frac{P(\mathrm{cavity} \wedge \mathrm{toothache} \wedge \mathrm{catch})}{\sum_{c\in\{\mathrm{cavity},\neg\mathrm{cavity}\}} P(c \wedge \mathrm{toothache} \wedge \mathrm{catch})} \\ &= \frac{P(\mathrm{cavity}) \cdot P(\mathrm{toothache} \mid \mathrm{cavity}) \cdot P(\mathrm{catch} \mid}{\sum_{c\in\{\mathrm{cavity},\neg\mathrm{cavity}\}} P(c) \cdot P(\mathrm{toothache} \mid c) \cdot P(} \end{aligned}$$

SOME RIGHTS RESERVED

Unknowns

- What if we don't know catch? (I'm not a dentist, I don't have a probe...)
- We split our effects into $\{E_1, \ldots, E_n\} = \{O_1, \ldots, O_{n_O}\} \cup \{U_1, \ldots, U_{n_U}\}$ – the *observed* and *unknown* random variables.
- Let $D_U := \mathrm{dom}(U_1) \times \ldots \times \mathrm{dom}(U_{n_u})$. Then

$$\mathbb{P}(C|O_1, \ldots, O_{n_O}) = \frac{\mathbb{P}(C, O_1, \ldots, O_{n_O})}{\mathbb{P}(O_1, \ldots, O_{n_O})}$$

$$= \frac{\sum_{u \in D_U} \mathbb{P}(C, O_1, \ldots, O_{n_O}, U_1 = u_1, \ldots, U_{n_u} = u_{n_u})}{\sum_{c \in \mathrm{dom}(C)} \sum_{u \in D_U} \mathbb{P}(O_1, \ldots, O_{n_O}, C = c, U_1 = u_1, \ldots, U_{n_u} =}$$

$$= \frac{\sum_{u \in D_U} \mathbb{P}(C) \cdot (\prod_{i=1}^{n_O} \mathbb{P}(O_i|C)) \cdot (\prod_{j=1}^{n_U} \mathbb{P}(U_j =}{\sum_{c \in \mathrm{dom}(C)} \sum_{u \in D_U} P(C = c) \cdot (\prod_{i=1}^{n_O} \mathbb{P}(O_i|C = c)) \cdot (\prod_{j=1}^{n_U} P(}$$

$$= \frac{\mathbb{P}(C) \cdot (\prod_{i=1}^{n_O} \mathbb{P}(O_i|C)) \cdot (\sum_{u \in D_U} \prod_{j=1}^{n_U} \mathbb{P}(U_j =}{\sum_{c \in \mathrm{dom}(C)} P(C = c) \cdot (\prod_{i=1}^{n_O} \mathbb{P}(O_i|C = c)) \cdot (\sum_{u \in D_U} \prod_{j=1}^{n_U} P(}$$

...oof...

Unknowns

- Continuing from above:

$$\mathbb{P}(C|O_1,\ldots,O_{n_O}) = \frac{\mathbb{P}(C)\cdot(\prod_{i=1}^{n_O}\mathbb{P}(O_i|C))\cdot(\sum_{u\in D_U}\prod_{j=1}^{n_U}\mathbb{P}(U_j =}{\sum_{c\in\mathrm{dom}(C)}P(C=c)\cdot(\prod_{i=1}^{n_O}\mathbb{P}(O_i|C=c))\cdot(\sum_{u\in D_U}\prod_{j=1}^{n_U}P}$$

- First, note that $\sum_{u\in D_U}\prod_{j=1}^{n_U}P(U_j = u_j \mid (C=c)) = 1$ (We're summing over all possible events on the (conditionally independent) $U_1,\ldots,U_{n_U}$ given $C=c$)
-

$$\mathbb{P}(C|O_1,\ldots,O_{n_O}) = \frac{\mathbb{P}(C)\cdot(\prod_{i=1}^{n_O}\mathbb{P}(O_i|C))}{\sum_{c\in\mathrm{dom}(C)}P(C=c)\cdot(\prod_{i=1}^{n_O}\mathbb{P}(O_i|C=c))}$$

- Secondly, note that the *denominator* is
  1. the same for any given observations $O_1,\ldots,O_{n_O}$, independent of the value of C, and
  2. the *sum* over all the *numerators* in the full distribution.

  That is: The denominator only serves to *scale* what is *almost* already the distribution $\mathbb{P}(C|O_1,\ldots,O_{n_O})$ to sum up to 1.

SOME RIGHTS RESERVED

Normalization

- **Definition 2.21 (Normalization).** Given a vector $w := \langle w_1, \ldots, w_k \rangle$ of numbers in $[0,1]$ where $\sum_{i=1}^{k} w_i \leq 1$.
  Then the normalized vector $\alpha(w)$ is defined (component-wise) as

  $$(\alpha(w))_i := \frac{w_i}{\sum_{j=1}^{k} w_j}.$$

  Note that $\sum_{i=1}^{k} \alpha(w)_i = 1$, i.e. $\alpha(w)$ is a probability distribution.
- This finally gives us:
  Theorem 2.22 (Inference in a Naive Bayes model). *Let $C, E_1, \ldots, E_n$ a naive Bayes model and $E_1, \ldots, E_n = O_1, \ldots, O_{n_O}, U_1, \ldots, U_{n_U}$.*
  Then

  $$\mathbb{P}(C|O_1 = o_1, \ldots, O_{n_O} = o_{n_O}) = \alpha(\mathbb{P}(C) \cdot (\prod_{i=1}^{n_O} \mathbb{P}(O_i = o_i|C)))$$

- Note, that this is entirely independent of the *unknown* random variables $U_1, \ldots, U_{n_U}$!
- Also, note that this is just a fancy way of saying "first, compute all the numerators, then divide all of them by their sums".

SOME RIGHTS RESERVED

Dentistry Example

- Putting things together, we get:

$$\begin{aligned}\mathbb{P}(\mathrm{cavity}|\mathrm{toothache}=\mathsf{T}) &= \alpha(\mathbb{P}(\mathrm{cavity})\cdot\mathbb{P}(\mathrm{toothache}=\mathsf{T}|\mathrm{cavity}))\\ &= \alpha(\langle P(\mathrm{cavity})\cdot P(\mathrm{toothache}\mid\mathrm{cavity}), P(\neg\mathrm{cavity})\cdot P(\mathrm{t}\end{aligned}$$

- Say we have $P(\mathrm{cavity})=0.1$, $P(\mathrm{toothache}\mid\mathrm{cavity})=0.8$, and $P(\mathrm{toothache}\mid\neg\mathrm{cavity})=0.05$. Then

$$\mathbb{P}(\mathrm{cavity}|\mathrm{toothache}=\mathsf{T})=\alpha(\langle 0.1\cdot 0.8, 0.9\cdot 0.05\rangle)=\alpha(\langle 0.08, 0.045\rangle)$$

$0.08+0.045=0.125$, hence

$$\mathbb{P}(\mathrm{cavity}|\mathrm{toothache}=\mathsf{T})=\langle\frac{0.08}{0.125},\frac{0.045}{0.125}\rangle=\langle 0.64, 0.36\rangle$$

SOME RIGHTS RESERVED

Naive Bayes Classification

We can use a naive Bayes model as a very simple *classifier*:

- Assume we want to classify newspaper articles as one of the categories *politics*, *sports*, *business*, *fluff*, etc. based on the words they contain.
- Given a large set of articles, we can determine the relevant probabilities by counting the occurrences of the categories $\mathbb{P}(\mathrm{category})$, and of words per category – i.e. $\mathbb{P}(\mathrm{word}_i|\mathrm{category})$ for some (huge) list of words $(\mathrm{word}_i)_{i=1}^{n}$.
- We assume that the occurrence of each word is conditionally independent of the occurrence of any other word given the category of the document. (This assumption is clearly wrong, but it makes the model simple and often works well in practice.) ($\leadsto$ "Idiot Bayes model")
- Given a new article, we just count the occurrences k_i of the words in it and compute

$$\mathbb{P}(\mathrm{category}|\mathrm{word}_1 = k_1, \ldots, \mathrm{word}_n = k_n) = \alpha(\mathbb{P}(\mathrm{category}) \cdot (\prod_{i=1}^{n} \mathbb{P}(\mathrm{word}_i = k_i|\mathrm{categ}$$

- We then choose the category with the highest probability.

SOME RIGHTS RESERVED

21.2.3 Inference by Enumeration

SOME RIGHTS RESERVED

Inference by Enumeration

- The rules we established for naive Bayes models, i.e. Bayes's theorem, the product rule and chain rule, marginalization and normalization, are *general* techniques for probabilistic reasoning, and their usefulness is not limited to the naive Bayes models.
- More generally:
- **Theorem 2.23.** *Let* $Q, E_1, \ldots, E_{n_E}, U_1, \ldots, U_{n_U}$ *be random variables and* $D := \mathrm{dom}(U_1) \times \ldots \times \mathrm{dom}(U_{n_U})$. *Then*

  $$\mathbb{P}(Q|E_1 = e_1, \ldots, E_{n_E} = e_{n_e}) = \alpha(\sum_{u \in D} \mathbb{P}(Q, E_1 = e_1, \ldots, E_{n_E} = e_{n_e}, U_1 = u_1, \ldots, U$$

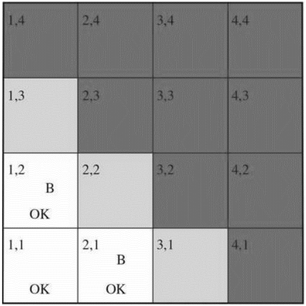
  .

  We call Q *the* ***query variable****,* $E_1, \ldots, E_{n_E}$ *the* ***evidence****, and* $U_1, \ldots, U_{n_U}$ *the* ***unknown*** *(or* ***hidden****)* ***variables****, and computing a conditional probability this way* ***enumeration****.*
- Note that this is just a "mathy" way of saying we
  1. sum over all relevant entries of the full joint probability distribution of the variables, and
  2. normalize the result to yield a probability distribution.

SOME RIGHTS RESERVED

21.2.4 Example – The Wumpus is Back

Example: The Wumpus is Back



- We have a maze where
  - Every cell except $[1,1]$ possibly contains a *pit*, with 20% probability.
  - pits cause a *breeze* in neighboring cells (we forget the wumpus and the gold for now)
- Where should the agent go, if there is a breeze at $[1,2]$ and $[2,1]$?
- Pure logical inference can conclude nothing about which square is *most likely* to be safe!

We can model this using the Boolean random variables:

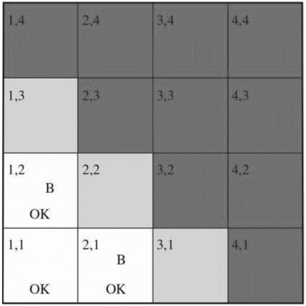
- $P_{i,j}$ for $i,j \in \{1,2,3,4\}$, stating there is a pit at square $[i,j]$, and
- $B_{i,j}$ for $(i,j) \in \{(1,1),(1,2),(2,1)\}$, stating there is a breeze at square $[i,j]$

$\Rightarrow$ let's apply our machinery!

SOME RIGHTS RESERVED

Wumpus: Probabilistic Model

- **First:** Let's try to compute the full joint probability distribution $\mathbb{P}(P_{1,1}, \ldots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$.
  1. By the product rule, this is equal to $\mathbb{P}(B_{1,1}, B_{1,2}, B_{2,1} | P_{1,1}, \ldots, P_{4,4}) \cdot \mathbb{P}(P_{1,1}, \ldots, P_{4,4})$.
  2. Note that $\mathbb{P}(B_{1,1}, B_{1,2}, B_{2,1} | P_{1,1}, \ldots, P_{4,4})$ is either 1 (if all the $B_{i,j}$ are consistent with the positions of the pits $P_{k,l}$) or 0 (otherwise).
  3. Since the pits are spread independently, we have $\mathbb{P}(P_{1,1}, \ldots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} \mathbb{P}(P_{i,j})$

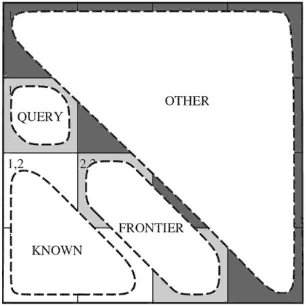


- $\leadsto$ We know all of these probabilities.
- $\leadsto$ We can now use enumeration to compute $\mathbb{P}(P_{i,j} | < known >) = \alpha(\sum_{< unknowns >} \mathbb{P}(P_{i,j}, < known >, < unknowns >))$

SOME RIGHTS RESERVED

Wumpus Continued

- **Problem:** We only know $P_{i,j}$ for three fields. If we want to compute e.g. $P_{1,3}$ via enumeration, that leaves $2^{4^2-4} = 4096$ terms to sum over!
- **Let's do better.**



- Let $b := \neg B_{1,1} \wedge B_{1,2} \wedge B_{2,1}$ (All the breezes we know about)
- Let $p := \neg P_{1,1} \wedge \neg P_{1,2} \wedge \neg P_{2,1}$. (All the pits we know about)
- Let $F := \{P_{3,1} \wedge P_{2,2}, \neg P_{3,1} \wedge P_{2,2}, P_{3,1} \wedge \neg P_{2,2}, \neg P_{3,1} \wedge P_{2,2}\}$ (the current *"frontier"*)
- Let O be (the set of assignments for) all the other variables $P_{i,j}$. (i.e. except p, F and our query $P_{1,3}$)

Then the observed breezes b are conditionally independent of O given p and F. (Whether there is a pit anywhere else does not influence the breezes we observe.)

- $\Rightarrow P(b \mid P_{1,3},p,O,F) = P(b \mid P_{1,3},p,F)$. Let's exploit this!

SOME RIGHTS RESERVED

Optimized Wumpus

- In particular:

$$\mathbb{P}(P_{1,3}|p,b) {=} \alpha(\sum_{o\in O,f\in F} \mathbb{P}(P_{1,3},b,p,f,o)) {=} \alpha(\sum_{o\in O,f\in F} P(b \mid P_{1,3},p,o,f)\cdot\mathbb{P}(P_{1,3},p$$

$$= \alpha(\sum_{f\in F}\sum_{o\in O} P(b \mid P_{1,3},p,f)\cdot\mathbb{P}(P_{1,3},p,f,o)) {=} \alpha(\sum_{f\in F} P(b \mid P_{1,3},p,f)\cdot($$

$$= \alpha(\sum_{f\in F} P(b \mid P_{1,3},p,f)\cdot(\sum_{o\in O}\mathbb{P}(P_{1,3})\cdot P(p)\cdot P(f)\cdot P(o)))$$

$$= \alpha(\mathbb{P}(P_{1,3})\cdot P(p)\cdot(\sum_{f\in F}\underbrace{P(b \mid P_{1,3},p,f)}_{\in\{0,1\}}\cdot P(f)\cdot\underbrace{(\sum_{o\in O}P(o)))}_{=1})$$

⇝ this is just a sum over the frontier, i.e. 4 terms ☺

- So: $\mathbb{P}(P_{1,3}|p,b) = \alpha(\langle 0.2\cdot(0.8)^3\cdot(1\cdot 0.04+1\cdot 0.16+1\cdot 0.16+0), 0.8\cdot(0.8)^3\cdot(1\cdot 0.04+1\cdot 0.16+0+0)\rangle) \approx \langle 0.31, 0.69\rangle$
- Analogously: $\mathbb{P}(P_{3,1}|p,b) = \langle 0.31, 0.69\rangle$ and $\mathbb{P}(P_{2,2}|p,b) = \langle 0.86, 0.14\rangle$ (⇒ avoid $[2,2]$!)

SOME RIGHTS RESERVED

Cooking Recipe

- In general, when you want to reason probabilistically, a good heuristic is:
  1. Try to frame the full joint probability distribution in terms of the probabilities you know. Exploit product rule/chain rule, independence, conditional independence, marginalization **and domain knowledge** (as e.g. $\mathbb{P}(b|p,f)\in\{0,1\}$)
     $\leadsto$ the problem can be solved at all!
  2. **Simplify**: Start with the equation for enumeration:
     $$\mathbb{P}(Q|E_1,\ldots)=\alpha(\sum_{u\in U}\mathbb{P}(Q,E_1,\ldots,U_1=u_1,\ldots))$$
  3. Substitute by the result of 1., and again, exploit all of our machinery
  4. Implement the resulting (system of) equation(s)
  5. ???
  6. Profit

SOME RIGHTS RESERVED

Summary

- Probability distributions and conditional probability distributions allow us to represent random variables as convenient datastructures in an implementation (Assuming they are finite domain...)
- The full joint probability distribution allows us to compute all probabilities of statements about the random variables contained (But possibly inefficient)
- Marginalization and normalization are the specific techniques for extracting the *specific* probabilities we are interested in from the full joint probability distribution.
- The product and chain rule, exploiting (conditional) independence, Bayes' Theorem, and of course *domain specific* knowledge allow us to do so much more efficiently.
- Naive Bayes models are one example where all these techniques come together.

SOME RIGHTS RESERVED

Chapter 22
Probabilistic Reasoning: Bayesian Networks

22.1 Introduction

SOME RIGHTS RESERVED

John, Mary, and My Brand-New Alarm

- **Example 1.1 (From Russell/Norvig).**
  - I got very valuable stuff at home. So I bought an alarm. Unfortunately, the alarm just rings at home, doesn't call me on my mobile.
  - I've got two neighbors, Mary and John, who'll call me if they hear the alarm.
  - The problem is that, sometimes, the alarm is caused by an earthquake.
  - Also, John might confuse the alarm with his telephone, and Mary might miss the alarm altogether because she typically listens to loud music.

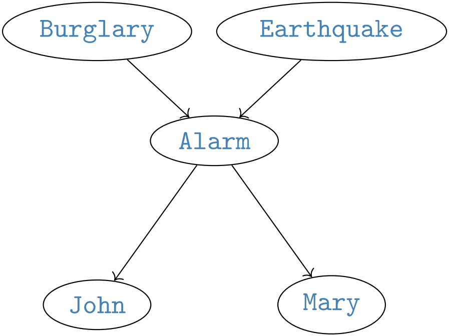
  ⇝ Random variables: `Burglary`, `Earthquake`, `Alarm`, `John`, `Mary`. Given that both John and Mary call me, what is the probability of a burglary?
- ⇝ This is *almost* a naive Bayes model, but with multiple causes (`Burglary` and `Earthquake`) for the `Alarm`, which in turn may cause `John` and/or `Mary`.

SOME RIGHTS RESERVED

John, Mary, and My Alarm: Assumptions

We assume:

- We (should) know $\mathbb{P}(\texttt{Alarm}|\texttt{Burglary},\texttt{Earthquake})$, $\mathbb{P}(\texttt{John}|\texttt{Alarm})$, and $\mathbb{P}(\texttt{Mary}|\texttt{Alarm})$.
- `Burglary` and `Earthquake` are independent.
- `John` and `Mary` are conditionally independent given `Alarm`.
- Moreover: Both `John` and `Mary` are conditionally independent of *any other random variables* in the graph given `Alarm`. (Only `Alarm` causes them, and everything else only causes them indirectly *through* `Alarm`)



- **First Step:** Construct the full joint probability distribution,
- **Second Step:** Use **enumeration** to compute $\mathbb{P}(\texttt{Burglary}|\texttt{John}=\mathsf{T},\texttt{Mary}=\mathsf{T})$.

SOME RIGHTS RESERVED

John, Mary, and My Alarm: The Distribution

-

$$\begin{aligned}
&\mathbb{P}(\mathtt{John},\mathtt{Mary},\mathtt{Alarm},\mathtt{Burglary},\mathtt{Earthquake})\\
=&\mathbb{P}(\mathtt{John}|\mathtt{Mary},\mathtt{Alarm},\mathtt{Burglary},\mathtt{Earthquake})\cdot\mathbb{P}(\mathtt{Mary}|\mathtt{Alarm},\mathtt{Burglary},\mathtt{Eartho}\\
&\cdot\mathbb{P}(\mathtt{Alarm}|\mathtt{Burglary},\mathtt{Earthquake})\cdot\mathbb{P}(\mathtt{Burglary}|\mathtt{Earthquake})\cdot\mathbb{P}(\mathtt{Earthquake}\\
=&\mathbb{P}(\mathtt{John}|\mathtt{Alarm})\cdot\mathbb{P}(\mathtt{Mary}|\mathtt{Alarm})\cdot\mathbb{P}(\mathtt{Alarm}|\mathtt{Burglary},\mathtt{Earthquake})\cdot\mathbb{P}(\mathtt{Burglar}
\end{aligned}$$

- We plug into the equation for enumeration:

$$\mathbb{P}(\mathtt{Burglary}|\mathtt{John}=\mathsf{T},\mathtt{Mary}=\mathsf{T})=\alpha(\mathbb{P}(\mathtt{Burglary})\sum_{a\in\{\mathsf{T},\mathsf{F}\}}P(\mathtt{John}\mid\mathtt{Alarm}=a)\cdot$$

$$\cdot\sum_{q\in\{\mathsf{T},\mathsf{F}\}}\mathbb{P}(\mathtt{Alarm}=a|\mathtt{Burglary},\mathtt{Earthquake}=q)P(\mathtt{Earthquake}=q))$$

- ⇝ Now let's scale things up to arbitrarily many variables!

SOME RIGHTS RESERVED

Bayesian Networks: Definition

- **Definition 1.2.** A Bayesian network consists of
  1. a directed acyclic graph $\langle\mathcal{X},E\rangle$ of random variables $\mathcal{X}=\{X_1,\ldots,X_n\}$, and
  2. a conditional probability distribution $\mathbb{P}(X_i|\mathrm{Parents}(X_i))$ for every $X_i\in\mathcal{X}$ (also called the CPT for conditional probability table)

  such that every X_i is conditionally independent of any conjunctions of non-descendents of X_i given $\mathrm{Parents}(X_i)$.
- **Definition 1.3.** Let $\langle\mathcal{X},E\rangle$ be a directed acyclic graph, $X\in\mathcal{X}$, and E^* the reflexive transitive closure of E. The non-descendents of X are the elements of the set $\mathrm{NonDesc}(X):=\{Y\,|\,(X,Y)\notin E^*\}\backslash\mathrm{Parents}(X)$.
- Note that the roots of the graph are conditionally independent given the empty set; i.e. they are independent.
- **Theorem 1.4.** *The full joint probability distribution of a Bayesian network $\langle\mathcal{X},E\rangle$ is given by*

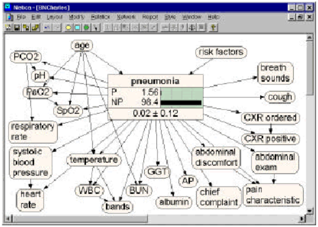
$$\mathbb{P}(X_1,\ldots,X_n)=\prod_{X_i\in\mathcal{X}}\mathbb{P}(X_i|\mathrm{Parents}(X_i))$$

SOME RIGHTS RESERVED

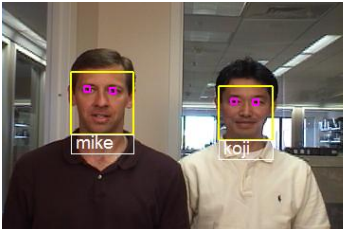
Some Applications

- A ubiquitous problem: Observe “symptoms”, need to infer “causes”.

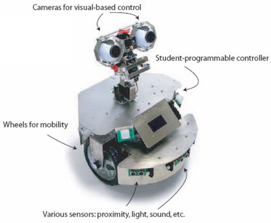
Medical Diagnosis



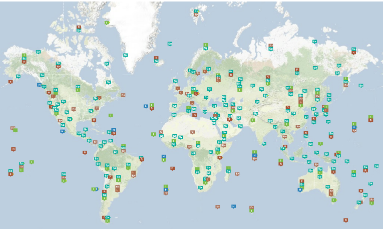
Face Recognition



Self-Localization


Cameras for visual-based control
Student-programmable controller
Wheels for mobility
Various sensors: proximity, light, sound, etc.

Nuclear Test Ban



22.2 Constructing Bayesian Networks

SOME RIGHTS RESERVED

Compactness of Bayesian Networks

- **Definition 2.1.** Given random variables $X_1, \ldots, X_n$ with finite domains $D_1, \ldots, D_n$, the size of $\mathcal{B} := \langle \{X_1, \ldots, X_n\}, E\rangle$ is defined as

$$\mathrm{size}(\mathcal{B}) := \sum_{i=1}^{n} |D_i| \cdot \left(\prod_{X_j \in \mathrm{Parents}(X_i)} |D_j|\right)$$

- **Note:** $\mathrm{size}(\mathcal{B}) \mathrel{\widehat{=}}$ The total number of entries in the conditional probability distributions.

SOME RIGHTS RESERVED

Compactness of Bayesian Networks

- **Definition 2.5.** Given random variables $X_1, \ldots, X_n$ with finite domains $D_1, \ldots, D_n$, the size of $\mathcal{B} := \langle \{X_1, \ldots, X_n\}, E \rangle$ is defined as

$$\mathrm{size}(\mathcal{B}) := \sum_{i=1}^{n} |D_i| \cdot \left(\prod_{X_j \in \mathrm{Parents}(X_i)} |D_j| \right)$$

- **Note:** $\mathrm{size}(\mathcal{B}) \mathrel{\widehat{=}}$ The total number of entries in the conditional probability distributions.
- **Note:** Smaller BN $\rightsquigarrow$ need to assess less probabilities, more efficient inference.
- **Observation 2.6.** *Explicit full joint probability distribution has size* $\prod_{i=1}^{n} |D_i|$.
- **Observation 2.7.** *If* $|\mathrm{Parents}(X_i)| \leq k$ *for every* X_i*, and* $D_{\max}$ *is the largest random variable domain, then* $\mathrm{size}(\mathcal{B}) \leq n|D_{\max}|^{k+1}$.

SOME RIGHTS RESERVED

Compactness of Bayesian Networks

- **Definition 2.9.** Given random variables $X_1, \ldots, X_n$ with finite domains $D_1, \ldots, D_n$, the size of $\mathcal{B} := \langle\{X_1, \ldots, X_n\}, E\rangle$ is defined as

$$\mathrm{size}(\mathcal{B}) := \sum_{i=1}^{n} |D_i| \cdot (\prod_{X_j \in \mathrm{Parents}(X_i)} |D_j|)$$

- **Note:** $\mathrm{size}(\mathcal{B}) \,\widehat{=}$ The total number of entries in the conditional probability distributions.
- **Note:** Smaller BN $\leadsto$ need to assess less probabilities, more efficient inference.
- **Observation 2.10.** *Explicit full joint probability distribution has size* $\prod_{i=1}^{n} |D_i|$.
- **Observation 2.11.** *If* $|\mathrm{Parents}(X_i)| \leq k$ *for every* X_i, *and* $D_{\max}$ *is the largest random variable domain, then* $\mathrm{size}(\mathcal{B}) \leq n|D_{\max}|^{k+1}$.
- **Example 2.12.** For $|D_{\max}| = 2$, $n = 20$, $k = 4$ we have $2^{20} = 1048576$ probabilities, but a Bayesian network of size $\leq 20 \cdot 2^5 = 640 \ldots$!
- In the *worst case*, $\mathrm{size}(\mathcal{B}) = n \cdot (\prod_{\cdot=i}^{1} n)|D_i|$, namely if every variable depends on all its predecessors in the chosen variable ordering.
- **Intuition:** BNs are compact – i.e. of small size – if each variable is directly influenced only by few of its predecessor variables.

SOME RIGHTS RESERVED

Keeping Networks Small

- To keep our Bayesian networks small, we can:
  1. **Reduce the number of edges:** ⇒ Order the variables to allow for exploiting conditional independence (causes before effects), or
  2. **represent the conditional probability distributions efficiently:**
     2.1 For Boolean random variables X, we only need to store $\mathrm{P}(X = \mathsf{T}|\mathrm{Parents}(X))$ ($\mathrm{P}(X = \mathsf{F}|\mathrm{Parents}(X)) = 1 - \mathrm{P}(X = \mathsf{T}|\mathrm{Parents}(X))$) (Cuts the number of entries in half!)
     2.2 Introduce different **kinds** of nodes exploiting domain knowledge; e.g. deterministic and noisy disjunction nodes.

SOME RIGHTS RESERVED

Reducing Edges: Variable Order Matters

- Given a set of random variables $X_1,\ldots,X_n$, consider the following (impractical, but illustrative) pseudo-algorithm for constructing a Bayesian network:
- **Definition 2.13 (BN construction algorithm).**

```
1. Initialize $BN := \langle\{X_1,\ldots,X_n\},E\rangle$ where $E = \emptyset$.
2. Fix any variable ordering, $X_1,\ldots,X_n$.
3. for $i := 1,\ldots,n$ do
    a. Choose a minimal set $\mathrm{Parents}(X_i) \subseteq \{X_1,\ldots,X_{i-1}\}$ such that
           $\mathbb{P}(X_i|X_{i-1},\ldots,X_1) = \mathbb{P}(X_i|\mathrm{Parents}(X_i))$
    b. For each $X_j \in \mathrm{Parents}(X_i)$, insert $(X_j,X_i)$ into $E$.
    c. Associate $X_i$ with $\mathbb{P}(X_i|\mathrm{Parents}(X_i))$.
```

- **Attention:** *Which* variables we need to include into $\mathrm{Parents}(X_i)$ depends on what "$\{X_1,\ldots,X_{i-1}\}$" is ... !
- **Thus:** The size of the resulting BN depends on the chosen variable ordering $X_1,\ldots,X_n$.
- **In Particular:** The size of a Bayesian network is *not* a fixed property of the domain. It depends on the skill of the designer.

SOME RIGHTS RESERVED

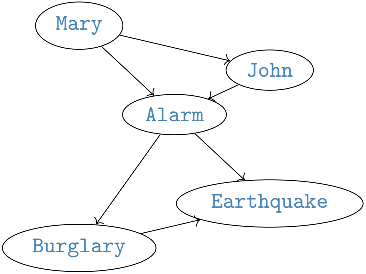
John and Mary Depend on the Variable Order!

- **Example 2.14.** `Mary`, `John`, `Alarm`, `Burglary`, `Earthquake`.

SOME RIGHTS RESERVED

John and Mary Depend on the Variable Order!

- **Example 2.15.** `Mary`, `John`, `Alarm`, `Burglary`, `Earthquake`.



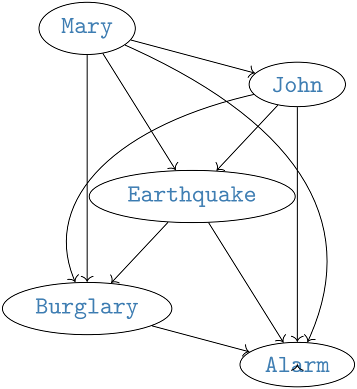
John and Mary Depend on the Variable Order! Ctd.

- **Example 2.16.** `Mary`, `John`, `Earthquake`, `Burglary`, `Alarm`.

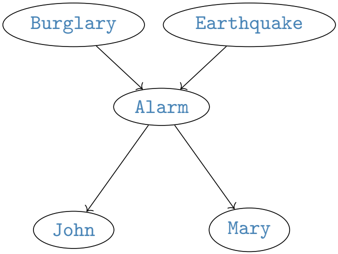
SOME RIGHTS RESERVED

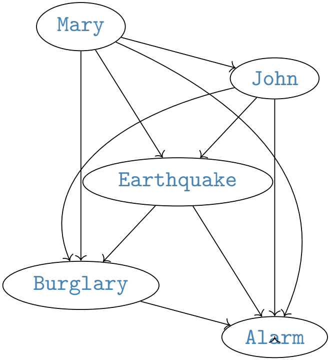
John and Mary Depend on the Variable Order! Ctd.

- **Example 2.17.** `Mary`, `John`, `Earthquake`, `Burglary`, `Alarm`.

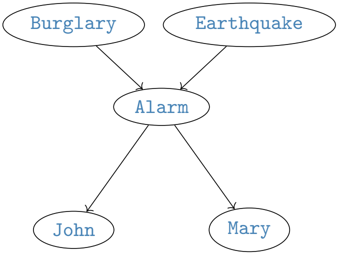


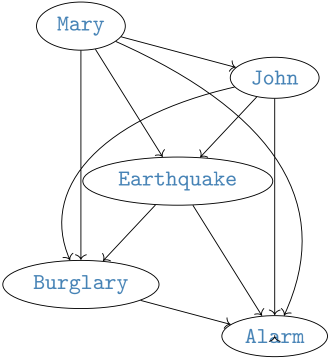
John and Mary, What Went Wrong?





John and Mary, What Went Wrong?

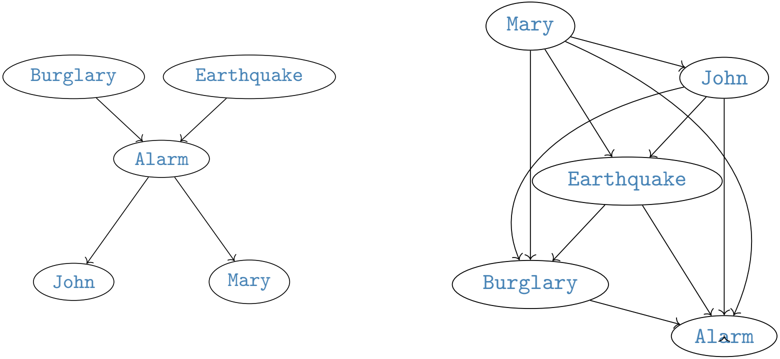




- **Intuition:** These BNs link from *effects* to their *causes*!
  ⇒ Even though Mary and John are conditionally independent given Alarm, this is not exploited, since Alarm is not ordered before Mary and John!

SOME RIGHTS RESERVED

John and Mary, What Went Wrong?



- **Intuition:** These BNs link from *effects* to their *causes*!
  ⇒ Even though `Mary` and `John` are conditionally independent given `Alarm`, this is not exploited, since `Alarm` is not ordered before `Mary` and `John`!

⇒ **Rule of Thumb:** We should order causes before symptoms.

SOME RIGHTS RESERVED

Representing Conditional Distributions: Deterministic Nodes

- **Definition 2.18.** A node X in a Bayesian network is called deterministic, if its value is completely determined by the values of $\mathrm{Parents}(X)$.

Representing Conditional Distributions: Deterministic Nodes

- **Definition 2.21.** A node X in a Bayesian network is called deterministic, if its value is completely determined by the values of $\mathrm{Parents}(X)$.
- **Example 2.22.** The *sum of two dice throws* S is entirely determined by the values of the two dice *First* and *Second*.

SOME RIGHTS RESERVED

Representing Conditional Distributions: Deterministic Nodes

- **Definition 2.24.** A node X in a Bayesian network is called deterministic, if its value is completely determined by the values of $\mathrm{Parents}(X)$.
- **Example 2.25.** The *sum of two dice throws* S is entirely determined by the values of the two dice *First* and *Second*.
- **Example 2.26.** In the *Wumpus* example, the *breezes* are entirely determined by the *pits*

SOME RIGHTS RESERVED

Representing Conditional Distributions: Deterministic Nodes

- **Definition 2.27.** A node X in a Bayesian network is called deterministic, if its value is completely determined by the values of $\mathrm{Parents}(X)$.
- **Example 2.28.** The *sum of two dice throws* S is entirely determined by the values of the two dice *First* and *Second*.
- **Example 2.29.** In the *Wumpus* example, the *breezes* are entirely determined by the *pits*
- $\leadsto$ *Deterministic* nodes model direct, *causal* relationships.
- $\leadsto$ If X is deterministic, then $\mu(X \mid \mathrm{Parents}(X)) \in \{0,1\}$

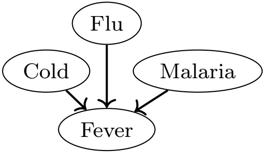
Representing Conditional Distributions: Deterministic Nodes

- **Definition 2.30.** A node X in a Bayesian network is called deterministic, if its value is completely determined by the values of $\mathrm{Parents}(X)$.
- **Example 2.31.** The *sum of two dice throws* S is entirely determined by the values of the two dice *First* and *Second*.
- **Example 2.32.** In the *Wumpus* example, the *breezes* are entirely determined by the *pits*
- $\rightsquigarrow$ *Deterministic* nodes model direct, *causal* relationships.
- $\rightsquigarrow$ If X is deterministic, then $\mu(X \mid \mathrm{Parents}(X)) \in \{0,1\}$
- $\rightsquigarrow$ we can replace the conditional probability distribution $\mathbb{P}(X|\mathrm{Parents}(X))$ by a boolean function.

SOME RIGHTS RESERVED

Representing Conditional Distributions: Noisy Nodes

- Sometimes, values of nodes are "almost deterministic":
- **Example 2.33 (Inhibited Causal Dependencies).** Assume the network on the right contains *all* possible causes of fever. (Or add a dummy-node for "other causes") *If* there is a fever, then *one* of them (at least) must be the cause, but none of them *necessarily* cause a fever: The causal relation between parent and child is inhibited.

  $\rightsquigarrow$ We can model the inhibitions by individual inhibition factors q_d.
- **Definition 2.34.** The conditional probability distribution of a noisy disjunction node X with $\mathrm{Parents}(X) = X_1, \ldots, X_n$ in a Bayesian network is given by $P(X \mid X_1, \ldots, X_n) = 1 - (\prod_{\{j \mid X_j = \mathsf{T}\}} q_j)$, where the q_i are the inhibition factors of $X_i \in \mathrm{Parents}(X)$, defined as $q_i := P(\neg X \mid \neg X_1, \ldots, \neg X_{i-1}, X_i, \neg X_{i+1}, \ldots, \neg X_n)$
- $\rightsquigarrow$ Instead of a distribution with 2^k parameters, we only need k parameters!

Representing Conditional Distributions: Noisy Nodes

- **Example 2.35.** Assume the following inhibition factors for **??**:

$$\begin{aligned} q_{\mathrm{cold}} &= P(\neg\mathrm{fever} \mid \mathrm{cold}, \neg\mathrm{flu}, \neg\mathrm{malaria}) = 0.6 \\ q_{\mathrm{flu}} &= P(\neg\mathrm{fever} \mid \neg\mathrm{cold}, \mathrm{flu}, \neg\mathrm{malaria}) = 0.2 \\ q_{\mathrm{malaria}} &= P(\neg\mathrm{fever} \mid \neg\mathrm{cold}, \neg\mathrm{flu}, \mathrm{malaria}) = 0.1 \end{aligned}$$

  If we model Fever as a noisy disjunction node, then the general rule $P(X_i \mid \mathrm{Parents}(X_i)) = \prod_{\{j \mid X_j = \mathsf{T}\}} q_j$ for the CPT gives the following table:

Cold	Flu	Malaria	$P(\mathrm{Fever})$	$P(\neg\mathrm{Fever})$
F	F	F	0.0	1.0
F	F	T	0.9	0.1
F	T	F	0.8	0.2
F	T	T	0.98	$0.02 = 0.2 \cdot 0.1$
T	F	F	0.4	0.6
T	F	T	0.94	$0.06 = 0.6 \cdot 0.1$
T	T	F	0.88	$0.12 = 0.6 \cdot 0.2$
T	T	T	0.988	$0.012 = 0.6 \cdot 0.2 \cdot 0.1$

SOME RIGHTS RESERVED

Representing Conditional Distributions: Summary

- Note that deterministic nodes and noisy disjunction nodes are just two examples of "specialized" kinds of nodes in a Bayesian network.
- In general, noisy logical relationships in which a variable depends on k parents can be described by $\mathcal{O}(k)$ parameters instead of $\mathcal{O}(2^k)$ for the full conditional probability table. This can make assessment (and learning) tractable.
- **Example 2.36.** The CPCS network [**PraProMid:kelbn94**] uses noisy-OR and noisy-MAX distributions to model relationships among diseases and symptoms in internal medicine. With 448 nodes and 906 links, it requires only 8,254 values instead of 133,931,430 for a network with full conditional probability distributions.

22.3 Inference in Bayesian Networks

SOME RIGHTS RESERVED

Probabilistic Inference Tasks in Bayesian Networks

- Remember:
- **Definition 3.1 (Probabilistic Inference Task).** Let $X_1, \ldots, X_n = Q_1, \ldots, Q_{n_Q}, E_1, \ldots, E_{n_E}, U_1, \ldots, U_{n_U}$ be a set of random variables, a probabilistic inference task.
  We wish to compute the conditional probability distribution $\mathbb{P}(Q_1, \ldots, Q_{n_Q} | E_1 = e_1, \ldots, E_{n_E} = e_{n_E})$.
  We call
  - a $Q_1, \ldots, Q_{n_Q}$ the query variables,
  - a $E_1, \ldots, E_{n_E}$ the evidence variables, and
  - $U_1, \ldots, U_{n_U}$ the hidden variables.
- We know the full joint probability distribution: $\mathbb{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbb{P}(X_i | \mathrm{Parents}(X_i))$
- And we know about enumeration:

$$\mathbb{P}(Q_1, \ldots, Q_{n_Q} | E_1 = e_1, \ldots, E_{n_E} = e_{n_E}) =$$

$$\alpha(\sum_{u \in D_U} \mathbb{P}(Q_1, \ldots, Q_{n_Q}, E_1 = e_1, \ldots, E_{n_E} = e_{n_E}, U_1 = u_1, \ldots, U_{n_U} = u_{n_U}))$$

(where $D_U = \mathrm{dom}(U_1) \times \ldots \times \mathrm{dom}(U_{n_U})$)

SOME RIGHTS RESERVED

Enumeration: The Alarm-Example

- Remember our example: $\mathbb{P}(\mathtt{Burglary}|\mathtt{John},\mathtt{Mary})$ (hidden variables: $\mathtt{Alarm}, \mathtt{Earthquake}$)

  $=\alpha(\sum_{b_a,b_e\in\{\mathsf{T},\mathsf{F}\}} P(\mathtt{John},\mathtt{Mary},\mathtt{Alarm}=b_a,\mathtt{Earthquake}=b_e,\mathtt{Burglary}))$
  $=\alpha(\sum_{b_a,b_e\in\{\mathsf{T},\mathsf{F}\}} P(\mathtt{John}\mid\mathtt{Alarm}=b_a)\cdot P(\mathtt{Mary}\mid\mathtt{Alarm}=b_a)$
  $\cdot\mathbb{P}(\mathtt{Alarm}=b_a|\mathtt{Earthquake}=b_e,\mathtt{Burglary})\cdot P(\mathtt{Earthquake}=b_e)\cdot\mathbb{P}(\mathtt{Burglary}))$
- $\leadsto$ These are 5 factors in 4 summands ($b_a, b_e\in\{\mathsf{T},\mathsf{F}\}$) over two cases ($\mathtt{Burglary}\in\{\mathsf{T},\mathsf{F}\}$),
- $\leadsto$ 38 arithmetic operations ($+3$ for α)
- **General worst case:** $\mathcal{O}(n2^n)$
- **Let's do better!**

SOME RIGHTS RESERVED

Enumeration: First Improvement

- **Some abbreviations:** $j := \texttt{John}, m := \texttt{Mary}, a := \texttt{Alarm}, e := \texttt{Earthquake}, b := \texttt{Burglary},$
-

$$\mathbb{P}(b|j,m) = \alpha(\sum_{b_a,b_e\in\{\mathsf{T},\mathsf{F}\}} P(j \mid a = b_a)\cdot P(m \mid a = b_a)\cdot \mathbb{P}(a = b_a|e = b_e, b)\cdot P(e$$

- Let's "optimize":

$$\mathbb{P}(b|j,m) = \alpha(\mathbb{P}(b)\cdot(\sum_{b_e\in\{\mathsf{T},\mathsf{F}\}} P(e = b_e)\cdot(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a = b_a|e = b_e, b)\cdot P(j \mid a =$$

⇝ 3 factors in 2 summand + 2 factors in 2 summands + two factors in the outer product, over two cases = 28 arithmetic operations (+3 for α)

Second Improvement: Variable Elimination 1

- Consider $\mathbb{P}(j|b = \mathsf{T})$.
- Using enumeration:

$$=\alpha(P(b)\cdot(\sum_{b_e\in\{\mathsf{T},\mathsf{F}\}} P(e = b_e)\cdot(\sum_{a_e\in\{\mathsf{T},\mathsf{F}\}} P(a = a_e \mid e = b_e, b)\cdot\mathbb{P}(j|a = a_e)\cdot(\sum_{a_m\in\{\mathsf{T}$$

$\leadsto$ $\mathbb{P}(\mathtt{John}|\mathtt{Burglary} = \mathsf{T})$ does not depend on `Mary` (duh...)

- **More generally**:
- **Lemma 3.2.** *Given a query $\mathbb{P}(Q_1, \ldots, Q_{n_Q}|E_1 = e_1, \ldots, E_{n_E} = e_{n_E})$, we can ignore (and remove) all hidden leaves of the Bayesian network.*
- ...doing so yields new leaves, which we can then ignore again, etc., until:
- **Lemma 3.3.** *Given a query $\mathbb{P}(Q_1, \ldots, Q_{n_Q}|E_1 = e_1, \ldots, E_{n_E} = e_{n_E})$, we can ignore (and remove) all hidden variables that are not ancestors of any of the $Q_1, \ldots, Q_{n_Q}$ or $E_1, \ldots, E_{n_E}$.*

SOME RIGHTS RESERVED

Enumeration: First Algorithm

- Assume the $X_1, \ldots, X_n$ are topologically sorted (causes before effects)

```
function ENUMERATE-QUERY($Q$,$\langle E_1 = e_1, \ldots, E_{n_E} = e_{n_E} \rangle$)
    $P := \langle\rangle$                                  /* $= \mathbb{P}(Q|E_i = e_i)$ */
    $X_1, \ldots, X_n :=$ variables filtered according to ??, topologically sorted
    for all $q \in \mathrm{dom}(Q)$ do
        $P_i := $ENUMALL($\langle X_1, \ldots, X_n \rangle$,$\langle E_1 = e_1, \ldots, E_{n_E} = e_{n_E}, Q = q \rangle$)
    return $\alpha(P)$

function ENUMALL($\langle Y_1, \ldots, Y_{n_Y} \rangle$,$\langle A_1 = a_1, \ldots, A_{n_A} = a_{n_A} \rangle$)
                          /* By construction, $\mathrm{Parents}(Y_1) \subset \{A_1, \ldots, A_{n_A}\}$ */
    if $n_y = 0$ then return 1.0
    else if $Y_1 = A_j$ then return $P(A_j = a_j \mid \mathrm{Parents}(A_j))\cdot$ENUMALL($\langle Y_2, \ldots, Y_{n_Y} \rangle$,$\langle A_1 = a_1, \ldots, A_{n_A} = a_{n_A} \rangle$)
    else return $\sum_{y \in \mathrm{dom}(Y_1)} P(Y_1 = y \mid \mathrm{Parents}(Y_1))\cdot$ENUMALL($\langle Y_2, \ldots, Y_{n_Y} \rangle$,$\langle A_1 = a_1, \ldots, A_{n_A} = a_{n_A}, Y_1 = y \rangle$)
```

- **General worst case Complexity:** $\mathcal{O}(2^n)$ – better, but still not great

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m

  $\textsc{Enumerate-Query}(b, \langle j = \mathsf{T}, m = \mathsf{T}\rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) =$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m

  ENUMERATE-QUERY$(b, \langle j = \mathsf{T}, m = \mathsf{T} \rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) =$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := \textsc{EnumAll}(\langle b, e, a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}\rangle)$
  - $P_1 := \textsc{EnumAll}(\langle b, e, a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}\rangle)$

  $\Leftarrow \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha()$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := \underbrace{\textsc{EnumAll}(\langle b, e, a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}\rangle)}_{=P(b)\cdot\textsc{EnumAll}(\langle e,a,j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T}\rangle)}$
  - $P_1 := \underbrace{\textsc{EnumAll}(\langle b, e, a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}\rangle)}_{=P(\neg b)\cdot\textsc{EnumAll}(\langle e,a,j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F}\rangle)}$

  $\Leftarrow\ \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b)\cdot)$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b) \cdot \textsc{EnumAll}(\langle e, a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}\rangle)$
  - $P_1 := P(\neg b) \cdot \textsc{EnumAll}(\langle e, a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}\rangle)$

  $\Leftarrow \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b)\cdot)$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b) \cdot \underbrace{\textsc{EnumAll}(\langle e, a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}\rangle)}_{=(\sum_{b_e \in \{\mathsf{T},\mathsf{F}\}} P(e=b_e)\cdot\textsc{EnumAll}(\langle a,j,m\rangle\cdot\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=b_e\rangle))}$
  - $P_1 := P(\neg b) \cdot \underbrace{\textsc{EnumAll}(\langle e, a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}\rangle)}_{=(\sum_{b_e \in \{\mathsf{T},\mathsf{F}\}} P(e=b_e)\cdot\textsc{EnumAll}(\langle a,j,m\rangle\cdot\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=b_e\rangle))}$

  $\Leftarrow \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b) \cdot (\sum_{b_e \in \{\mathsf{T},\mathsf{F}\}} P(e = b_e)\cdot))$$

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b) \cdot \left[+ \begin{array}{l} P(e) \cdot \textsc{EnumAll}(\langle a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \mathsf{T}\rangle) \\ P(\neg e) \cdot \textsc{EnumAll}(\langle a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \mathsf{F}\rangle) \end{array}\right.$
  - $P_1 := P(\neg b) \cdot \left[+ \begin{array}{l} P(e) \cdot \textsc{EnumAll}(\langle a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \mathsf{T}\rangle) \\ P(\neg e) \cdot \textsc{EnumAll}(\langle a, j, m\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \mathsf{F}\rangle) \end{array}\right.$
  - $\Leftarrow \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b) \cdot (\sum_{b_e \in \{\mathsf{T},\mathsf{F}\}} P(e = b_e) \cdot))$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 :=$
$$P(b)\cdot\left[\begin{array}{l} P(e)\cdot \underbrace{\textsc{EnumAll}(\langle a,j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=\mathsf{T}\rangle)}_{=(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} P(a=b_a\ |\ b,e)\cdot\textsc{EnumAll}(\langle j,m\rangle\cdot\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=\mathsf{T},a=b_a\rangle))} \\ +\ P(\neg e)\cdot \underbrace{\textsc{EnumAll}(\langle a,j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=\mathsf{F}\rangle)}_{=(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} P(a=b_a\ |\ b,\neg e)\cdot\textsc{EnumAll}(\langle j,m\rangle\cdot\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=\mathsf{F},a=b_a\rangle))} \end{array}\right.$$
  - $P_1 :=$
$$P(\neg b)\cdot\left[\begin{array}{l} P(e)\cdot \underbrace{\textsc{EnumAll}(\langle a,j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=\mathsf{T}\rangle)}_{=(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} P(a=b_a\ |\ \neg b,e)\cdot\textsc{EnumAll}(\langle j,m\rangle\cdot\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=\mathsf{T},a=b_a\rangle))} \\ +\ P(\neg e)\cdot \underbrace{\textsc{EnumAll}(\langle a,j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=\mathsf{F}\rangle)}_{=(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} P(a=b_a\ |\ \neg b,\neg e)\cdot\textsc{EnumAll}(\langle j,m\rangle\cdot\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=\mathsf{F},a=b_a\rangle))} \end{array}\right.$$

$\Leftarrow\ \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j=\mathsf{T},m=\mathsf{T}) = \alpha(\mathbb{P}(b)\cdot(\sum_{b_e\in\{\mathsf{T},\mathsf{F}\}} P(e=b_e)\cdot(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a=b_a|e=b_e,b)\cdot\cdot)))$$

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b) \cdot$
    $$\left[+\begin{array}{l} P(e)\cdot\left[+\begin{array}{l} P(a \mid b{,}e)\cdot \textsc{EnumAll}(\langle j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=\mathsf{T},a=\mathsf{T}\rangle) \\ P(\neg a \mid b{,}e)\cdot \textsc{EnumAll}(\langle j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=\mathsf{T},a=\mathsf{F}\rangle) \end{array}\right. \\ P(\neg e)\cdot\left[+\begin{array}{l} P(a \mid b{,}\neg e)\cdot \textsc{EnumAll}(\langle j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=\mathsf{F},a=\mathsf{T}\rangle) \\ P(\neg a \mid b{,}\neg e)\cdot \textsc{EnumAll}(\langle j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{T},e=\mathsf{F},a=\mathsf{F}\rangle \end{array}\right. \end{array}\right.$$
  - $P_1 := P(\neg b) \cdot$
    $$\left[+\begin{array}{l} P(e)\cdot\left[+\begin{array}{l} P(a \mid \neg b{,}e)\cdot \textsc{EnumAll}(\langle j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=\mathsf{T},a=\mathsf{T}\rangle) \\ P(\neg a \mid \neg b{,}e)\cdot \textsc{EnumAll}(\langle j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=\mathsf{T},a=\mathsf{F}\rangle) \end{array}\right. \\ P(\neg e)\cdot\left[+\begin{array}{l} P(a \mid \neg b{,}\neg e)\cdot \textsc{EnumAll}(\langle j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=\mathsf{F},a=\mathsf{T}\rangle \\ P(\neg a \mid \neg b{,}\neg e)\cdot \textsc{EnumAll}(\langle j,m\rangle,\langle j=\mathsf{T},m=\mathsf{T},b=\mathsf{F},e=\mathsf{F},a= \end{array}\right. \end{array}\right.$$
  - $\Leftarrow \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j=\mathsf{T},m=\mathsf{T})=\alpha(\mathbb{P}(b)\cdot(\sum_{b_e\in\{\mathsf{T},\mathsf{F}\}}P(e=b_e)\cdot(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}}\mathbb{P}(a=b_a|e=b_e,b)\cdot\cdot)))$$

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b) \cdot$

$$\left[\begin{array}{l} P(e) \cdot \left[\begin{array}{l} P(a \mid b,e) \cdot \underbrace{\textsc{EnumAll}(\langle j, m\rangle, \langle j = T, m = T, b = T, e = T, a = T\rangle)}_{=P(j \mid a)\cdot\textsc{EnumAll}(\langle m\rangle,\langle j=T,m=T,b=T,e=T,a=T\rangle)} \\ + \\ P(\neg a \mid b,e) \cdot \underbrace{\textsc{EnumAll}(\langle j, m\rangle, \langle j = T, m = T, b = T, e = T, a = F\rangle)}_{=P(j \mid \neg a)\cdot\textsc{EnumAll}(\langle m\rangle,\langle j=T,m=T,b=T,e=T,a=F\rangle)} \end{array}\right. \\ + \\ P(\neg e) \cdot \left[\begin{array}{l} P(a \mid b,\neg e) \cdot \underbrace{\textsc{EnumAll}(\langle j, m\rangle, \langle j = T, m = T, b = T, e = F, a = T\rangle}_{=P(j \mid a)\cdot\textsc{EnumAll}(\langle m\rangle,\langle j=T,m=T,b=T,e=F,a=T\rangle)} \\ + \\ P(\neg a \mid b,\neg e) \cdot \underbrace{\textsc{EnumAll}(\langle j, m\rangle, \langle j = T, m = T, b = T, e = F, a = F}_{=P(j \mid \neg a)\cdot\textsc{EnumAll}(\langle m\rangle,\langle j=T,m=T,b=T,e=F,a=F\rangle)} \end{array}\right. \end{array}\right.$$

  - $P_1 := P(\neg b) \cdot$

$$\left[\begin{array}{l} P(e) \cdot \left[\begin{array}{l} P(a \mid \neg b,e) \cdot \underbrace{\textsc{EnumAll}(\langle j, m\rangle, \langle j = T, m = T, b = F, e = T, a = T\rangle)}_{=P(j \mid a)\cdot\textsc{EnumAll}(\langle m\rangle,\langle j=T,m=T,b=F,e=T,a=T\rangle)} \\ + \\ P(\neg a \mid \neg b,e) \cdot \underbrace{\textsc{EnumAll}(\langle j, m\rangle, \langle j = T, m = T, b = F, e = T, a = F\rangle)}_{=P(j \mid \neg a)\cdot\textsc{EnumAll}(\langle m\rangle,\langle j=T,m=T,b=F,e=T,a=F\rangle)} \end{array}\right. \\ + \\ P(\neg e) \cdot \left[\begin{array}{l} P(a \mid \neg b,\neg e) \cdot \underbrace{\textsc{EnumAll}(\langle j, m\rangle, \langle j = T, m = T, b = F, e = F, a = T}_{=P(j \mid a)\cdot\textsc{EnumAll}(\langle m\rangle,\langle j=T,m=T,b=F,e=F,a=T\rangle)} \\ + \\ P(\neg a \mid \neg b,\neg e) \cdot \underbrace{\textsc{EnumAll}(\langle j, m\rangle, \langle j = T, m = T, b = F, e = F, a =}_{=P(j \mid \neg a)\cdot\textsc{EnumAll}(\langle m\rangle,\langle j=T,m=T,b=F,e=F} \end{array}\right. \end{array}\right.$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b) \cdot$

$$\left[+\begin{array}{l} P(e) \cdot \left[+\begin{array}{l} P(a \mid b,e) \cdot P(j \mid a) \cdot \textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \mathsf{T}, a = \\ P(\neg a \mid b,e) \cdot P(j \mid \neg a) \cdot \textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \mathsf{T}, \end{array}\right. \\ P(\neg e) \cdot \left[+\begin{array}{l} P(a \mid b,\neg e) \cdot P(j \mid a) \cdot \textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \mathsf{F}, \\ P(\neg a \mid b,\neg e) \cdot P(j \mid \neg a) \cdot \textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \end{array}\right. \end{array}\right.$$

  - $P_1 := P(\neg b) \cdot$

$$\left[+\begin{array}{l} P(e) \cdot \left[+\begin{array}{l} P(a \mid \neg b,e) \cdot P(j \mid a) \cdot \textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \mathsf{T}, a \\ P(\neg a \mid \neg b,e) \cdot P(j \mid \neg a) \cdot \textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \mathsf{T} \end{array}\right. \\ P(\neg e) \cdot \left[+\begin{array}{l} P(a \mid \neg b,\neg e) \cdot P(j \mid a) \cdot \textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \mathsf{F} \\ P(\neg a \mid \neg b,\neg e) \cdot P(j \mid \neg a) \cdot \textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \end{array}\right. \end{array}\right.$$

  $\Leftarrow \alpha(\langle P_0, P_1 \rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b) \cdot (\sum_{b_e \in \{\mathsf{T},\mathsf{F}\}} P(e = b_e) \cdot (\sum_{b_a \in \{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a = b_a | e = b_e, b) \cdot P(j \mid a = b_a) \cdot)$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b) \cdot$

$$
\left[\begin{array}{l}
P(e) \cdot \left[\begin{array}{l}
P(a \mid b{,}e) \cdot P(j \mid a) \cdot \underbrace{\textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \mathsf{T}, a =}_{=P(m \mid a) \cdot \textsc{EnumAll}(\langle\rangle, \langle j=\mathsf{T}, m=\mathsf{T}, b=\mathsf{T}, e=\mathsf{T}, a=\mathsf{T}\rangle))} \\
+ \\
P(\neg a \mid b{,}e) \cdot P(j \mid \neg a) \cdot \underbrace{\textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \mathsf{T},}_{=P(m \mid \neg a) \cdot \textsc{EnumAll}(\langle\rangle, \langle j=\mathsf{T}, m=\mathsf{T}, b=\mathsf{T}, e=\mathsf{T}, a=}
\end{array}\right. \\
+ \\
P(\neg e) \cdot \left[\begin{array}{l}
P(a \mid b{,}\neg e) \cdot P(j \mid a) \cdot \underbrace{\textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e = \mathsf{F},}_{=P(m \mid a) \cdot \textsc{EnumAll}(\langle\rangle, \langle j=\mathsf{T}, m=\mathsf{T}, b=\mathsf{T}, e=\mathsf{F}, a=\mathsf{T}} \\
+ \\
P(\neg a \mid b{,}\neg e) \cdot P(j \mid \neg a) \cdot \underbrace{\textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e =}_{=P(m \mid \neg a) \cdot \textsc{EnumAll}(\langle\rangle, \langle j=\mathsf{T}, m=\mathsf{T}, b=\mathsf{T}, e=\mathsf{F}}
\end{array}\right.
\end{array}\right.
$$

  - $P_1 := P(\neg b) \cdot$

$$
\left[\begin{array}{l}
P(e) \cdot \left[\begin{array}{l}
P(a \mid \neg b{,}e) \cdot P(j \mid a) \cdot \underbrace{\textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \mathsf{T}, a}_{=P(m \mid a) \cdot \textsc{EnumAll}(\langle\rangle, \langle j=\mathsf{T}, m=\mathsf{T}, b=\mathsf{F}, e=\mathsf{T}, a=\mathsf{T}\rangle))} \\
+ \\
P(\neg a \mid \neg b{,}e) \cdot P(j \mid \neg a) \cdot \underbrace{\textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \mathsf{T}}_{=P(m \mid \neg a) \cdot \textsc{EnumAll}(\langle\rangle, \langle j=\mathsf{T}, m=\mathsf{T}, b=\mathsf{F}, e=\mathsf{T}, a=}
\end{array}\right. \\
+ \\
P(\neg e) \cdot \left[\begin{array}{l}
P(a \mid \neg b{,}\neg e) \cdot P(j \mid a) \cdot \underbrace{\textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e = \mathsf{F}}_{=P(m \mid a) \cdot \textsc{EnumAll}(\langle\rangle, \langle j=\mathsf{T}, m=\mathsf{T}, b=\mathsf{F}, e=\mathsf{F}, a=} \\
+ \\
P(\neg a \mid \neg b{,}\neg e) \cdot P(j \mid \neg a) \cdot \underbrace{\textsc{EnumAll}(\langle m \rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, e}_{=P(m \mid \neg a) \cdot \textsc{EnumAll}(\langle\rangle, \langle j=\mathsf{T}, m \quad e=}
\end{array}\right.
\end{array}\right.
$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b) \cdot$

$$\left[+ \begin{array}{l} P(e)\cdot \left[+ \begin{array}{l} P(a \mid b,e)\cdot P(j \mid a)\cdot P(m \mid a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e \\ P(\neg a \mid b,e)\cdot P(j \mid \neg a)\cdot P(m \mid \neg a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \end{array}\right. \\ P(\neg e)\cdot \left[+ \begin{array}{l} P(a \mid b,\neg e)\cdot P(j \mid a)\cdot P(m \mid a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T} \\ P(\neg a \mid b,\neg e)\cdot P(j \mid \neg a)\cdot P(m \mid \neg a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, \end{array}\right. \end{array}\right.$$

  - $P_1 := P(\neg b) \cdot$

$$\left[+ \begin{array}{l} P(e)\cdot \left[+ \begin{array}{l} P(a \mid \neg b,e)\cdot P(j \mid a)\cdot P(m \mid a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, \\ P(\neg a \mid \neg b,e)\cdot P(j \mid \neg a)\cdot P(m \mid \neg a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b \end{array}\right. \\ P(\neg e)\cdot \left[+ \begin{array}{l} P(a \mid \neg b,\neg e)\cdot P(j \mid a)\cdot P(m \mid a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \\ P(\neg a \mid \neg b,\neg e)\cdot P(j \mid \neg a)\cdot P(m \mid \neg a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T} \end{array}\right. \end{array}\right.$$

$\Leftarrow \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b)\cdot(\sum_{b_e \in \{\mathsf{T},\mathsf{F}\}} P(e = b_e)\cdot(\sum_{b_a \in \{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a = b_a|e = b_e, b)\cdot P(j \mid a = b_a)\cdot$$

Enumeration: Example

- Variable order: b, e, a, j, m
  - $P_0 := P(b)\cdot$

$$\left[+\begin{array}{l} P(e)\cdot\left[+\begin{array}{l} P(a \mid b,e)\cdot P(j \mid a)\cdot P(m \mid a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T}, e \\ P(\neg a \mid b,e)\cdot P(j \mid \neg a)\cdot P(m \mid \neg a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \end{array}\right. \\ P(\neg e)\cdot\left[+\begin{array}{l} P(a \mid b,\neg e)\cdot P(j \mid a)\cdot P(m \mid a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{T} \\ P(\neg a \mid b,\neg e)\cdot P(j \mid \neg a)\cdot P(m \mid \neg a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b \end{array}\right. \end{array}\right.$$

  - $P_1 := P(\neg b)\cdot$

$$\left[+\begin{array}{l} P(e)\cdot\left[+\begin{array}{l} P(a \mid \neg b,e)\cdot P(j \mid a)\cdot P(m \mid a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \mathsf{F}, \\ P(\neg a \mid \neg b,e)\cdot P(j \mid \neg a)\cdot P(m \mid \neg a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \end{array}\right. \\ P(\neg e)\cdot\left[+\begin{array}{l} P(a \mid \neg b,\neg e)\cdot P(j \mid a)\cdot P(m \mid a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T}, b = \\ P(\neg a \mid \neg b,\neg e)\cdot P(j \mid \neg a)\cdot P(m \mid \neg a)\cdot \textsc{EnumAll}(\langle\rangle, \langle j = \mathsf{T}, m = \mathsf{T} \end{array}\right. \end{array}\right.$$

$\Leftarrow\ \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b)\cdot(\sum_{b_e\in\{\mathsf{T},\mathsf{F}\}} P(e = b_e)\cdot(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a = b_a|e = b_e, b)\cdot P(j \mid a = b_a)\cdot$$

Enumeration: Example

- Variable order: b, e, a, j, m
  - $$P_0 := P(b)\cdot\left[+\begin{array}{l} P(e)\cdot\left[+\begin{array}{l} P(a\mid b,e)\cdot P(j\mid a)\cdot P(m\mid a)\cdot 1.0\\ P(\neg a\mid b,e)\cdot P(j\mid \neg a)\cdot P(m\mid \neg a)\cdot 1.0\end{array}\right.\\ P(\neg e)\cdot\left[+\begin{array}{l} P(a\mid b,\neg e)\cdot P(j\mid a)\cdot P(m\mid a)\cdot 1.0\\ P(\neg a\mid b,\neg e)\cdot P(j\mid \neg a)\cdot P(m\mid \neg a)\cdot 1.0\end{array}\right.\end{array}\right.$$
  - $$P_1 := P(\neg b)\cdot\left[+\begin{array}{l} P(e)\cdot\left[+\begin{array}{l} P(a\mid \neg b,e)\cdot P(j\mid a)\cdot P(m\mid a)\cdot 1.0\\ P(\neg a\mid \neg b,e)\cdot P(j\mid \neg a)\cdot P(m\mid \neg a)\cdot 1.0\end{array}\right.\\ P(\neg e)\cdot\left[+\begin{array}{l} P(a\mid \neg b,\neg e)\cdot P(j\mid a)\cdot P(m\mid a)\cdot 1.0\\ P(\neg a\mid \neg b,\neg e)\cdot P(j\mid \neg a)\cdot P(m\mid \neg a)\cdot 1.0\end{array}\right.\end{array}\right.$$

  $\Leftarrow\ \alpha(\langle P_0, P_1\rangle)$

$$\mathbb{P}(b|j=\mathsf{T},m=\mathsf{T})=\alpha(\mathbb{P}(b)\cdot(\sum_{b_e\in\{\mathsf{T},\mathsf{F}\}}P(e=b_e)\cdot(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}}\mathbb{P}(a=b_a|e=b_e,b)\cdot P(j\mid a=b_a)\cdot$$

Enumeration: Example

- Variable order: b, e, a, j, m

- $$P_0 := P(b) \cdot \left[+ \begin{array}{l} P(e) \cdot \left[+ \begin{array}{l} P(a \mid b,e) \cdot P(j \mid a) \cdot P(m \mid a) \cdot 1.0 \\ P(\neg a \mid b,e) \cdot P(j \mid \neg a) \cdot P(m \mid \neg a) \cdot 1.0 \end{array} \right. \\ P(\neg e) \cdot \left[+ \begin{array}{l} P(a \mid b,\neg e) \cdot P(j \mid a) \cdot P(m \mid a) \cdot 1.0 \\ P(\neg a \mid b,\neg e) \cdot P(j \mid \neg a) \cdot P(m \mid \neg a) \cdot 1.0 \end{array} \right. \end{array} \right.$$

- $$P_1 := P(\neg b) \cdot \left[+ \begin{array}{l} P(e) \cdot \left[+ \begin{array}{l} P(a \mid \neg b,e) \cdot P(j \mid a) \cdot P(m \mid a) \cdot 1.0 \\ P(\neg a \mid \neg b,e) \cdot P(j \mid \neg a) \cdot P(m \mid \neg a) \cdot 1.0 \end{array} \right. \\ P(\neg e) \cdot \left[+ \begin{array}{l} P(a \mid \neg b,\neg e) \cdot P(j \mid a) \cdot P(m \mid a) \cdot 1.0 \\ P(\neg a \mid \neg b,\neg e) \cdot P(j \mid \neg a) \cdot P(m \mid \neg a) \cdot 1.0 \end{array} \right. \end{array} \right.$$

$$\Leftarrow \quad \underbrace{\alpha(\langle P_0, P_1 \rangle)}_{=\langle \frac{P_0}{P_0+P_1}, \frac{P_1}{P_0+P_1} \rangle}$$

$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b) \cdot (\sum_{b_e \in \{\mathsf{T},\mathsf{F}\}} P(e = b_e) \cdot (\sum_{b_a \in \{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a = b_a | e = b_e, b) \cdot P(j \mid a = b_a) \cdot$$

SOME RIGHTS RESERVED

Enumeration: Example

- Variable order: b, e, a, j, m
  - $$P_0 := P(b) \cdot \left[+ \begin{array}{l} P(e) \cdot \left[+ \begin{array}{l} P(a \mid b,e) \cdot P(j \mid a) \cdot P(m \mid a) \cdot 1.0 \\ P(\neg a \mid b,e) \cdot P(j \mid \neg a) \cdot P(m \mid \neg a) \cdot 1.0 \end{array}\right. \\ P(\neg e) \cdot \left[+ \begin{array}{l} P(a \mid b,\neg e) \cdot P(j \mid a) \cdot P(m \mid a) \cdot 1.0 \\ P(\neg a \mid b,\neg e) \cdot P(j \mid \neg a) \cdot P(m \mid \neg a) \cdot 1.0 \end{array}\right. \end{array}\right.$$
  - $$P_1 := P(\neg b) \cdot \left[+ \begin{array}{l} P(e) \cdot \left[+ \begin{array}{l} P(a \mid \neg b,e) \cdot P(j \mid a) \cdot P(m \mid a) \cdot 1.0 \\ P(\neg a \mid \neg b,e) \cdot P(j \mid \neg a) \cdot P(m \mid \neg a) \cdot 1.0 \end{array}\right. \\ P(\neg e) \cdot \left[+ \begin{array}{l} P(a \mid \neg b,\neg e) \cdot P(j \mid a) \cdot P(m \mid a) \cdot 1.0 \\ P(\neg a \mid \neg b,\neg e) \cdot P(j \mid \neg a) \cdot P(m \mid \neg a) \cdot 1.0 \end{array}\right. \end{array}\right.$$

  $\Leftarrow \langle \frac{P_0}{P_0+P_1}, \frac{P_1}{P_0+P_1} \rangle$

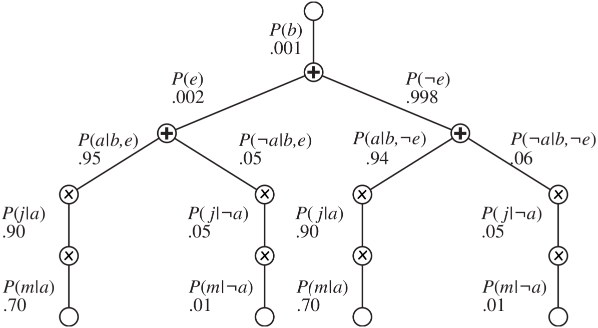
$$\mathbb{P}(b|j = \mathsf{T}, m = \mathsf{T}) = \alpha(\mathbb{P}(b) \cdot (\sum_{b_e \in \{\mathsf{T},\mathsf{F}\}} P(e = b_e) \cdot (\sum_{b_a \in \{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a = b_a | e = b_e, b) \cdot P(j \mid a = b_a) \cdot$$

SOME RIGHTS RESERVED

The Evaluation of $P(b \mid j,m)$ as a "Search Tree"

$$\mathbb{P}(b|j,m) = \alpha(\mathbb{P}(b)\cdot(\sum_{b_e\in\{\mathsf{T},\mathsf{F}\}} P(e=b_e)\cdot(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a=b_a|e=b_e,b)\cdot P(j \mid a=b_a)$$

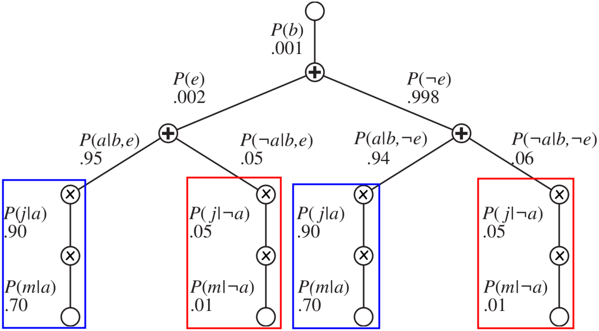
Note: Enumerate-Query corresponds to depth-first traversal of an arithmetic expression-tree:



The Evaluation of $P(b \mid j,m)$ as a "Search Tree"

$$\mathbb{P}(b|j,m) = \alpha(\mathbb{P}(b)\cdot(\sum_{b_e\in\{T,F\}} P(e=b_e)\cdot(\sum_{b_a\in\{T,F\}} \mathbb{P}(a=b_a|e=b_e,b)\cdot P(j \mid a=b_a)$$

Note: Enumerate-Query corresponds to depth-first traversal of an arithmetic expression-tree:



Variable Elimination 2

- $$\mathbb{P}(b|j,m) = \alpha(\mathbb{P}(b)\cdot(\sum_{b_e\in\{T,F\}} P(e=b_e)\cdot(\sum_{b_a\in\{T,F\}} \mathbb{P}(a=b_a|e=b_e,b)\cdot P(j\mid a=$$

  The last two factors $P(j\mid a=b_a)$, $P(m\mid a=b_a)$ only depend on a, but are "trapped" behind the summation over e, hence computed twice in two distinct recursive calls to EnumAll

- **Idea:** Instead of left-to-right (top-down DFS), operate right-to-left (bottom-up) and store intermediate "factors" along with their "dependencies":

$$\alpha(\underbrace{\mathbb{P}(b)}_{f_7(b)}\cdot(\underbrace{\sum_{b_e\in\{T,F\}} \underbrace{P(e=b_e)}_{f_5(e)}\cdot\underbrace{(\sum_{b_a\in\{T,F\}} \underbrace{\mathbb{P}(a=b_a|e=b_e,b)}_{f_3(a,b,e)}\cdot\underbrace{P(j\mid a=b_a)}_{f_2(a)}\cdot\underbrace{P(m\mid a}_{f_1(a}}_{f_4(b,e)}}_{f_6(b)}$$

Variable Elimination: Example

- We only show variable elimination by example: (implementation details get tricky, but the idea is simple)
  $\mathbb{P}(b)\cdot$
  $(\sum_{b_e\in\{\mathsf{T},\mathsf{F}\}} P(e=b_e)\cdot(\sum_{b_a\in\{\mathsf{T},\mathsf{F}\}} \mathbb{P}(a=b_a|e=b_e,b)\cdot P(j\mid a=b_a)\cdot P(m\mid a=$
  - Assume reverse topological order of variables: m, j, a, e, b
    - m is an evidence variable with value T and dependency a, which is a hidden variable. We introduce a new "factor" $\mathrm{f}(a){:=}\mathrm{f}_1(a) := \langle P(m\mid a), P(m\mid\neg a)\rangle$.
    - j works analogously, $\mathrm{f}_2(a) := \langle P(j\mid a), P(j\mid\neg a)\rangle$. We "multiply" with the existing factor, yielding
      $\mathrm{f}(a) := \langle\mathrm{f}_1(a)\cdot\mathrm{f}_2(a), \mathrm{f}_1(\neg a)\cdot\mathrm{f}_2(\neg a)\rangle{=}\langle P(m\mid a)\cdot P(j\mid a), P(m\mid\neg a)\cdot P(j\mid\neg a)\rangle$
    - a is a hidden variable with dependencies e (hidden) and b (query).
      1. We introduce a new "factor" $\mathrm{f}_3(a,e,b)$, a $2\times 2\times 2$ table with the relevant conditional probabilities $\mathbb{P}(a|e,b)$.
      2. We multiply each entry of f_3 with the relevant entries of the existing factor f, yielding $\mathrm{f}(a,e,b)$.
      3. We "sum out" the resulting factor over a, yielding a new factor $\mathrm{f}(e,b)=\mathrm{f}(a,e,b)+\mathrm{f}(\neg a,e,b)$.
    - ...
- $\rightsquigarrow$ can speed things up by a factor of 1000! (or more, depending on the order of variables!)

SOME RIGHTS RESERVED

The Complexity of Exact Inference

- **Definition 3.4.** A graph G is called singly connected, or a polytree (otherwise multiply connected), if there is at most one undirected path between any two nodes in G.
- **Theorem 3.5 (Good News).** *On singly connected Bayesian networks, variable elimination runs in polynomial time.*

SOME RIGHTS RESERVED

The Complexity of Exact Inference

- **Definition 3.8.** A graph G is called singly connected, or a polytree (otherwise multiply connected), if there is at most one undirected path between any two nodes in G.
- **Theorem 3.9 (Good News).** *On singly connected Bayesian networks, variable elimination runs in polynomial time.*
- Is our BN for Mary & John a polytree? (Yes.)

The Complexity of Exact Inference

- **Definition 3.12.** A graph G is called singly connected, or a polytree (otherwise multiply connected), if there is at most one undirected path between any two nodes in G.
- **Theorem 3.13 (Good News).** *On singly connected Bayesian networks, variable elimination runs in polynomial time.*
- Is our BN for Mary & John a polytree? (Yes.)
- **Theorem 3.14 (Bad News).** *For multiply connected Bayesian networks, probabilistic inference is #P-hard.* (#P *is harder than* NP, *i.e.* NP $\subseteq$ #P)
- **So?:** Life goes on ... In the hard cases, if need be we can throw exactitude to the winds and approximate.
- **Example 3.15.** Sampling techniques as in MCTS.

SOME RIGHTS RESERVED

22.4 Conclusion

SOME RIGHTS RESERVED

Summary

- Bayesian networks (BN) are a wide-spread tool to model uncertainty, and to reason about it. A BN represents conditional independence relations between random variables. It consists of a graph encoding the variable dependencies, and of conditional probability tables (CPTs).
- Given a variable ordering, the BN is small if every variable depends on only a few of its predecessors.
- Probabilistic inference requires to compute the probability distribution of a set of query variables, given a set of evidence variables whose values we know. The remaining variables are hidden.
- Inference by enumeration takes a BN as input, then applies Normalization+Marginalization, the chain rule, and exploits conditional independence. This can be viewed as a tree search that branches over all values of the hidden variables.
- Variable elimination avoids unnecessary computation. It runs in polynomial time for poly-tree BNs. In general, exact probabilistic inference is #P-hard. Approximate probabilistic inference methods exist.

SOME RIGHTS RESERVED

Topics We Didn't Cover Here

- **Inference by sampling**: A whole zoo of methods for doing this exists.

SOME RIGHTS RESERVED

Topics We Didn't Cover Here

- **Inference by sampling**: A whole zoo of methods for doing this exists.
- **Clustering**: Pre-combining subsets of variables to reduce the running time of inference.

Topics We Didn't Cover Here

- **Inference by sampling**: A whole zoo of methods for doing this exists.
- **Clustering**: Pre-combining subsets of variables to reduce the running time of inference.
- **Compilation to SAT**: More precisely, to "weighted model counting" in CNF formulas. Model counting extends DPLL with the ability to determine the number of satisfying interpretations. Weighted model counting allows to define a mass for each such interpretation (= the probability of an atomic event).

SOME RIGHTS RESERVED

Topics We Didn't Cover Here

- **Inference by sampling**: A whole zoo of methods for doing this exists.
- **Clustering**: Pre-combining subsets of variables to reduce the running time of inference.
- **Compilation to SAT**: More precisely, to "weighted model counting" in CNF formulas. Model counting extends DPLL with the ability to determine the number of satisfying interpretations. Weighted model counting allows to define a mass for each such interpretation (= the probability of an atomic event).
- **Dynamic BN**: BN with one slice of variables at each "time step", encoding probabilistic behavior over time.

SOME RIGHTS RESERVED

Topics We Didn't Cover Here

- **Inference by sampling**: A whole zoo of methods for doing this exists.
- **Clustering**: Pre-combining subsets of variables to reduce the running time of inference.
- **Compilation to SAT**: More precisely, to "weighted model counting" in CNF formulas. Model counting extends DPLL with the ability to determine the number of satisfying interpretations. Weighted model counting allows to define a mass for each such interpretation (= the probability of an atomic event).
- **Dynamic BN**: BN with one slice of variables at each "time step", encoding probabilistic behavior over time.
- **Relational BN**: BN with predicates and object variables.

SOME RIGHTS RESERVED

Topics We Didn't Cover Here

- **Inference by sampling**: A whole zoo of methods for doing this exists.
- **Clustering**: Pre-combining subsets of variables to reduce the running time of inference.
- **Compilation to SAT**: More precisely, to "weighted model counting" in CNF formulas. Model counting extends DPLL with the ability to determine the number of satisfying interpretations. Weighted model counting allows to define a mass for each such interpretation (= the probability of an atomic event).
- **Dynamic BN**: BN with one slice of variables at each "time step", encoding probabilistic behavior over time.
- **Relational BN**: BN with predicates and object variables.
- **First-order BN**: Relational BN with quantification, i.e. probabilistic logic. E.g., the BLOG language developed by Stuart Russel and co-workers.

SOME RIGHTS RESERVED

Chapter 23
Making Simple Decisions Rationally

SOME RIGHTS RESERVED

23.1 Introduction

SOME RIGHTS RESERVED

Overview

- We now know how to update our world model, represented as (a set of) random variables, given observations. Now we need to *act*.

SOME RIGHTS RESERVED

Overview

- We now know how to update our world model, represented as (a set of) random variables, given observations. Now we need to *act*.
- For that we need to answer two questions:
- **Questions:**
  - Given a world model and a set of *actions*, what will the likely consequences of each action be?
  - How “good” are these consequences?

SOME RIGHTS RESERVED

Overview

- We now know how to update our world model, represented as (a set of) random variables, given observations. Now we need to *act*.
- For that we need to answer two questions:
- **Questions:**
  - Given a world model and a set of *actions*, what will the likely consequences of each action be?
  - How "good" are these consequences?
- **Idea:**
  - Represent actions as "special random variables":
    Given disjoint actions $a_1, \ldots, a_n$, introduce a random variable A with domain $\{a_1, \ldots, a_n\}$. Then we can model/query $\mathbb{P}(X|A = a_i)$.
  - Assign *numerical values* to the possible outcomes of actions (i.e. a function $u\colon \mathrm{dom}(X) \to \mathbb{R}$) indicating their desirability.
  - Choose the action that maximizes the *expected value* of u

SOME RIGHTS RESERVED

Overview

- We now know how to update our world model, represented as (a set of) random variables, given observations. Now we need to *act*.
- For that we need to answer two questions:
- **Questions:**
  - Given a world model and a set of *actions*, what will the likely consequences of each action be?
  - How "good" are these consequences?
- **Idea:**
  - Represent actions as "special random variables":
    Given disjoint actions $a_1, \ldots, a_n$, introduce a random variable A with domain $\{a_1, \ldots, a_n\}$. Then we can model/query $\mathbb{P}(X|A = a_i)$.
  - Assign *numerical values* to the possible outcomes of actions (i.e. a function $u\colon \mathrm{dom}(X) \to \mathbb{R}$) indicating their desirability.
  - Choose the action that maximizes the *expected value* of u

Definition 1.4. Decision theory investigates decision problems, i.e. how a utility-based agent a deals with choosing among actions based on the desirability of their outcomes given by a real-valued utility function U on states $s \in S$: i.e. $U\colon S \to \mathbb{R}$.

-

SOME RIGHTS RESERVED

Decision Theory

- If our states are random variables, then we obtain a random variable for the utility function:
- **Observation:** Let $X_i : \Omega \to D_i$ random variables on a probability model $\langle\Omega, P\rangle$ and $f : D_1 \times \ldots \times D_n \to E$. Then $F(x) := f(X_0(x), \ldots, X_n(x))$ is a random variable $\Omega \to E$.

SOME RIGHTS RESERVED

Decision Theory

- If our states are random variables, then we obtain a random variable for the utility function:
- **Observation:** Let $X_i\colon\Omega\rightarrow D_i$ random variables on a probability model $\langle\Omega,P\rangle$ and $f\colon D_1\times\ldots\times D_n\rightarrow E$. Then $F(x):=f(X_0(x),\ldots,X_n(x))$ is a random variable $\Omega\rightarrow E$.
- **Definition 1.7.** Given a probability model $\langle\Omega,P\rangle$ and a random variable $X\colon\Omega\rightarrow D$ with $D\subseteq\mathbb{R}$, then $E(X):=\sum_{x\in D}P(X=x)\cdot x$ is called the expected value (or expectation) of X. (Assuming the sum/series is actually defined!)
  Analogously, let $e_1,\ldots,e_n$ a sequence of events. Then the expected value of X given $e_1,\ldots,e_n$ is defined as $E(X|e_1,\ldots,e_n):=\sum_{x\in D}\mu(X=x\mid e_1,\ldots,e_n)\cdot x$.

SOME RIGHTS RESERVED

Decision Theory

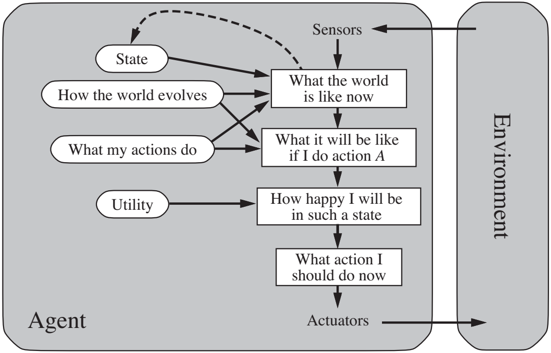
- If our states are random variables, then we obtain a random variable for the utility function:
- **Observation:** Let $X_i\colon\Omega\to D_i$ random variables on a probability model $\langle\Omega,P\rangle$ and $f\colon D_1\times\ldots\times D_n\to E$. Then $F(x):=f(X_0(x),\ldots,X_n(x))$ is a random variable $\Omega\to E$.
- **Definition 1.9.** Given a probability model $\langle\Omega,P\rangle$ and a random variable $X\colon\Omega\to D$ with $D\subseteq\mathbb{R}$, then $E(X):=\sum_{x\in D}P(X=x)\cdot x$ is called the expected value (or expectation) of X. (Assuming the sum/series is actually defined!)
  Analogously, let $e_1,\ldots,e_n$ a sequence of events. Then the expected value of X given $e_1,\ldots,e_n$ is defined as $E(X|e_1,\ldots,e_n):=\sum_{x\in D}\mu(X=x\mid e_1,\ldots,e_n)\cdot x$.
- Putting things together:
- **Definition 1.10.** Let $A\colon\Omega\to D$ a random variable (where D is a set of actions) $X_i\colon\Omega\to D_i$ random variables (the state), and $U\colon D_1\times\ldots\times D_n\to\mathbb{R}$ a utility function. Then the expected utility of the action $a\in D$ is the expected value of U (interpreted as a random variable) given $A=a$; i.e.

$$\mathrm{EU}(a):=\sum_{\langle x_1,\ldots,x_n\rangle\in D_1\times\ldots\times D_n}\mu(X_1=x_1,\ldots,X_n=x_n\mid A=a)\cdot U(x_1,\ldots,x_n)$$

SOME RIGHTS RESERVED

Utility-based Agents

- **Definition 1.11.** A utility-based agent uses a world model along with a utility function that models its preferences among the states of that world. It chooses the action that leads to the best expected utility.
- **Agent Schema:**


Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action *A*
How happy I will be in such a state
What action I should do now
Actuators
Agent
Environment

SOME RIGHTS RESERVED

Maximizing Expected Utility (Ideas)

- **Definition 1.12 (MEU principle for Rationality).** We call an action rational if it maximizes expected utility (MEU). An utility-based agent is called rational, iff it always chooses a rational action.
- **Hooray:** This solves all of AI. (in principle)
- **Problem:** There is a long, long way towards an operationalization ;)
- **Note:** An agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities.
- **Example 1.13.** A reflex agent for tic tac toe based on a perfect lookup table is rational if we take (the negative of) "winning/drawing in n steps" as the utility function.
- **Example 1.14 (AI1).** Heuristics in tree search (greedy search, A^*) and game-play (minimax, alpha-beta pruning) maximize "expected" utility.
  $\Rightarrow$ In fully observable, deterministic environments, "expected utility" reduces to a specific determined utility value:
  $\mathrm{EU}(a) = U(T(S(s,e),a))$, where e the most recent percept, s the current state, S the sensor function and T the transition function.
- Now let's figure out how to actually assign utilities!

SOME RIGHTS RESERVED

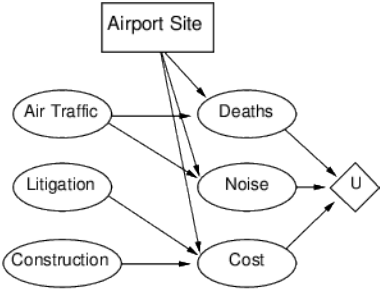
23.2 Decision Networks

SOME RIGHTS RESERVED

Decision networks

Definition 2.1. A decision network is a Bayesian network with two additional kinds of nodes:

- action nodes, representing a set of possible actions, and (square nodes)
- A single utility node (also called value node). (diamond node)



- **General Algorithm:** Given evidence $E_j = e_j$, and action nodes $A_1, \ldots, A_k$, compute the expected utility of each action, given the evidence, i.e. return the sequence of actions

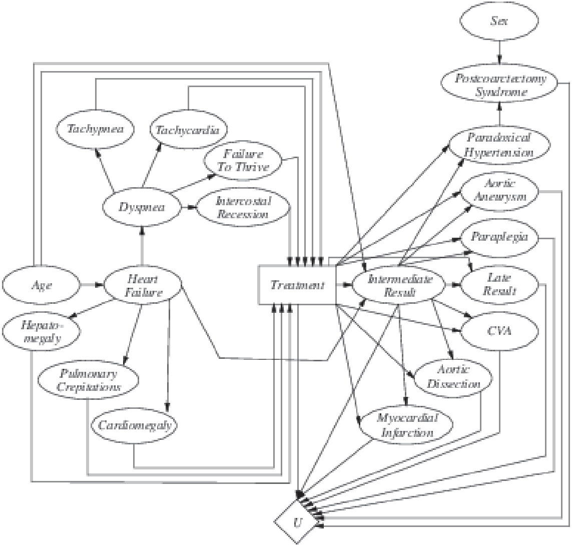
$$\operatorname{argmax}\ a_1, \ldots, a_k \overbrace{\sum_{\langle x_1, \ldots, x_n \rangle} \underbrace{\mu(X_i = x_i \mid A_1 = a_1, \ldots, A_k = a_k, E_j = e_j)}_{\text{usual Bayesian Network inference}} \cdot U(X_i = x_i)}^{=\text{expected utility of } a_1, \ldots, a_k}$$

- **Note** the sheer amount of summands in the sum above in the general case! (⇒ We will simplify where possible later)

SOME RIGHTS RESERVED

Decision Networks: Example

- **Example 2.2 (A Decision-Network for Aortic Coarctation).** from [**Lucas:kadtes96**]



SOME RIGHTS RESERVED

23.3 Preferences and Utilities

SOME RIGHTS RESERVED

Preferences in Deterministic Environments

- **Problem:** How do we determine the utility of a state? (We cannot directly measure our satisfaction/happiness in a possibly future state...) (What unit would we even use?)
- **Example 3.1.** I have to decide whether to go to class today (or sleep in). What is the utility of this lecture? (obviously 42)
- **Idea:** We can let people/agents choose between two states (subjective preference) and derive a utility from these choices.
- **Example 3.2.** "*Give me your cell-phone or I will give you a bloody nose*". $\leadsto$ To make a decision in a deterministic environment, the agent must determine whether it prefers a state without phone to one with a bloody nose?
- **Definition 3.3.** Given states A and B (we call them prizes) an agent can express preferences of the form
  - $A \succ B$ A prefered over B
  - $A \sim B$ indifference between A and B
  - $A \succeq B$ B not prefered over A

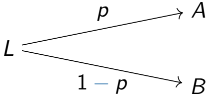
  i.e. Given a set $\mathcal{S}$ (of states), we define binary relations $\succ$ and $\sim$ on $\mathcal{S}$.

SOME RIGHTS RESERVED

Preferences in Non-Deterministic Environments

- **Problem:** In nondeterministic environments we do not have full information about the states we choose between.
- **Example 3.4 (Airline Food).** "*Do you want chicken or pasta*" (but we cannot see through the tin foil)
- **Definition 3.5.**

Let $\mathcal{S}$ a set of states. We call a random variable X with domain $\{A_1, \ldots, A_n\} \subseteq \mathcal{S}$ a lottery and write $[p_1,A_1; \ldots; p_n,A_n]$, where $p_i = P(X = A_i)$.



- **Idea:** A lottery represents the result of a nondeterministic action that can have outcomes A_i with prior probability p_i. For the binary case, we use $[p,A;1-p,B]$. We can then extend preferences to include lotteries, as a measure of how *strongly* we prefer one prize over another.
- **Convention:** We assume $\mathcal{S}$ to be *closed under lotteries*, i.e. lotteries themselves are also states. That allows us to consider lotteries such as $[p,A;1-p,[q,B;1-q,C]]$.

SOME RIGHTS RESERVED

Rational Preferences

- **Note:** Preferences of a rational agent must obey certain constraints – An agent with *rational* preferences can be described as an MEU-agent.
- **Definition 3.6.** We call a set $\succ$ of preferences rational, iff the following constraints hold:

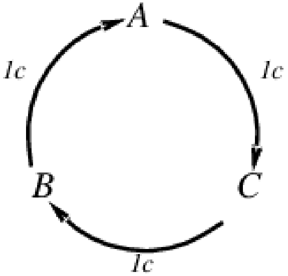
Orderability	$A\succ B \vee B\succ A \vee A\sim B$
Transitivity	$A\succ B \wedge B\succ C \Rightarrow A\succ C$
Continuity	$A \succ B \succ C \Rightarrow (\exists p.[p,A;1-p,C]\sim B)$
Substitutability	$A\sim B \Rightarrow [p,A;1-p,C]\sim[p,B;1-p,C]$
Monotonicity	$A\succ B \Rightarrow ((p > q) \Leftrightarrow [p,A;1-p,B]\succ[q,A;1-q,B])$
Decomposability	$[p,A;1-p,[q,B;1-q,C]]\sim[p,A\,;\,((1-p)q),B\,;\,((1-p)(1-q)),C]$

- From a set of rational preferences, we can obtain a meaningful utility function.

SOME RIGHTS RESERVED

Rational preferences contd.

- Violating the rationality constraints from ??? leads to self-evident irrationality.
- **Example 3.7.** An agent with intransitive preferences can be induced to give away all its money:
  - If $B \succ C$, then an agent who has C would pay (say) 1 cent to get B
  - If $A \succ B$, then an agent who has B would pay (say) 1 cent to get A
  - If $C \succ A$, then an agent who has A would pay (say) 1 cent to get C



SOME RIGHTS RESERVED

23.4 Utilities

SOME RIGHTS RESERVED

Ramseys Theorem and Value Functions

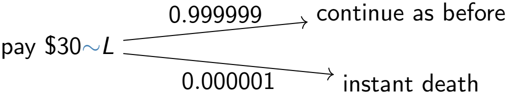
- **Theorem 4.1.** (Ramsey, 1931; von Neumann and Morgenstern, 1944) *Given a rational set of preferences there exists a real valued function U such that* $U(A) \geq U(B)$*, iff* $A \succeq B$ *and* $U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i)$
- This is an existence theorem, uniqueness not guaranteed.
- **Note:** Agent behavior is *invariant* w.r.t. positive linear transformations, i.e. an agent with utility function $U'(x) = k_1 U(x) + k_2$ where $k_1 > 0$ behaves exactly like one with U.
- **Observation:** With deterministic prizes only (no lottery choices), only a total ordering on prizes can be determined.
- **Definition 4.2.** We call a total ordering on states a value function or ordinal utility function. (If we don't need to care about *relative* utilities of states, e.g. to compute non-trivial expected utilities, that's all we need anyway!)

SOME RIGHTS RESERVED

Utilities

- **Intuition:** Utilities map states to real numbers.
- **Question:** Which numbers exactly?
- **Definition 4.3 (Standard approach to assessment of human utilities).**
  Compare a given state A to a standard lottery L_p that has
  - "best possible prize" $u_\top$ with probability p
  - "worst possible catastrophe" $u_\bot$ with probability $1-p$

  adjust lottery probability p until $A \sim L_p$. Then $U(A)=p$.
- **Example 4.4.** Choose $u_\top \mathrel{\widehat{=}}$ current state, $u_\bot \mathrel{\widehat{=}}$ instant death



SOME RIGHTS RESERVED

Popular Utility Functions

- **Definition 4.5.** Normalized utilities: $u_\top = 1$, $u_\bot = 0$.
  (Not very meaningful, but at least it's independent of the specific problem...)

SOME RIGHTS RESERVED

Popular Utility Functions

- **Definition 4.8.** Normalized utilities: $u_\top = 1$, $u_\bot = 0$.
  (Not very meaningful, but at least it's independent of the specific problem...)
- **Obviously**: *Money* (Very intuitive, often easy to determine, but actually not well-suited as a utility function (see later))

SOME RIGHTS RESERVED

Popular Utility Functions

- **Definition 4.11.** Normalized utilities: $u_\top = 1$, $u_\bot = 0$.
  (Not very meaningful, but at least it's independent of the specific problem...)
- **Obviously**: *Money* (Very intuitive, often easy to determine, but actually not well-suited as a utility function (see later))
- **Definition 4.12.** Micromorts: one millionth chance of instant death.
  (useful for Russian roulette, paying to reduce product risks, etc.)
- **But:** Not necessarily a good measure of risk, if the risk is "merely" severe injury or illness. . .

SOME RIGHTS RESERVED

Popular Utility Functions

- **Definition 4.14.** Normalized utilities: $u_\top = 1$, $u_\perp = 0$. (Not very meaningful, but at least it's independent of the specific problem...)
- **Obviously**: *Money* (Very intuitive, often easy to determine, but actually not well-suited as a utility function (see later))
- **Definition 4.15.** Micromorts: one millionth chance of instant death. (useful for Russian roulette, paying to reduce product risks, etc.)
- **But:** Not necessarily a good measure of risk, if the risk is "merely" severe injury or illness. . .
- The following measure is better (more informative)
- **Definition 4.16.** QALYs: quality adjusted life years
  QALYs are useful for medical decisions involving substantial risk.

SOME RIGHTS RESERVED

Comparing Utilities

- **Problem:** What is the monetary value of a micromort?

Comparing Utilities

- **Problem:** What is the monetary value of a micromort?
- **Just ask people:** What would you pay to avoid playing Russian roulette with a million-barrelled revolver? (Usually: quite a lot!)

SOME RIGHTS RESERVED

Comparing Utilities

- **Problem:** What is the monetary value of a micromort?
- **Just ask people:** What would you pay to avoid playing Russian roulette with a million-barrelled revolver? (Usually: quite a lot!)
- **But their behavior suggests a lower price:**
  - Driving in a car for 370km incurs a risk of one micromort;
  - Over the life of your car – say, $150,000$km that's 400 micromorts.
  - People appear to be willing to pay about $10,000$€ more for a safer car that halves the risk of death. (⇝ 25€ per micromort)

  This figure has been confirmed across many individuals and risk types.

SOME RIGHTS RESERVED

Comparing Utilities

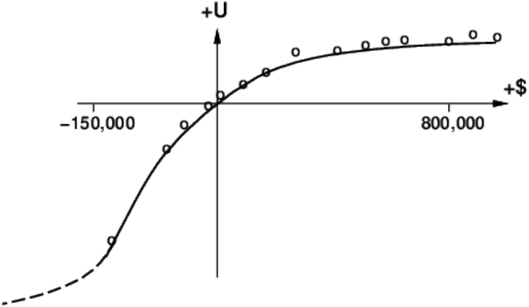
- **Problem:** What is the monetary value of a micromort?
- **Just ask people:** What would you pay to avoid playing Russian roulette with a million-barrelled revolver? (Usually: quite a lot!)
- **But their behavior suggests a lower price:**
  - Driving in a car for 370km incurs a risk of one micromort;
  - Over the life of your car – say, $150,000$km that's 400 micromorts.
  - People appear to be willing to pay about $10,000$€ more for a safer car that halves the risk of death. ($\leadsto$ 25€ per micromort)

  This figure has been confirmed across many individuals and risk types.
- Of course, this argument holds only for small risks. Most people won't agree to kill themselves for 25M€. (Also: People are pretty bad at estimating and comparing risks, especially if they are small.) (Various cognitive biases and heuristics are at work here!)

SOME RIGHTS RESERVED

Money vs. Utility

- Money does *not* behave as a utility function should.
- Given a lottery L with expected monetary value $\mathrm{EMV}(L)$, usually $U(L) < U(\mathrm{EMV}(L))$, i.e., people are risk averse.
- **Utility curve:** For what probability p am I indifferent between a prize x and a lottery $[p,M\$;1-p,0\$]$ for large numbers M?
- Typical empirical data, extrapolated with risk prone behavior for debitors:



- **Empirically:** Comes close to the logarithm on the natural numbers.

23.5 Multi-Attribute Utility

SOME RIGHTS RESERVED

Utility Functions on Attributes

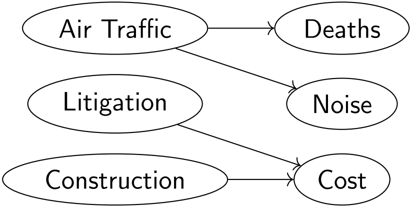
- **Recap:** So far we understand how to obtain utility functions $u\colon S \to \mathbb{R}$ on states $s \in S$ from (rational) preferences.
- **But** in practice, our actions often impact *multiple* distinct "attributes" that need to be weighed against each other.
  $\Rightarrow$ Lotteries become complex very quickly
- **Definition 5.1.** Let $X_1, \ldots, X_n$ be random variables with domains $D_1, \ldots, D_n$. Then we call a function $u\colon D_1 \times \ldots \times D_n \to \mathbb{R}$ a (multi-attribute) utility function on attributes $X_1, \ldots, X_n$.

- **Note:** In the general (worst) case, a multi-attribute utility function on n random variables with domain sizes k each requires k^n parameters to represent.
- **But:** A utility function on multiple attributes often has "internal structure" that we can exploit to simplify things.
  For example, the distinct attributes are often "independent" with respect to their utility (a higher-quality product is better than a lower-quality one that costs the same, and a cheaper product is better than an expensive one of the same quality)

SOME RIGHTS RESERVED

Multi-Attribute Utility: Example

- **Example 5.2 (Assessing an Airport Site).**



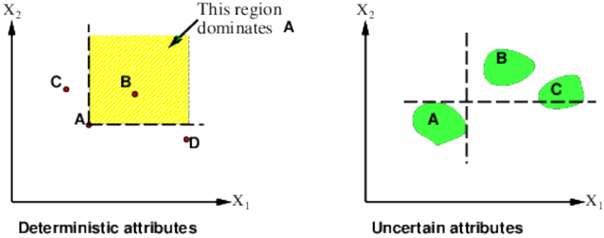
- Attributes: Deaths, Noise, Cost.
- **Question**: What is $U(\text{Deaths}, \text{Noise}, \text{Cost})$ for a projected airport?

- How can complex utility function be assessed from preference behaviour?
- **Idea 1:** Identify conditions under which decisions can be made without complete identification of $U(X_1, \ldots, X_n)$.
- **Idea 2:** Identify various types of *independence* in preferences and derive consequent canonical forms for $U(X_1, \ldots, X_n)$.

SOME RIGHTS RESERVED

Strict Dominance

- **First Assumption:** *U* is often *monotone* in each argument. (wlog. growing)
- **Definition 5.3.** *(Informally)* An action B strictly dominates an action A, iff every possible outcome of B is at least as good as every possible outcome of A,

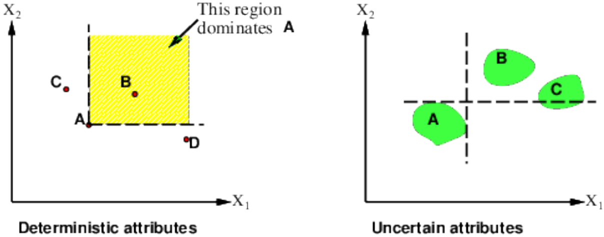


- If A strictly dominates B, we can just ignore B entirely.

SOME RIGHTS RESERVED

Strict Dominance

- **First Assumption:** U is often *monotone* in each argument. (wlog. growing)
- **Definition 5.4.** *(Informally)* An action B strictly dominates an action A, iff every possible outcome of B is at least as good as every possible outcome of A,



- If A strictly dominates B, we can just ignore B entirely.
- **Observation:** Strict dominance seldom holds in practice (life is difficult) but is useful for narrowing down the field of contenders.

SOME RIGHTS RESERVED

Stochastic Dominance

- **Definition 5.5.** Let X_1, X_2 distributions with domains $\subseteq \mathbb{R}$. X_1 stochastically dominates X_2 iff for all $t \in \mathbb{R}$, we have $P(X_1 \geq t) \geq P(X_2 \geq t)$, and for some t, we have $P(X_1 \geq t) > P(X_2 \geq t)$.
- **Observation 5.6.** *If U is monotone in X_1, and $\mathbb{P}(X_1|a)$ stochastically dominates $\mathbb{P}(X_1|b)$ for actions a, b, then a is always the better choice than b, with all other attributes X_i being equal.*
  $\Rightarrow$ If some action $\mathbb{P}(X_i|a)$ stochastically dominates $\mathbb{P}(X_i|b)$ for all attributes X_i, we can ignore b.

SOME RIGHTS RESERVED

Stochastic Dominance

- **Definition 5.8.** Let X_1, X_2 distributions with domains $\subseteq \mathbb{R}$.
  X_1 stochastically dominates X_2 iff for all $t \in \mathbb{R}$, we have $P(X_1 \geq t) \geq P(X_2 \geq t)$, and for some t, we have $P(X_1 \geq t) > P(X_2 \geq t)$.
- **Observation 5.9.** *If U is monotone in X_1, and $\mathbb{P}(X_1|a)$ stochastically dominates $\mathbb{P}(X_1|b)$ for actions a, b, then a is always the better choice than b, with all other attributes X_i being equal.*
  $\Rightarrow$ If some action $\mathbb{P}(X_i|a)$ stochastically dominates $\mathbb{P}(X_i|b)$ for all attributes X_i, we can ignore b.
- **Observation:** Stochastic dominance can often be determined without exact distributions using *qualitative* reasoning.
- **Example 5.10 (Construction cost increases with distance).** If airport location S_1 is closer to the city than $S_2 \rightsquigarrow S_1$ stochastically dominates S_2 on cost.q

SOME RIGHTS RESERVED

Preference structure: Deterministic

- **Recall:** In deterministic environments an agent has a value function.
- **Definition 5.11.** X_1 and X_2 preferentially independent of X_3 iff preference between $\langle x_1, x_2, z\rangle$ and $\langle x'_1, x'_2, z\rangle$ does not depend on z. (i.e. the tradeoff between x_1 and x_2 is independent of z)
- **Example 5.12.** E.g., $\langle \mathrm{Noise}, \mathrm{Cost}, \mathrm{Safety}\rangle$: are preferentially independent $\langle$20,000 suffer, 4.6 G\$, 0.06 deaths/mpm$\rangle$ vs. $\langle$70,000 suffer, 4.2 G\$, 0.06 deaths/mpm$\rangle$
- **Theorem 5.13 (Leontief, 1947).** *If every pair of attributes is preferentially independent of its complement, then every subset of attributes is preferentially independent of its complement: mutual preferential independence.*
- **Theorem 5.14 (Debreu, 1960).** *Mutual preferential independence implies that there is an additive value function: $V(S) = \sum_i V_i(X_i(S))$, where V_i is a value function referencing just one variable X_i.*
- Hence assess n single-attribute functions. (often a good approximation)
- **Example 5.15.** The value function for the airport decision might be

$$V(noise, cost, deaths) = -noise \cdot 10^4 - cost - deaths \cdot 10^{12}$$

SOME RIGHTS RESERVED

Preference structure: Stochastic

- **Definition 5.16.** X is utility independent of Y iff preferences over lotteries in X do not depend on particular values in Y
- **Definition 5.17.** A set X is mutually utility independent (MUI), iff each subset is utility independent of its complement.

SOME RIGHTS RESERVED

Preference structure: Stochastic

- **Definition 5.19.** X is utility independent of Y iff preferences over lotteries in X do not depend on particular values in Y
- **Definition 5.20.** A set X is mutually utility independent (MUI), iff each subset is utility independent of its complement.
- **Theorem 5.21.** *For a MUI set of attributes $\mathcal{X}$, there is a multiplicative utility function of the form:* *[**Keeney:muf74**]*

  $$U = \sum_{(\{X_0,\ldots,X_k\}\subseteq\mathcal{X})} \prod_{i=1}^{k} U_i(X_i = x_i)$$

  $\Rightarrow$ *U can be represented using n single-attribute utility functions.*
- **System Support:** Routine procedures and software packages for generating preference tests to identify various canonical families of utility functions.

SOME RIGHTS RESERVED

Decision networks - Improvements

- There are multiple ways to improve inference in decision networks:
- Exploit “inner structure” of the utility function to simplify the computation,
- eliminate dominated actions,
- label pairs of nodes with *stochastic dominance*: If (the utility of) some attribute dominates (the utility of) another attribute, focus on the dominant one (e.g. if price is always more important than quality, ignore quality whenever the price between two choices differs)
- various techniques for variable elimination,
- policy iteration (more on that when we talk about Markov decision procedures)

SOME RIGHTS RESERVED

23.6 The Value of Information

SOME RIGHTS RESERVED

What if we do not have all information we need?

- We now know how to exploit the information we have to make decisions. But if we knew more, we might be able to make even better decisions in the long run - potentially at the cost of gaining utility. (exploration vs. exploitation)
- **Example 6.1 (Medical Diagnosis).**
  - We do not expect a doctor to already know the results of the diagnostic tests when the patient comes in.
  - Tests are often expensive, and sometimes hazardous. (directly or by delaying treatment)
  - **Therefore**: Only test, if
    - knowing the results lead to a significantly better treatment plan,
    - information from test results is not drowned out by a-priori likelihood.
- **Definition 6.2.** Information value theory is concerned with agent making decisions on information gathering rationally.

SOME RIGHTS RESERVED

Value of Information by Example

- **Idea:** Compute the expected *gain in utility* from acquring information.
- **Example 6.3 (Buying Oil Drilling Rights).** There are n blocks of drilling rights available, exactly one block actually has oil worth k€, in particular:
  - The prior probability of a block having oil is $\frac{1}{n}$ each (mutually exclusive).
  - The current price of each block is $\frac{k}{n}$€.
  - A "consultant" offers an accurate survey of block (say) 3. How much should we be willing to pay for the survey?

SOME RIGHTS RESERVED

Value of Information by Example

- **Idea:** Compute the expected *gain in utility* from acquring information.
- **Example 6.5 (Buying Oil Drilling Rights).** There are n blocks of drilling rights available, exactly one block actually has oil worth k€, in particular:
  - The prior probability of a block having oil is $\frac{1}{n}$ each (mutually exclusive).
  - The current price of each block is $\frac{k}{n}$€.
  - A "consultant" offers an accurate survey of block (say) 3. How much should we be willing to pay for the survey?
- **Solution:** Compute the expected value of the best action given the information, minus the expected value of the best action without information.
- **Example 6.6 (Oil Drilling Rights contd.).**
  - Survey may say "*oil in block 3 with probability* $\frac{1}{n}$" $\leadsto$ we buy block 3 for $\frac{k}{n}$€ and make a profit of $(k-\frac{k}{n})$€.
  - Survey may say "*no oil in block 3 with probability* $\frac{n-1}{n}$" $\leadsto$ we buy another block, and make an expected profit of $\frac{k}{n-1}-\frac{k}{n}$€.
  - Without the survery, the expected profit is 0
  - Expected profit is $\frac{1}{n}\cdot\frac{(n-1)k}{n}+\frac{n-1}{n}\cdot\frac{k}{n(n-1)}=\frac{k}{n}$.
  - So, we should pay up to $\frac{k}{n}$€ for the information. (as much as block 3 is worth!)

SOME RIGHTS RESERVED

General formula (VPI)

- **Definition 6.7.** Let A the set of available actions and F a random variable. Given evidence $E_i = e_i$, let α be the action that maximizes expected utility a priori, and α_f the action that maximizes expected utility given $F = f$, i.e.: $\alpha = \underset{a \in A}{\operatorname{argmax}} \mathrm{EU}(a|E_i = e_i)$ and $\alpha_f = \underset{a \in A}{\operatorname{argmax}} \mathrm{EU}(a|E_i = e_i, F = f)$
  The value of perfect information (VPI) on F given evidence $E_i = e_i$ is defined as

  $$\mathrm{VPI}_{E_i=e_i}(F) := \Big(\sum_{f \in \mathrm{dom}(F)} P(F = f \mid E_i = e_i) \cdot \mathrm{EU}(\alpha_f|E_i = e_i, F = f)\Big) - \mathrm{EU}(\alpha|E_i =$$

- **Intuition:** The VPI is the expected gain from knowing the value of F relative to the current expected utility, and considering the relative probabilities of the possible outcomes of F.

SOME RIGHTS RESERVED

Properties of VPI

- **Observation 6.8 (VPI is Non-negative).**
  $\mathrm{VPI}_E(F) \geq 0$ *for all j and E* *(in expectation, not post hoc)*
- **Observation 6.9 (VPI is Non-additive).**
  $\mathrm{VPI}_E(F, G) \neq \mathrm{VPI}_E(F) + \mathrm{VPI}_E(G)$ *(consider, e.g., obtaining F twice)*
- **Observation 6.10 (VPI is Order-independent).**

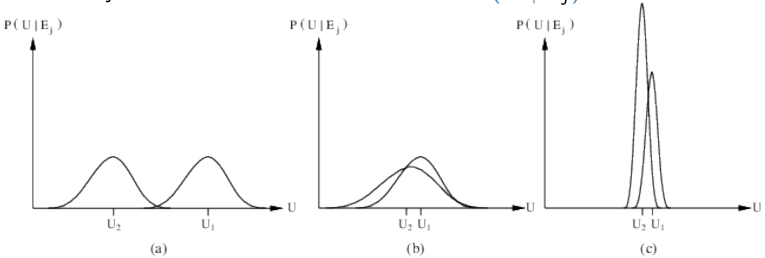
$$\mathrm{VPI}_E(F, G) = \mathrm{VPI}_E(F) + \mathrm{VPI}_{E,F}(G) = \mathrm{VPI}_E(G) + \mathrm{VPI}_{E,G}(F)$$

- **Note:** When more than one piece of evidence can be gathered, maximizing VPI for each to select one is not always optimal
  $\leadsto$ evidence-gathering becomes a sequential decision problem.

SOME RIGHTS RESERVED

Qualitative behavior of VPI

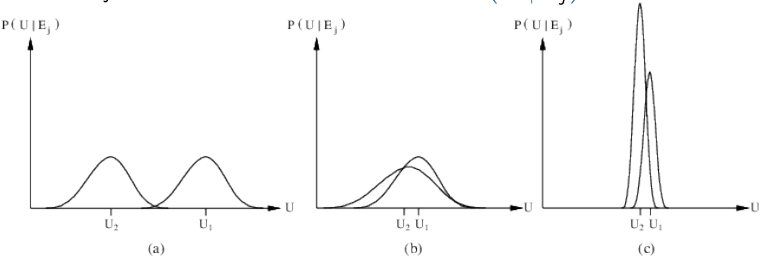
- **Question:** Say we have three distributions for $P(U \mid E_j)$



Qualitatively: What is the value of information (VPI) in these three cases?

Qualitative behavior of VPI

- **Question:** Say we have three distributions for $P(U \mid E_j)$

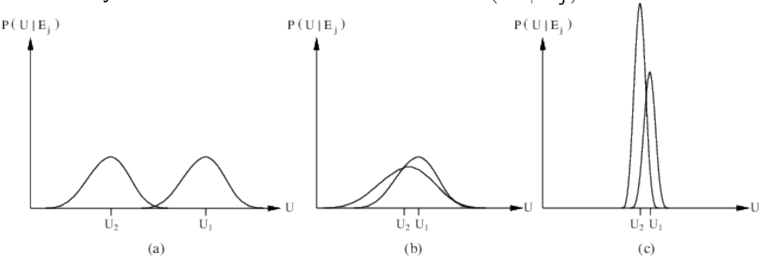


  Qualitatively: What is the value of information (VPI) in these three cases?

- **Answers:** qualitatively:
  a) Choice is obvious (a_1 almost certainly better) ⇝ information worth little

Qualitative behavior of VPI

- **Question:** Say we have three distributions for $P(U \mid E_j)$

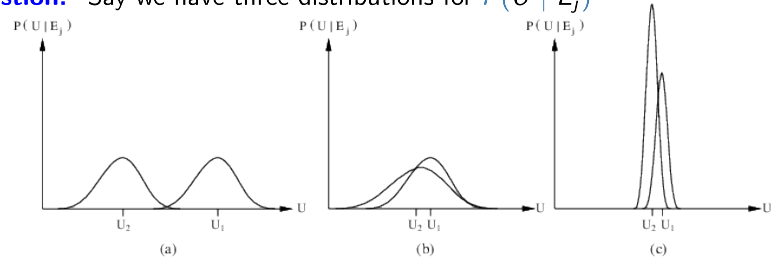


Qualitatively: What is the value of information (VPI) in these three cases?

- **Answers:** qualitatively:
  a) Choice is obvious (a_1 almost certainly better) ⇝ information worth little
  b) Choice is non-obvious (unclear) ⇝ information worth a lot

Qualitative behavior of VPI

- **Question:** Say we have three distributions for $P(U \mid E_j)$



Qualitatively: What is the value of information (VPI) in these three cases?

- **Answers:** qualitatively:
  a) Choice is obvious (a_1 almost certainly better) ⇝ information worth little
  b) Choice is non-obvious (unclear) ⇝ information worth a lot
  c) Choice is non-obvious (unclear) but makes little difference ⇝ information worth little

  Note two things
  - The difference between (b) and (c) is the width of the distribution, i.e. how close the possible outcomes are together
  - The fact that U_2 has a high peak in (c) means that its expected value is known with higher certainty than U_1. (irrelevant to the argument)

SOME RIGHTS RESERVED

A simple Information-Gathering Agent

- **Definition 6.11.** A simple information gathering agent. (gathers info before acting)

  ```
  function Information−Gathering−Agent (percept) returns an action
      persistent: $D$, a decision network
      integrate percept into $D$
      $j := \underset{k}{\operatorname{argmax}} \mathrm{VPI}_E(E_k)/Cost(E_k)$
      if $\mathrm{VPI}_E(E_j) > Cost(E_j)$ return Request($E_j$)
      else return the best action from $D$
  ```

  The next percept after Request(E_j) provides a value for E_j.

- **Problem:** The information gathering implemented here is myopic, i.e. only acquires a single evidence variable, or acts immediately. (cf. greedy search)
- But it works relatively well in practice. (e.g. outperforms humans for selecting diagnostic tests)
- Strategies for nonmyopic information gathering exist (Not discussed in this course)

Summary

- An MEU agent maximizes expected utility.
- Decision theory provides a framework for rational decision making.
- Decision networks augment Bayesian networks with action nodes and a utility node.
- rational preferences allow us to obtain a utility function (orderability, transitivity, continuity, substitutability, monotonicity, decomposability)
- multi-attribute utility functions can usually be “destructured” to allow for better inference and representation (can be monotone, attributes may dominate others, actions may dominate others, may be multiplicative,...)
- information value theory tells us when to explore rather than exploit, using
- VPI (value of perfect information) to determine how much to “pay” for information.

SOME RIGHTS RESERVED

Chapter 24
Temporal Probability Models

24.1 Modeling Time and Uncertainty

Stochastic Processes

The world changes in *stochastically predictable ways*.

Example 1.1.

- The weather changes, but the weather tomorrow is somewhat predictable *given* today's weather and other factors, (which in turn (somewhat) depends on yesterday's weather, which in turn...)
- the stock market changes, but the stock price tomorrow is probably related to today's price,
- A patient's blood sugar changes, but their blood sugar is related to their blood sugar 10 minutes ago (in particular if they didn't eat anything in between)

How do we model this?

SOME RIGHTS RESERVED

Stochastic Processes

The world changes in *stochastically predictable ways*.

Example 1.4.

- The weather changes, but the weather tomorrow is somewhat predictable *given* today's weather and other factors, (which in turn (somewhat) depends on yesterday's weather, which in turn...)
- the stock market changes, but the stock price tomorrow is probably related to today's price,
- A patient's blood sugar changes, but their blood sugar is related to their blood sugar 10 minutes ago (in particular if they didn't eat anything in between)

How do we model this?

Definition 1.5. Let $\langle\Omega, P\rangle$ a probability space and $\langle S, \preceq\rangle$ a (not necessarily *totally*) ordered set.

A sequence of random variables $(X_t)_{t\in S}$ with $\mathrm{dom}(X_t) = D$ is called a stochastic process over the time structure S.

Intuition: X_t models the outcome of the random variable X at time step t. The sample space Ω corresponds to the set of all possible sequences of outcomes.

Note: We will almost exclusively use $\langle S, \preceq\rangle = \langle\mathbb{N}, \leq\rangle$.

Definition 1.6. Given a stochastic process X_t over S and $a, b \in S$ with $a \preceq b$, we write $X_{a:b}$ for the sequence $X_a, X_{a+1}, \ldots, X_{b-1}, X_b$ and $E_{a:b}^{=e}$ for $E_a = e_a, \ldots, E_b = e_b$.

SOME RIGHTS RESERVED

Stochastic Processes (Running Example)

Example 1.7 (Umbrellas). You are a security guard in a secret underground facility, want to know it if is raining outside. Your only source of information is whether the director comes in with an umbrella.

- We have a stochastic process $\mathtt{Rain}_0, \mathtt{Rain}_1, \mathtt{Rain}_2, \ldots$ of hidden variables, and
- a related stochastic process $\mathtt{Umbrella}_0, \mathtt{Umbrella}_1, \mathtt{Umbrella}_2, \ldots$ of evidence variables.

...and a combined stochastic process $\langle \mathtt{Rain}_0, \mathtt{Umbrella}_0 \rangle, \langle \mathtt{Rain}_1, \mathtt{Umbrella}_1 \rangle, \ldots$

Note that $\mathtt{Umbrella}_t$ only depends on $\mathtt{Rain}_t$, not on e.g. $\mathtt{Umbrella}_{t-1}$ (except indirectly through $\mathtt{Rain}_t$ / $\mathtt{Rain}_{t-1}$).

Definition 1.8. We call a stochastic process of *hidden* variables a state variable.

SOME RIGHTS RESERVED

Markov Processes

Idea: Construct a Bayesian network from these variables (parents?)
...without everything exploding in size...?

Markov Processes

Idea: Construct a Bayesian network from these variables (parents?)
...without everything exploding in size...?

Definition 1.11. Let $(X_t)_{t \in S}$ a stochastic process. X has the (nth order) Markov property iff X_t only depends on a bounded subset of $\mathrm{X}_{0:t-1}$ – i.e. for all $t \in S$ we have $\mathbb{P}(X_t|X_0, \ldots X_{t-1}) = \mathbb{P}(X_t|X_{t-n}, \ldots X_{t-1})$ for some $n \in S$.
A stochastic process with the Markov property for some n is called a (nth order) Markov process.

SOME RIGHTS RESERVED

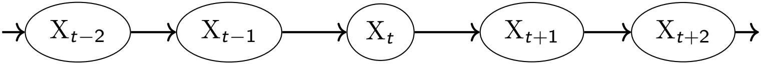
Markov Processes

Idea: Construct a Bayesian network from these variables (parents?)
...without everything exploding in size...?

Definition 1.13. Let $(X_t)_{t \in S}$ a stochastic process. X has the (nth order) Markov property iff X_t only depends on a bounded subset of $\mathrm{X}_{0:t-1}$ – i.e. for all $t \in S$ we have $\mathbb{P}(X_t | X_0, \ldots X_{t-1}) = \mathbb{P}(X_t | X_{t-n}, \ldots X_{t-1})$ for some $n \in S$.
A stochastic process with the Markov property for some n is called a (nth order) Markov process.

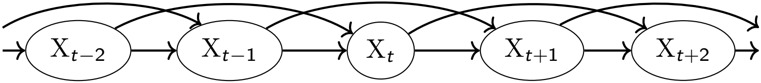
Important special cases:
Definition 1.14.

- First-order Markov property: $\mathbb{P}(\mathrm{X}_t | \mathrm{X}_{0:t-1}) = \mathbb{P}(\mathrm{X}_t | \mathrm{X}_{t-1})$



  A first order Markov process is called a Markov chain.

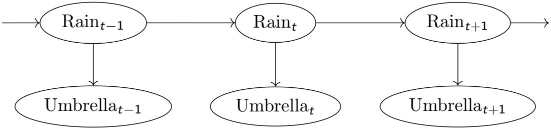
- Second-order Markov property: $\mathbb{P}(\mathrm{X}_t | \mathrm{X}_{0:t-1}) = \mathbb{P}(\mathrm{X}_t | \mathrm{X}_{t-2}, \mathrm{X}_{t-1})$



SOME RIGHTS RESERVED

Markov Process Example: The Umbrella

Example 1.15 (Umbrellas continued). We model the situation in a Bayesian network:

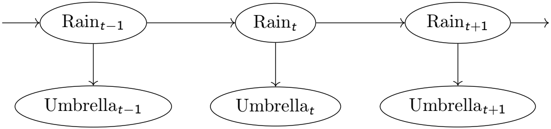


Problem: This network does not actually have the First-order Markov property...

SOME RIGHTS RESERVED

Markov Process Example: The Umbrella

Example 1.16 (Umbrellas continued). We model the situation in a Bayesian network:



Problem: This network does not actually have the First-order Markov property...

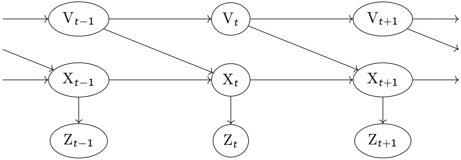
Possible fixes: We have two ways to fix this:

1. Increase the order of the Markov process. (more dependencies ⇒ more complex inference)
2. Add more state variables, e.g., Temp_t, $\mathrm{Pressure}_t$. (more information sources)

SOME RIGHTS RESERVED

Markov Process Example: Robot Motion

Example 1.17 (Random Robot Motion). Assume we want to track a robot wandering randomly on the X/Y plane, whose position we can only observe roughly (e.g. by approximate GPS coordinates:) Markov chain



- the velocity V_i may change unpredictably.
- the exact position X_i depends on previous position X_{i-1} and velocity V_{i-1}
- the position X_i influences the observed position Z_i.

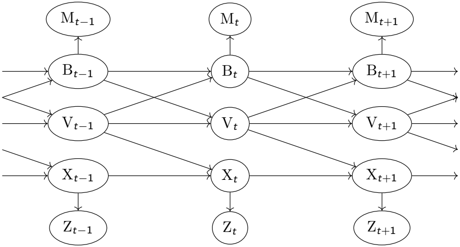
Example 1.18 (Battery Powered Robot). If the robot has a *battery*, the Markov property is violated!

- Battery exhaustion has a systematic effect on the change in velocity.
- This depends on how much power was used by all previous manoeuvres.

Markov Process Example: Robot Motion

Idea: We can restore the Markov property by including a state variable for the charge level B_t. (Better still: Battery level sensor)

Example 1.19 (Battery Powered Robot Motion).



- Battery level B_i is influenced by previous level $B_{i-1}$and velocity V_{i-1}.
- Velocity V_i is influenced by previous level $B_{i-1}$and velocity V_{i-1}.
- Battery meter M_i is only influenced by Battery level B_i.

SOME RIGHTS RESERVED

Stationary Markov Processes as Transition Models

Remark 1.20. Given a stochastic process with state variables X_t and evidence variables E_t, then $\mathbb{P}(X_t|\mathrm{X}_{0:t})$ is a transition model and $\mathbb{P}(E_t|\mathrm{X}_{0:t},\mathrm{E}_{1:t-1})$ a sensor model in the sense of a model-based agent.
Note that we assume that the X_t do not depend on the E_t.
Also note that with the Markov property, the transition model simplifies to $\mathbb{P}(\mathrm{X}_t|\mathrm{X}_{t-n})$.

Problem: Even with the Markov property the transition model is infinite. ($t \in \mathbb{N}$)

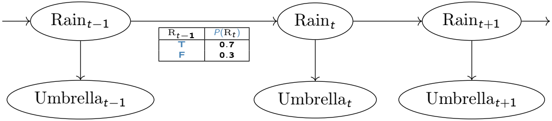
SOME RIGHTS RESERVED

Stationary Markov Processes as Transition Models

Remark 1.23. Given a stochastic process with state variables X_t and evidence variables E_t, then $\mathbb{P}(X_t|\mathrm{X}_{0:t})$ is a transition model and $\mathbb{P}(E_t|\mathrm{X}_{0:t},\mathrm{E}_{1:t-1})$ a sensor model in the sense of a model-based agent.
Note that we assume that the X_t do not depend on the E_t.
Also note that with the Markov property, the transition model simplifies to $\mathbb{P}(\mathrm{X}_t|\mathrm{X}_{t-n})$.

Problem: Even with the Markov property the transition model is infinite. ($t \in \mathbb{N}$)
Definition 1.24. A Markov chain is called stationary if the transition model is independent of time, i.e. $\mathbb{P}(X_t|X_{t-1})$ is the same for all t.

Example 1.25 (Umbrellas are stationary). $\mathbb{P}(\mathtt{Rain}_t|\mathtt{Rain}_{t-1})$ does not depend on t. (need only one table)



R_{t-1}	$P(\mathrm{R}_t)$
T	0.7
F	0.3

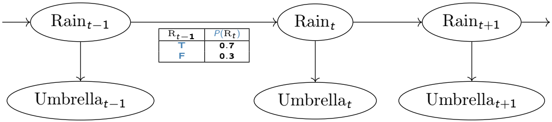
SOME RIGHTS RESERVED

Stationary Markov Processes as Transition Models

Remark 1.26. Given a stochastic process with state variables X_t and evidence variables E_t, then $\mathbb{P}(X_t|\mathrm{X}_{0:t})$ is a transition model and $\mathbb{P}(E_t|\mathrm{X}_{0:t},\mathrm{E}_{1:t-1})$ a sensor model in the sense of a model-based agent.
Note that we assume that the X_t do not depend on the E_t.
Also note that with the Markov property, the transition model simplifies to $\mathbb{P}(\mathrm{X}_t|\mathrm{X}_{t-n})$.

Problem: Even with the Markov property the transition model is infinite. ($t \in \mathbb{N}$)
Definition 1.27. A Markov chain is called stationary if the transition model is independent of time, i.e. $\mathbb{P}(X_t|X_{t-1})$ is the same for all t.

Example 1.28 (Umbrellas are stationary). $\mathbb{P}(\mathtt{Rain}_t|\mathtt{Rain}_{t-1})$ does not depend on t. (need only one table)



R_{t-1}	$P(\mathrm{R}_t)$
T	0.7
F	0.3

⚠ Don't confuse "stationary" (Markov processes) with "static" (environments).
We restrict ourselves to stationary Markov processes in AI-2.

SOME RIGHTS RESERVED

Markov Sensor Models

Recap: The sensor model $\mathbb{P}(E_t|\mathrm{X}_{0:t},\mathrm{E}_{1:t-1})$ allows us (using Bayes rule et al) to update our belief state about X_t given the observations $\mathrm{E}_{0:t}$.

Problem: The evidence variables E_t could depend on any of the variables $\mathrm{X}_{0:t},\mathrm{E}_{1:t-1}\ldots$

SOME RIGHTS RESERVED

Markov Sensor Models

Recap: The sensor model $\mathbb{P}(E_t|\mathrm{X}_{0:t}, \mathrm{E}_{1:t-1})$ allows us (using Bayes rule et al) to update our belief state about X_t given the observations $\mathrm{E}_{0:t}$.

Problem: The evidence variables E_t could depend on any of the variables $\mathrm{X}_{0:t}, \mathrm{E}_{1:t-1}\ldots$

Definition 1.31. We say that a sensor model has the sensor Markov property, iff $\mathbb{P}(E_t|\mathrm{X}_{0:t}, \mathrm{E}_{1:t-1}) = \mathbb{P}(E_t|X_t)$ – i.e., the sensor model depends only on the current state.

Assumptions on Sensor Models: We usually assume the sensor Markov property and make it stationary as well: $\mathbb{P}(E_t|X_t)$ is fixed for all t.

SOME RIGHTS RESERVED

Markov Sensor Models

Recap: The sensor model $\mathbb{P}(E_t|\mathrm{X}_{0:t},\mathrm{E}_{1:t-1})$ allows us (using Bayes rule et al) to update our belief state about X_t given the observations $\mathrm{E}_{0:t}$.

Problem: The evidence variables E_t could depend on any of the variables $\mathrm{X}_{0:t},\mathrm{E}_{1:t-1}\ldots$

Definition 1.33. We say that a sensor model has the sensor Markov property, iff $\mathbb{P}(E_t|\mathrm{X}_{0:t},\mathrm{E}_{1:t-1})=\mathbb{P}(E_t|X_t)$ – i.e., the sensor model depends only on the current state.

Assumptions on Sensor Models: We usually assume the sensor Markov property and make it stationary as well: $\mathbb{P}(E_t|X_t)$ is fixed for all t.

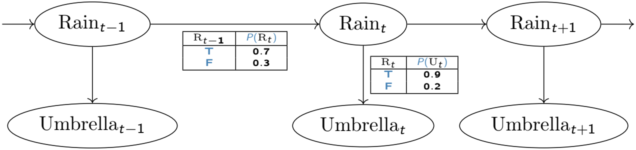
Definition 1.34 (Note).

- If a Markov chain X is stationary and discrete, we can represent the transition model as a matrix $\mathrm{T}_{ij}:=P(X_t=j\mid X_{t-1}=i)$.
- If a sensor model has the sensor Markov property, we can represent each observation $E_t=e_t$ at time t as the diagonal matrix O_t with $\mathrm{O}_{t\,ii}:=P(E_t=e_t\mid X_t=i)$.
- A pair $\langle X,E\rangle$ where X is a (stationary) Markov chains, E_i only depends on X_i, and E has the sensor Markov property is called a (stationary) Hidden Markov Model (HMM). (X and E are single variables)

SOME RIGHTS RESERVED

Umbrellas, the full Story

Example 1.35 (Umbrellas, Transition & Sensor Models).



R_{t-1}	$P(\mathrm{R}_t)$
T	0.7
F	0.3

R_{t}	$P(\mathrm{U}_t)$
T	0.9
F	0.2

This is a hidden Markov model

Observation 1.36. *If we know the initial prior probabilities* $\mathbb{P}(X_0)$ *(*$\widehat{=}$ *time* $t=0$*), then we can compute the full joint probability distribution as*

$$\mathbb{P}(\mathrm{X}_{0:t},\mathrm{E}_{1:t})=\mathbb{P}(X_0)\cdot(\prod_{i=1}^{t}\mathbb{P}(X_i|X_{i-1})\cdot\mathbb{P}(E_i|X_i))$$

SOME RIGHTS RESERVED

24.2 Inference: Filtering, Prediction, and Smoothing

SOME RIGHTS RESERVED

Inference tasks

Definition 2.1. Given a Markov process with state variables X_t and evidence variables E_t, we are interested in the following Markov inference tasks:

- Filtering (or monitoring) $\mathbb{P}(X_t|E_{1:t}^{=e})$: Given the sequence of observations up until time t, compute the likely state of the world at *current* time t.
- Prediction (or state estimation) $\mathbb{P}(X_{t+k}|E_{1:t}^{=e})$ for $k > 0$: Given the sequence of observations up until time t, compute the likely *future* state of the world at time $t + k$.
- Smoothing (or hindsight) $\mathbb{P}(X_{t-k}|E_{1:t}^{=e})$ for $0 < k < t$: Given the sequence of observations up until time t, compute the likely *past* state of the world at time $t - k$.
- Most likely explanation $\underset{x_{1:t}}{\operatorname{argmax}}\left(P(X_{1:t}^{=x} \mid E_{1:t}^{=e})\right)$: Given the sequence of observations up until time t, compute the most likely sequence of states that led to these observations.

  Note: The most likely sequence of states is *not* (necessarily) the sequence of most likely states ;-)

In this section, we assume X and E to represent *multiple* variables, where X jointly forms a Markov chain and the E jointly have the sensor Markov property.

In the case where X and E are stationary *single* variables, we have a stationary hidden Markov model and can use the matrix forms.

SOME RIGHTS RESERVED

Filtering (Computing the Belief State given Evidence)

Note:

- Using the full joint probability distribution, we can compute any conditional probability we want, but not necessarily efficiently.
- We want to use filtering to update our "world model" $\mathbb{P}(X_t)$ based on a new observation $E_t = e_t$ and our *previous* world model $\mathbb{P}(X_{t-1})$.

$\Rightarrow$ We want a function $\mathbb{P}(X_t|E_{1:t}^{=e}) = F(e_t, \underbrace{\mathbb{P}(X_{t-1}|E_{1:t-1}^{=e})}_{F(e_{t-\mathbf{1}},\ldots)})$

Filtering (Computing the Belief State given Evidence)

Note:

- Using the full joint probability distribution, we can compute any conditional probability we want, but not necessarily efficiently.
- We want to use filtering to update our "world model" $\mathbb{P}(X_t)$ based on a new observation $E_t = e_t$ and our *previous* world model $\mathbb{P}(X_{t-1})$.

$\Rightarrow$ We want a function $\mathbb{P}(X_t | E_{1:t}^{=e}) = F(e_t, \underbrace{\mathbb{P}(X_{t-1} | E_{1:t-1}^{=e})}_{F(e_{t-1}, \ldots)})$

Spoiler:

$$F(e_t, \mathbb{P}(X_{t-1} | E_{1:t-1}^{=e})) = \alpha(\mathrm{O}_t \cdot \mathrm{T}^T \cdot \mathbb{P}(X_{t-1} | E_{1:t-1}^{=e}))$$

Filtering Derivation

$$\begin{aligned}
\mathbb{P}(X_t|E_{1:t}^{=e}) &= \mathbb{P}(X_t|E_t = e_t, E_{1:t-1}^{=e}) && \text{(div} \\
&= \alpha(\mathbb{P}(E_t = e_t|X_t, E_{1:t-1}^{=e}) \cdot \mathbb{P}(X_t|E_{1:t-1}^{=e})) && \text{(usi} \\
&= \alpha(\mathbb{P}(E_t = e_t|X_t) \cdot \mathbb{P}(X_t|E_{1:t-1}^{=e})) && \text{(sen} \\
&= \alpha(\mathbb{P}(E_t = e_t|X_t) \cdot (\sum_{x\in\mathrm{dom}(X)} \mathbb{P}(X_t|X_{t-1} = x, E_{1:t-1}^{=e}) \cdot P(X_{t-1} = x \mid E_{1:t-1}^{=e})) \\
&= \alpha(\underbrace{\mathbb{P}(E_t = e_t|X_t)}_{\text{sensor model}} \cdot (\sum_{x\in\mathrm{dom}(X)} \underbrace{\mathbb{P}(X_t|X_{t-1} = x)}_{\text{transition model}} \cdot \underbrace{P(X_{t-1} = x \mid E_{1:t-1}^{=e})))}_{\text{recursive call}} && \text{(cor}
\end{aligned}$$

SOME RIGHTS RESERVED

Filtering Derivation

$$\begin{aligned}
\mathbb{P}(X_t|E_{1:t}^{=e}) &= \mathbb{P}(X_t|E_t = e_t, E_{1:t-1}^{=e}) && \text{(div} \\
&= \alpha(\mathbb{P}(E_t = e_t|X_t, E_{1:t-1}^{=e}) \cdot \mathbb{P}(X_t|E_{1:t-1}^{=e})) && \text{(usi} \\
&= \alpha(\mathbb{P}(E_t = e_t|X_t) \cdot \mathbb{P}(X_t|E_{1:t-1}^{=e})) && \text{(sen} \\
&= \alpha(\mathbb{P}(E_t = e_t|X_t) \cdot (\sum_{x \in \mathrm{dom}(X)} \mathbb{P}(X_t|X_{t-1} = x, E_{1:t-1}^{=e}) \cdot P(X_{t-1} = x \mid E_{1:t-1}^{=e})) \\
&= \alpha(\underbrace{\mathbb{P}(E_t = e_t|X_t)}_{\text{sensor model}} \cdot (\sum_{x \in \mathrm{dom}(X)} \underbrace{\mathbb{P}(X_t|X_{t-1} = x)}_{\text{transition model}} \cdot \underbrace{P(X_{t-1} = x \mid E_{1:t-1}^{=e})))}_{\text{recursive call}} && \text{(cor}
\end{aligned}$$

Reminder: In a stationary HMM, we have the matrices $\mathsf{T}_{ij} = P(X_t = j \mid X_{t-1} = i)$ and $\mathsf{O}_{t\,ii} = P(E_t = e_t \mid X_t = i)$.
Then interpreting $\mathbb{P}(X_{t-1}|E_{1:t-1}^{=e})$ as a vector, the above corresponds exactly to the matrix multiplication $\alpha(\mathsf{O}_t \cdot \mathsf{T}^T \cdot \mathbb{P}(X_{t-1}|E_{1:t-1}^{=e}))$

Definition 2.3. We call the inner part of the above expression the forward algorithm, i.e. $\mathbb{P}(X_t|E_{1:t}^{=e}) = \alpha(\mathrm{FORWARD}(e_t, \mathbb{P}(X_{t-1}|E_{1:t-1}^{=e}))) =: \mathsf{f}_{1:t}$.

SOME RIGHTS RESERVED

Filtering the Umbrellas

Example 2.4. Let's assume:

- $\mathbb{P}(\mathtt{R}_0) = \langle 0.5, 0.5 \rangle$, (Note that with growing t (and evidence), the impact of the prior at $t = 0$ vanishes anyway)
- $P(\mathtt{R}_{t+1} \mid \mathtt{R}_t) = 0.6$, $P(\neg\mathtt{R}_{t+1} \mid \neg\mathtt{R}_t) = 0.8$, $P(\mathtt{U}_t \mid \mathtt{R}_t) = 0.9$ and $P(\neg\mathtt{U}_t \mid \neg\mathtt{R}_t) = 0.85$

$$\Rightarrow \mathsf{T} = \begin{pmatrix} 0.6 & 0.4 \\ 0.2 & 0.8 \end{pmatrix}$$

- The director carries an umbrella on days 1 and 2, and *not* on day 3.

$$\Rightarrow \mathsf{O}_1 = \mathsf{O}_2 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.15 \end{pmatrix} \text{ and } \mathsf{O}_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.85 \end{pmatrix}.$$

Then:

- $\mathsf{f}_{1:1} := \mathbb{P}(\mathtt{R}_1|\mathtt{U}_1 = \mathsf{T}) = \alpha(\mathbb{P}(\mathtt{U}_1 = \mathsf{T}|\mathtt{R}_1) \cdot (\sum_{b \in \{\mathsf{T},\mathsf{F}\}} \mathbb{P}(\mathtt{R}_1|\mathtt{R}_0 = b) \cdot P(\mathtt{R}_0 = b)))$

$$= \alpha(\langle 0.9, 0.15 \rangle \cdot (\langle 0.6, 0.4 \rangle \cdot 0.5 + \langle 0.2, 0.8 \rangle \cdot 0.5)) = \alpha(\langle 0.36, 0.09 \rangle) = \langle 0.8, 0.2 \rangle$$

- Using matrices:

$$\alpha(\mathsf{O}_1 \cdot \mathsf{T}^T \cdot \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}) = \alpha(\begin{pmatrix} 0.9 & 0 \\ 0 & 0.15 \end{pmatrix} \cdot \begin{pmatrix} 0.6 & 0.2 \\ 0.4 & 0.8 \end{pmatrix} \cdot \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix})$$

$$= \alpha(\begin{pmatrix} 0.9 \cdot 0.6 & 0.9 \cdot 0.2 \\ 0.15 \cdot 0.4 & 0.15 \cdot 0.8 \end{pmatrix} \cdot \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}) =$$

$$= \alpha(\begin{pmatrix} 0.9 \cdot 0.6 \cdot 0.5 + 0.9 \cdot 0.2 \cdot 0.5 \\ 0.15 \cdot 0.4 \cdot 0.5 + 0.15 \cdot 0.8 \cdot 0.5 \end{pmatrix}) = \alpha(\begin{pmatrix} 0.36 \\ 0.09 \end{pmatrix})$$

SOME RIGHTS RESERVED

Filtering the Umbrellas (Continued)

Example 2.5. $f_{1:1} := \mathbb{P}(R_1|U_1 = T) = \langle 0.8, 0.2\rangle$

- $f_{1:2} := \mathbb{P}(R_2|U_2 = T, U_1 = T) = \alpha(O_2 \cdot T^T \cdot f_{1:1}) = \alpha(\mathbb{P}(U_2 = T|R_2) \cdot (\sum_{b\in\{T,F\}} \mathbb{P}(R_2|R_1 = b) \cdot f_{1:1}(b)))$
  $=\alpha(\langle 0.9, 0.15\rangle \cdot (\langle 0.6, 0.4\rangle \cdot 0.8 + \langle 0.2, 0.8\rangle \cdot 0.2)) = \alpha(\langle 0.468, 0.072\rangle) = \langle 0.87, 0.13\rangle$
- $f_{1:3} := \mathbb{P}(R_3|U_3 = F, U_2 = T, U_1 = T) = \alpha(O_3 \cdot T^T \cdot f_{1:2})$
  $=\alpha(\mathbb{P}(U_3 = F|R_3) \cdot (\sum_{b\in\{T,F\}} \mathbb{P}(R_3|R_2 = b) \cdot f_{1:2}(b)))$
  $=\alpha(\langle 0.1, 0.85\rangle \cdot (\langle 0.6, 0.4\rangle \cdot 0.87 + \langle 0.2, 0.8\rangle \cdot 0.13)) = \alpha(\langle 0.0547, 0.3853\rangle) = \langle 0.12, 0.88\rangle$

SOME RIGHTS RESERVED

Prediction in Markov Chains

Prediction: $\mathbb{P}(X_{t+k}|E_{1:t}^{=e})$ for $k > 0$.

Intuition: Prediction is filtering without new evidence – i.e. we can use filtering until t, and then continue as follows:

Lemma 2.6. *By the same reasoning as filtering:*

$$\mathbb{P}(X_{t+k+1}|E_{1:t}^{=e}) = \sum_{x\in\mathrm{dom}(X)} \underbrace{\mathbb{P}(X_{t+k+1}|X_{t+k}=x)}_{\textit{transition model}} \cdot \underbrace{P(X_{t+k}=x \mid E_{1:t}^{=e})}_{\textit{recursive call}} \underbrace{=\mathrm{T}^{T} \cdot \mathbb{P}(X_{t+k}}_{\textit{HMM}}$$

Prediction in Markov Chains

Prediction: $\mathbb{P}(X_{t+k}|E_{1:t}^{=e})$ for $k > 0$.

Intuition: Prediction is filtering without new evidence – i.e. we can use filtering until t, and then continue as follows:

Lemma 2.8. *By the same reasoning as filtering:*

$$\mathbb{P}(X_{t+k+1}|E_{1:t}^{=e}) = \sum_{x\in\mathrm{dom}(X)} \underbrace{\mathbb{P}(X_{t+k+1}|X_{t+k}=x)}_{\textit{transition model}} \cdot \underbrace{P(X_{t+k}=x \mid E_{1:t}^{=e})}_{\textit{recursive call}} \underbrace{= \mathrm{T}^{T} \cdot \mathbb{P}(X_{t+k}}_{\textit{HMM}}$$

Observation 2.9. *As $k \to \infty$, $\mathbb{P}(X_{t+k}|E_{1:t}^{=e})$ converges towards a fixed point called the stationary distribution of the Markov chain.* (*which we can compute from the equation $S = \mathrm{T}^{T} \cdot S$*)

⇝ the impact of the evidence vanishes.

⇝ The stationary distribution only depends on the transition model.

⇝ There is a small window of time (depending on the transition model) where the evidence has enough impact to allow for prediction beyond the mere stationary distribution, called the mixing time of the Markov chain.

⇝ Predicting the future is difficult, and the further into the future, the more difficult it is (*Who knew...*)

SOME RIGHTS RESERVED

Smoothing

Smoothing: $\mathbb{P}(X_{t-k}|E_{1:t}^{=e})$ for $k > 0$.

Intuition: Use filtering to compute $\mathbb{P}(X_t|E_{1:t-k}^{=e})$, then recurse *backwards* from t until $t-k$.

$$\begin{aligned}
\mathbb{P}(X_{t-k}|E_{1:t}^{=e}) &= \mathbb{P}(X_{t-k}|E_{t-(k-1):t}^{=e}, E_{1:t-k}^{=e}) && \text{(Divide the ev} \\
&= \alpha(\mathbb{P}(E_{t-(k-1):t}^{=e}|X_{t-k}, E_{1:t-k}^{=e}) \cdot \mathbb{P}(X_{t-k}|E_{1:t-k}^{=e})) && \text{(Bayes Rule)} \\
&= \alpha(\underbrace{\mathbb{P}(E_{t-(k-1):t}^{=e}|X_{t-k})}_{=:\mathrm{b}_{t-(k-\mathbf{1}):t}} \cdot \underbrace{\mathbb{P}(X_{t-k}|E_{1:t-k}^{=e})}_{=\mathrm{f}_{\mathbf{1}:t-k}}) && \text{(cond. indepe} \\
&= \alpha(\mathrm{f}_{1:t-k} \times \mathrm{b}_{t-(k-1):t})
\end{aligned}$$

(where × denotes component-wise multiplication)

Smoothing (continued)

Definition 2.10 (Backward message). $\mathrm{b}_{t-k:t} = \mathbb{P}(E^{=e}_{t-k:t} | X_{t-(k+1)})$

$$= \sum_{x \in \mathrm{dom}(X)} \mathbb{P}(E^{=e}_{t-k:t} | X_{t-k} = x, X_{t-(k+1)}) \cdot \mathbb{P}(X_{t-k} = x | X_{t-(k+1)})$$

$$= \sum_{x \in \mathrm{dom}(X)} P(E^{=e}_{t-k:t} \mid X_{t-k} = x) \cdot \mathbb{P}(X_{t-k} = x | X_{t-(k+1)})$$

$$= \sum_{x \in \mathrm{dom}(X)} P(E_{t-k} = e_{t-k}, E^{=e}_{t-(k-1):t} \mid X_{t-k} = x) \cdot \mathbb{P}(X_{t-k} = x | X_{t-(k+1)})$$

$$= \sum_{x \in \mathrm{dom}(X)} \underbrace{P(E_{t-k} = e_{t-k} \mid X_{t-k} = x)}_{\text{sensor model}} \cdot \underbrace{P(E^{=e}_{t-(k-1):t} \mid X_{t-k} = x)}_{=\mathrm{b}_{t-(k-\mathbf{1}):t}} \cdot \underbrace{\mathbb{P}(X_{t-k} = x}_{\text{transition}}$$

Note: in a stationary hidden Markov model, we get the matrix formulation $\mathrm{b}_{t-k:t} = \mathrm{T} \cdot \mathrm{O}_{t-k} \cdot \mathrm{b}_{t-(k-1):t}$

Smoothing (continued)

Definition 2.12 (Backward message). $b_{t-k:t} = \mathbb{P}(E^{=e}_{t-k:t} | X_{t-(k+1)})$

$$= \sum_{x \in \mathrm{dom}(X)} \mathbb{P}(E^{=e}_{t-k:t} | X_{t-k} = x, X_{t-(k+1)}) \cdot \mathbb{P}(X_{t-k} = x | X_{t-(k+1)})$$

$$= \sum_{x \in \mathrm{dom}(X)} P(E^{=e}_{t-k:t} \mid X_{t-k} = x) \cdot \mathbb{P}(X_{t-k} = x | X_{t-(k+1)})$$

$$= \sum_{x \in \mathrm{dom}(X)} P(E_{t-k} = e_{t-k}, E^{=e}_{t-(k-1):t} \mid X_{t-k} = x) \cdot \mathbb{P}(X_{t-k} = x | X_{t-(k+1)})$$

$$= \sum_{x \in \mathrm{dom}(X)} \underbrace{P(E_{t-k} = e_{t-k} \mid X_{t-k} = x)}_{\text{sensor model}} \cdot \underbrace{P(E^{=e}_{t-(k-1):t} \mid X_{t-k} = x)}_{=b_{t-(k-\mathbf{1}):t}} \cdot \underbrace{\mathbb{P}(X_{t-k} = x}_{\text{transition}}$$

Note: in a stationary hidden Markov model, we get the matrix formulation $b_{t-k:t} = T \cdot O_{t-k} \cdot b_{t-(k-1):t}$

Definition 2.13. We call the associated algorithm the backward algorithm, i.e. $\mathbb{P}(X_{t-k} | E^{=e}_{1:t}) =$
$\alpha(\underbrace{\mathrm{FORWARD}(e_{t-k}, f_{1:t-(k+1)})}_{f_{\mathbf{1}:t-k}} \times \underbrace{\mathrm{BACKWARD}(e_{t-(k-1)}, b_{t-(k-2):t})}_{b_{t-(k-\mathbf{1}):t}})$.

As a starting point for the recursion, we let $b_{t+1:t}$ the uniform vector with 1 in every component.

SOME RIGHTS RESERVED

Smoothing example

Example 2.14 (Smoothing Umbrellas). Reminder: We assumed $\mathbb{P}(R_0) = \langle 0.5, 0.5\rangle$, $P(R_{t+1} \mid R_t) = 0.6$, $P(\neg R_{t+1} \mid \neg R_t) = 0.8$, $P(U_t \mid R_t) = 0.9$, $P(\neg U_t \mid \neg R_t) = 0.85$

$\Rightarrow T = \begin{pmatrix} 0.6 & 0.4 \\ 0.2 & 0.8 \end{pmatrix}$, $O_1 = O_2 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.15 \end{pmatrix}$ and $O_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.85 \end{pmatrix}$.

(The director carries an umbrella on days 1 and 2, and *not* on day 3)

$f_{1:1} = \langle 0.8, 0.2\rangle$, $f_{1:2} = \langle 0.87, 0.13\rangle$ and $f_{1:3} = \langle 0.12, 0.88\rangle$

Let's compute

$$\mathbb{P}(R_1 | U_1 = T, U_2 = T, U_3 = F) = \alpha(f_{1:1} \times b_{2:3})$$

- We need to compute $b_{2:3}$ and $b_{3:3}$:
- $b_{3:3} = T \cdot O_3 \cdot b_{4:3} = \begin{pmatrix} 0.6 & 0.4 \\ 0.2 & 0.8 \end{pmatrix} \cdot \begin{pmatrix} 0.1 & 0 \\ 0 & 0.85 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.4 \\ 0.7 \end{pmatrix}$
- $b_{2:3} = T \cdot O_2 \cdot b_{3:3} = \begin{pmatrix} 0.6 & 0.4 \\ 0.2 & 0.8 \end{pmatrix} \cdot \begin{pmatrix} 0.9 & 0 \\ 0 & 0.15 \end{pmatrix} \cdot \begin{pmatrix} 0.4 \\ 0.7 \end{pmatrix} = \begin{pmatrix} 0.258 \\ 0.156 \end{pmatrix}$

$$\Rightarrow \alpha(\begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix} \times \begin{pmatrix} 0.258 \\ 0.156 \end{pmatrix}) = \alpha(\begin{pmatrix} 0.2064 \\ 0.0312 \end{pmatrix}) = \begin{pmatrix} 0.87 \\ 0.13 \end{pmatrix}$$

$\Rightarrow$ Given the evidence $U_2, \neg U_3$, the posterior probability for R_1 went up from 0.8 to 0.87!

SOME RIGHTS RESERVED

Forward/Backward Algorithm for Smoothing

Definition 2.15. Forward backward algorithm: returns the sequence of posterior distributions $\mathbb{P}(X_1)\ldots\mathbb{P}(X_t)$ given evidence $e_1, \ldots, e_t$:

```
function FORWARD-BACKWARD($\langle e_1, \ldots, e_t\rangle$,$\mathbb{P}(X_0)$)
    $f := \langle\mathbb{P}(X_0)\rangle$
    $b := \langle 1, 1, \ldots\rangle$
    $S := \langle\mathbb{P}(X_0)\rangle$
    for $i = 1, \ldots, t$ do
        $f_i :=$ FORWARD($f_{i-1}, e_i$)                    /* filtering */
    for $i = t, \ldots, 1$ do
        $S_i := \alpha(f_i \times b)$                        /* smoothing */
        $b :=$ BACKWARD($b, e_i$)
    return $S$
```

Time complexity linear in t (polytree inference), Space complexity $\mathcal{O}(t \cdot |\mathrm{f}|)$.

Country dance algorithm

Idea: If T and O_i are invertible, we can avoid storing all forward messages in the smoothing algorithm by running filtering backwards:

$$f_{1:i+1} = \alpha(O_{i+1} \cdot T^T \cdot f_{1:i})$$

$$\Rightarrow f_{1:i} = \alpha({T^T}^{-1} \cdot {O_{i+1}}^{-1} \cdot f_{1:i+1})$$

⇒ we can trade space complexity for time complexity:

- In the first for-loop, we only compute the final $f_{1:t}$ (No need to store the intermediate results)
- In the second for-loop, we compute both $f_{1:i}$ and $b_{t-i:t}$ (Only one copy of $f_{1:i}$, $b_{t-i:t}$ is stored)

⇒ constant space.

But: Requires that both matrices are invertible, i.e. *every observation must be possible in every state*. (Possible hack: increase the probabilities of 0 to "negligibly small")

SOME RIGHTS RESERVED

Most Likely Explanation

Smoothing allows us to compute the *sequence of most likely states* $X_1, \ldots, X_t$ given $E_{1:t}^{=e}$. What if we want the *most likely sequence* of states? i.e. $\max\limits_{x_1,\ldots,x_t}(P(X_{1:t}^{=x} \mid E_{1:t}^{=e}))$?

Example 2.16. Given the sequence $\mathtt{U}_1, \mathtt{U}_2, \neg\mathtt{U}_3, \mathtt{U}_4, \mathtt{U}_5$, the most likely state for $\mathtt{R}_3$ is F, but the most likely sequence *might* be that it rained throughout...

Prominent Application: In speech recognition, we want to find the most likely word sequence, given what we have heard. (can be quite noisy)

Idea:

- For every $x_t \in \mathrm{dom}(X)$ and $0 \leq i \leq t$, recursively compute the most likely path $X_1, \ldots, X_i$ ending in $X_i = x_i$ given the observed evidence.
- remember the x_{i-1} that most likely leads to x_i.
- Among the resulting paths, pick the one to *the* $X_t = x_t$ with the most likely path,
- and then recurse backwards.

$\leadsto$ we want to know $\max\limits_{x_1,\ldots,x_{t-1}} \mathbb{P}(X_{1:t-1}^{=x}, X_t | E_{1:t}^{=e})$, and then pick the x_t with the maximal value.

SOME RIGHTS RESERVED

Most Likely Explanation (continued)

By the same reasoning as for filtering:

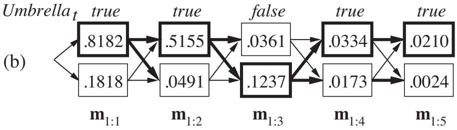
$$\max_{x_1,\ldots,x_{t-1}} \mathbb{P}(X_{1:t-1}^{=x}, X_t | E_{1:t}^{=e})$$

$$= \alpha(\underbrace{\mathbb{P}(E_t = e_t | X_t)}_{\text{sensor model}} \cdot \max_{x_{t-1}} (\underbrace{\mathbb{P}(X_t | X_{t-1} = x_{t-1})}_{\text{transition model}} \cdot \underbrace{\max_{x_1,\ldots,x_{t-2}} (P(X_{1:t-2}^{=x}, X_{t-1} = x_{t-1}}_{=:\mathbf{m}_{1:t-1}(x_{t-1})}$$

$m_{1:t}(i)$ gives the maximal probability that the most likely path up to t leads to state $X_t = i$.

Note that we can leave out the α, since we're only interested in the maximum.

Example 2.17. For the sequence [T, T, F, T, T]:



bold arrows: best predecessor measured by "best preceding sequence probability × transition probability"

SOME RIGHTS RESERVED

The Viterbi Algorithm

Definition 2.18. The Viterbi algorithm now proceeds as follows:

```
function $\textsc{Viterbi}(\langle e_1, \ldots, e_t \rangle, \mathbb{P}(X_0))$
    $m := \mathbb{P}(X_0)$                                                  /* $m_{1:i}$ */
    $\mathrm{prev} := \langle\rangle$          /* the most likely predecessor of each possible $x_i$ */
    for $i = 1, \ldots, t$ do
        $m' := \max_{x_{i-1}} (\mathbb{P}(E_i = e_i | X_i) \cdot \mathbb{P}(X_i | X_{i-1} = x_{i-1}) \cdot m_{x_{i-1}})$
        $\mathrm{prev}_{i-1} := \mathrm{argmax}_{x_{i-1}} (\mathbb{P}(E_i = e_i | X_i) \cdot \mathbb{P}(X_i | X_{i-1} = x_{i-1}) \cdot m_{x_{i-1}})$
        $m \longleftarrow m'$
    $P := \langle 0, 0, \ldots, \mathrm{argmax}_{(x \in \mathrm{dom}(X))} m_x \rangle$
    for $i = t-1, \ldots, 0$ do
        $P_i := \mathrm{prev}_{i, P_{i+1}}$
    return $P$
```

Observation 2.19. *Viterbi has linear time complexity and linear space complexity (needs to keep the most likely sequence leading to each state).*

SOME RIGHTS RESERVED

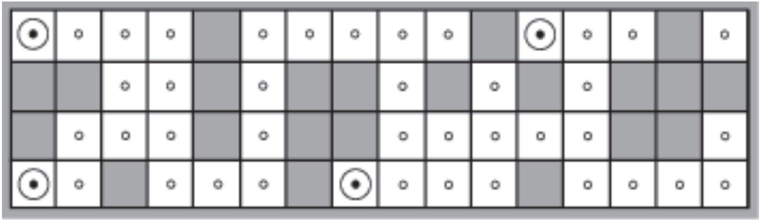
24.3 Hidden Markov Models – Extended Example

SOME RIGHTS RESERVED

Example: Robot Localization using Common Sense

Example 3.1 (Robot Localization in a Maze). A robot has four sonar sensors that tell it about obstacles in four directions: N, S, W, E.
We write the result where the sensor that detects obstacles in the north, south, and east as $\mathrm{N\;S\;E}$.

We filter out the impossible states:



a) Possible robot locations after $\mathrm{e}_1 = \mathrm{N\;S\;W}$

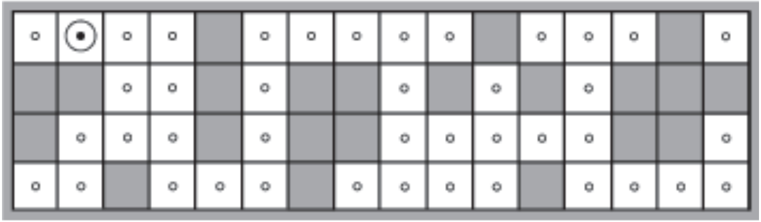
Remark 3.2. This only works for perfect sensors. (else no impossible states)
What if our sensors are imperfect?

SOME RIGHTS RESERVED

Example: Robot Localization using Common Sense

Example 3.3 (Robot Localization in a Maze). A robot has four sonar sensors that tell it about obstacles in four directions: N, S, W, E.
We write the result where the sensor that detects obstacles in the north, south, and east as $\mathrm{N\ S\ E}$.

We filter out the impossible states:



b) Possible robot locations after $\mathrm{e}_1 = \mathrm{N\ S\ W}$ and $\mathrm{e}_2 = \mathrm{N\ S}$

Remark 3.4. This only works for perfect sensors. (else no impossible states)
What if our sensors are imperfect?

HMM Example: Robot Localization (Modeling)

Example 3.5 (HMM-based Robot Localization). We have the following setup:

- A hidden Random variable X_t for robot location (domain: 42 empty squares)
- Let $N(i)$ be the set of neighboring fields of the field $X_i = x_i$
- The Transition matrix for the move action (T has $42^2 = 1764$ entries)

$$P(X_{t+1} = j \mid X_t = i) = \mathrm{T}_{ij} = \begin{cases} \frac{1}{|N(i)|} & \text{if } j \in N(i) \\ 0 & \text{else} \end{cases}$$

- We do not know where the robot starts: $P(X_0) = \frac{1}{n}$ (here $n = 42$)
- Evidence variable E_t: four bit presence/absence of obstacles in N, S, W, E. Let d_{it} be the number of wrong bits and ϵ the error rate of the sensor. Then

$$P(E_t = e_t \mid X_t = i) = \mathrm{O}_{t\,ii} = (1-\epsilon)^{4-d_{it}} \cdot \epsilon^{d_{it}}$$

(We assume the sensors are independent)

For example, the probability that the sensor on a square with obstacles in north and south would produce N S E is $(1-\epsilon)^3 \cdot \epsilon^1$.

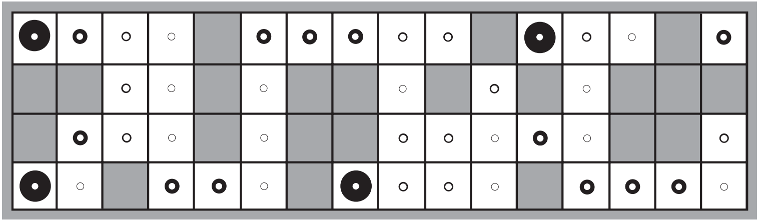
We can now use filtering for localization, smoothing to determine e.g. the starting location, and the Viterbi algorithm to find out how the robot got to where it is now.

SOME RIGHTS RESERVED

HMM Example: Robot Localization

We use HMM filtering equation $\mathsf{f}_{1:t+1} = \alpha \cdot \mathsf{O}_{t+1} \mathsf{T}^t \mathsf{f}_{1:t}$ to compute posterior distribution over locations. (i.e. robot localization)

Example 3.6. Redoing ???, with $\epsilon = 0.2$.



a) Posterior distribution over robot location after $\mathrm{E}_1 = \mathrm{N\ S\ W}$

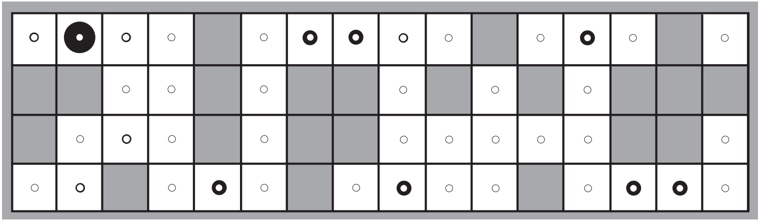
Still the same locations as in the "perfect sensing" case, but now other locations have non-zero probability.

SOME RIGHTS RESERVED

HMM Example: Robot Localization

We use HMM filtering equation $\mathsf{f}_{1:t+1} = \alpha \cdot \mathsf{O}_{t+1}\mathsf{T}^{t}\mathsf{f}_{1:t}$ to compute posterior distribution over locations. (i.e. robot localization)

Example 3.7. Redoing ???, with $\epsilon = 0.2$.



b) Posterior distribution over robot location after $\mathrm{E}_1 = \mathrm{N\ S\ W}$ and $\mathrm{E}_2 = \mathrm{N\ S}$

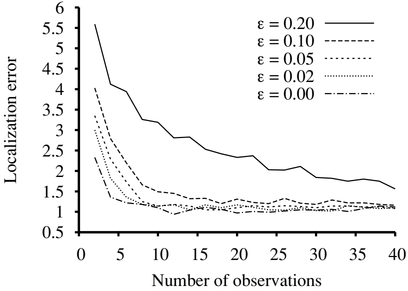
Still the same locations as in the "perfect sensing" case, but now other locations have non-zero probability.

SOME RIGHTS RESERVED

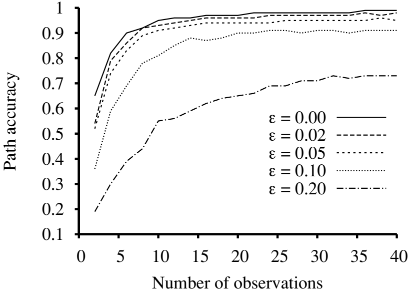
HMM Example: Further Inference Applications

Idea: We can use smoothing: $\mathbf{b}_{k+1:t} = \mathbf{T}\mathbf{O}_{k+1}\mathbf{b}_{k+2:t}$ to find out where it started and the Viterbi algorithm to find the most likely path it took.

Example 3.8. Performance of HMM localization vs. observation length (various error rates ϵ)



Localization error (Manhattan distance from true location)



Viterbi path accuracy (fraction of correct states on Viterbi path)

SOME RIGHTS RESERVED

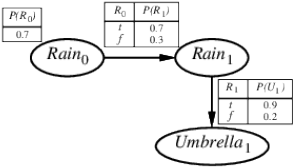
24.4 Dynamic Bayesian Networks

SOME RIGHTS RESERVED

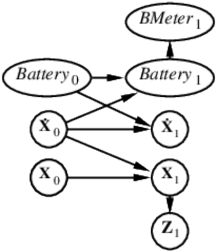
Dynamic Bayesian networks

- **Definition 4.1.** A Bayesian network $\mathcal{D}$ is called dynamic (a DBN), iff its random variables are indexed by a time structure. We assume that $\mathcal{D}$ is
  - time sliced, i.e. that the time slices $\mathcal{D}_t$ – the subgraphs of t-indexed random variables and the edges between them – are isomorphic.
  - a stationary Markov chain, i.e. that variables X_t can only have parents in $\mathcal{D}_t$ and $\mathcal{D}_{t-1}$.
- X_t, E_t contain arbitrarily many variables in a replicated Bayesian network.
- **Example 4.2.**

Umbrellas

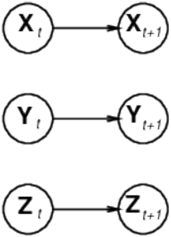


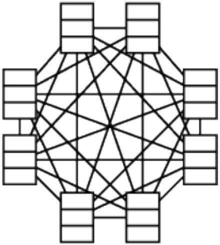
Robot Motion



DBNs vs. HMMs

- **Observation 4.3.**
  - *Every HMM is a single-variable DBN.* (*trivially*)
  - *Every DBN can be turned into an HMM.* (*combine variables into tuple* ⇒ *lose information about dependencies*)
  - *DBNs have sparse dependencies* $\rightsquigarrow$ *exponentially fewer parameters;*



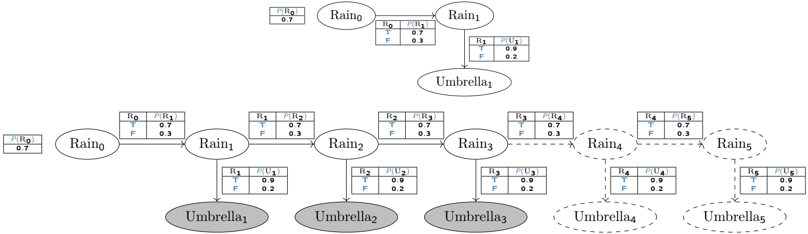


- **Example 4.4 (Sparse Dependencies).** With 20 Boolean state variables, three parents each, a DBN has $20 \cdot 2^3 = 160$ parameters, the corresponding HMM has $2^{20} \cdot 2^{20} \approx 10^{12}$.

SOME RIGHTS RESERVED

Exact inference in DBNs

- **Definition 4.5 (Naive method).** Unroll the network and run any exact algorithm.



- **Problem:** Inference cost for each update grows with t.
- **Definition 4.6.** Rollup filtering: add slice $t+1$, "sum out" slice t using variable elimination.
- **Observation:** Largest factor is $\mathcal{O}(d^{n+1})$, update cost $\mathcal{O}(d^{n+2})$, where d is the maximal domain size.
- **Note:** Much better than the HMM update cost of $\mathcal{O}(d^{2n})$

SOME RIGHTS RESERVED

Summary

- Temporal probability models use state and evidence variables replicated over time.
- Markov property and stationarity assumption, so we need both
  - a transition model and $P(X_t|X_{t-1})$
  - a sensor model $P(E_t|X_t)$.
- Tasks are filtering, prediction, smoothing, most likely sequence; (all done recursively with constant cost per time step)
- Hidden Markov models have a single discrete state variable; (used for speech recognition)
- DBNs subsume HMMs, exact update intractable.

SOME RIGHTS RESERVED

Chapter 25
Making Complex Decisions

Outline

We will now combine the ideas of stochastic process with that of acting based on maximizing expected utility:

- Markov decision processes (MDPs) for sequential environments.
- Value/policy iteration for computing utilities in MDPs.
- Partially observable MDP (POMDPs).
- Decision theoretic agents for POMDPs.

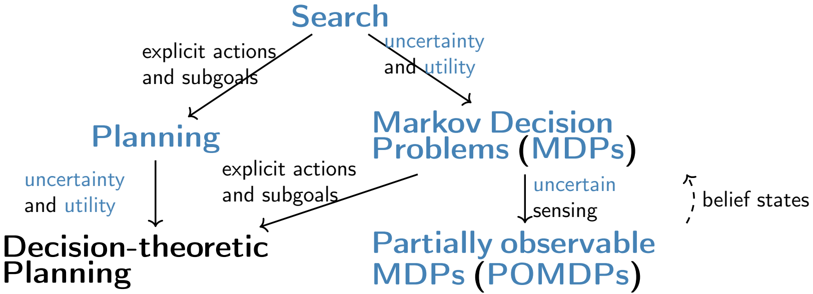
SOME RIGHTS RESERVED

25.1 Sequential Decision Problems

SOME RIGHTS RESERVED

Sequential Decision Problems

- **Definition 1.1.** In sequential decision problems, the agent's utility depends on a sequence of decisions (or their result states).
- **Definition 1.2.** Utility functions on action sequences are often expressed in terms of immediate rewards that are incurred upon reaching a (single) state.
- **Methods:** depend on the environment:
  - If it is fully observable ⇝ Markov decision process (MDPs)
  - else ⇝ partially observable MDP (POMDP).
- Sequential decision problems incorporate utilities, uncertainty, and sensing.
- **Preview:** Search problems and planning tasks are special cases.



Search

explicit actions and subgoals

uncertainty and utility

Planning

Markov Decision Problems (MDPs)

uncertainty and utility

explicit actions and subgoals

uncertain sensing

belief states

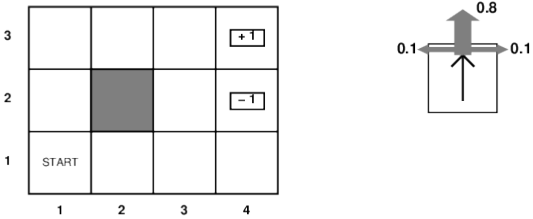
Decision-theoretic Planning

Partially observable MDPs (POMDPs)

SOME RIGHTS RESERVED

Markov Decision Problem: Running Example

- **Example 1.3 (Running Example: The 4x3 World).** A (fully observable) 4×3 environment with non-deterministic actions:



  - States $s \in \mathcal{S}$, actions $a \in \mathcal{A}s$.
  - Transition model: $P(s' \mid s,a) \mathrel{\widehat{=}}$ probability that a in s leads to s'.
  - reward function:

$$R(s) := \begin{cases} -0.04 & \text{if (small penalty) for nonterminal states} \\ \pm 1 & \text{if for terminal states} \end{cases}$$

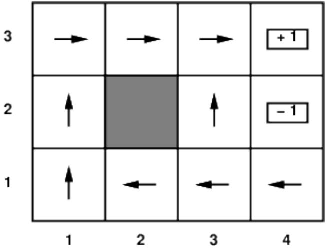
Markov Decision Process

- **Motivation:** Let us (for now) consider sequential decision problems in a fully observable, stochastic environment with a Markovian transition model on a *finite* set of states and an additive reward function. (We will switch to partially observable ones later)
- **Definition 1.4.** A Markov decision process (MDP) $\langle\mathcal{S},\mathcal{A},\mathcal{T},s_0,R\rangle$ consists of
  - a set of $\mathcal{S}$ of states (with initial state $s_0\in\mathcal{S}$),
  - for every state s, a set of actions $\mathcal{A}s$.
  - a transition model $\mathcal{T}(s,a)=\mathbb{P}(\mathcal{S}|s,a)$, and
  - a reward function $R\colon\mathcal{S}\to\mathbb{R}$; we call $R(s)$ a reward.
- **Idea:** We use the rewards as a utility function: The goal is to choose actions such that the expected *cumulative* rewards for the "foreseeable future" is maximized

  ⇒ need to take future actions and future states into account

Solving MDPs

- In MDPs, the aim is to find an optimal policy $\pi(s)$, which tells us the best action for every possible state s. (because we can't predict where we might end up, we need to consider all states)
- **Definition 1.5.** A policy π for an MDP is a function mapping each state s to an action $a \in \mathcal{A}s$.
  An optimal policy is a policy that maximizes the expected total rewards. (for some notion of "total"...)
- **Example 1.6.** Optimal policy when state penalty $R(s)$ is 0.04:

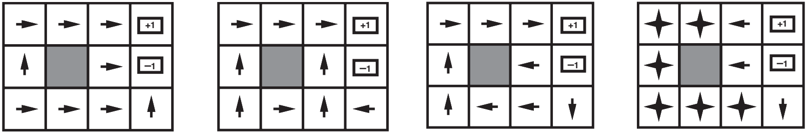


Note: When you run against a wall, you stay in your square.

SOME RIGHTS RESERVED

Risk and Reward

- **Example 1.7.** Optimal policy depends on the reward function $R(s)$.



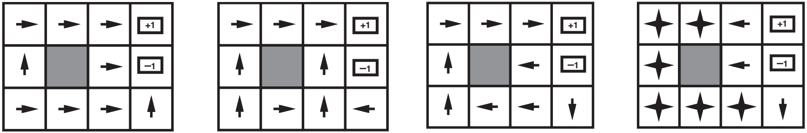
$R(s) < -1.6284$ $\quad -0.4278 < R(s) < -0.0850$ $\quad -0.0221 < R(s) < 0$ $\quad R(s) > 0$

- **Question:** Explain what you see in a qualitative manner!

SOME RIGHTS RESERVED

Risk and Reward

- **Example 1.8.** Optimal policy depends on the reward function $R(s)$.



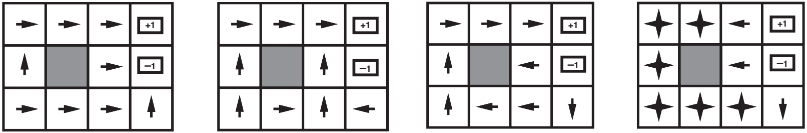
$R(s) < -1.6284$ $\quad -0.4278 < R(s) < -0.0850$ $\quad -0.0221 < R(s) < 0$ $\quad R(s) > 0$

- **Question:** Explain what you see in a qualitative manner!
- **Answer:** Careful risk/reward balancing is characteristic of MDPs.
  1. $-\infty \leq R(s) \leq -1.6284 \leadsto$ Life is so painful that agent heads for the next exit.

SOME RIGHTS RESERVED

Risk and Reward

- **Example 1.9.** Optimal policy depends on the reward function $R(s)$.



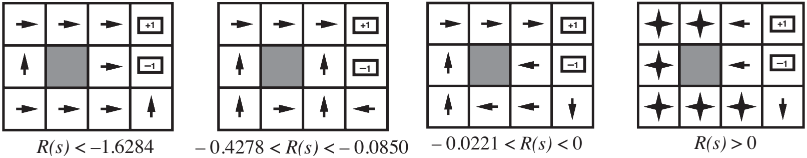
$R(s) < -1.6284$ $\quad$ $-0.4278 < R(s) < -0.0850$ $\quad$ $-0.0221 < R(s) < 0$ $\quad$ $R(s) > 0$

- **Question:** Explain what you see in a qualitative manner!
- **Answer:** Careful risk/reward balancing is characteristic of MDPs.
  1. $-\infty \leq R(s) \leq -1.6284 \leadsto$ Life is so painful that agent heads for the next exit.
  2. $-0.4278 \leq R(s) \leq -0.0850$, life is quite unpleasant; the agent takes the shortest route to the $+1$ state and is willing to risk falling into the -1 state by accident. In particular, the agent takes the shortcut from (3,1).

SOME RIGHTS RESERVED

Risk and Reward

- **Example 1.10.** Optimal policy depends on the reward function $R(s)$.

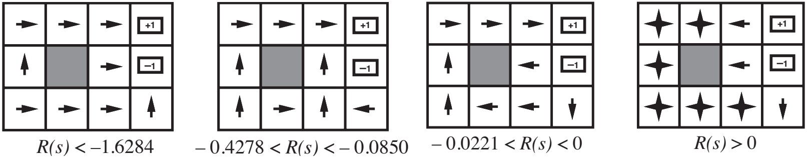


- **Question:** Explain what you see in a qualitative manner!
- **Answer:** Careful risk/reward balancing is characteristic of MDPs.
  1. $-\infty \leq R(s) \leq -1.6284 \rightsquigarrow$ Life is so painful that agent heads for the next exit.
  2. $-0.4278 \leq R(s) \leq -0.0850$, life is quite unpleasant; the agent takes the shortest route to the $+1$ state and is willing to risk falling into the -1 state by accident. In particular, the agent takes the shortcut from (3,1).
  3. Life is slightly dreary ($-0.0221 < R(s) < 0$) $\rightsquigarrow$ take no risks at all. In (4,1) and (3,2) head directly away from the -1 $\rightsquigarrow$ cannot fall in by accident.

SOME RIGHTS RESERVED

Risk and Reward

- **Example 1.11.** Optimal policy depends on the reward function $R(s)$.



- **Question:** Explain what you see in a qualitative manner!
- **Answer:** Careful risk/reward balancing is characteristic of MDPs.

1. $-\infty \leq R(s) \leq -1.6284 \leadsto$ Life is so painful that agent heads for the next exit.
2. $-0.4278 \leq R(s) \leq -0.0850$, life is quite unpleasant; the agent takes the shortest route to the $+1$ state and is willing to risk falling into the -1 state by accident. In particular, the agent takes the shortcut from (3,1).
3. Life is slightly dreary ($-0.0221 < R(s) < 0$) $\leadsto$ take no risks at all. In (4,1) and (3,2) head directly away from the -1 $\leadsto$ cannot fall in by accident.
4. If $R(s) > 0$, then life is positively enjoyable $\leadsto$ avoid both exits $\leadsto$ reap infinite rewards.

SOME RIGHTS RESERVED

25.2 Utilities over Time

SOME RIGHTS RESERVED

Utility of state sequences

Why rewards?

- **Recall:** We cannot observe/assess utility functions, only preferences $\rightsquigarrow$ induce utility functions from rational preferences
- **Problem:** In MDPs we need to understand preferences between *sequences* of states.
- **Definition 2.1.** We call preferences on reward sequences stationary, iff

$$[r,r_0,r_1,r_2,\ldots]\succ[r,r'_0,r'_1,r'_2,\ldots] \Leftrightarrow [r_0,r_1,r_2,\ldots]\succ[r'_0,r'_1,r'_2,\ldots]$$

(i.e. rewards over time are "independent" of each other)

- Good news:
  Theorem 2.2. *For stationary preferences, there are only two ways to combine rewards over time.*
  - *additive rewards:* $U([s_0,s_1,s_2,\ldots]) = R(s_0)+R(s_1)+R(s_2)+\cdots$
  - *discounted rewards:* $U([s_0,s_1,s_2,\ldots]) = R(s_0)+\gamma R(s_1)+\gamma^2 R(s_2)+\cdots$ *where* $0\leq\gamma\leq 1$ *is called discount factor.*

$\Rightarrow$ we can reduce utilities over time to rewards on individual states

SOME RIGHTS RESERVED

Utilities of State Sequences

Problem: Infinite lifetimes ⇝ additive rewards may become infinite.

Possible Solutions:

1. Finite horizon: terminate utility computation at a fixed time T

$$U([s_0,\ldots,s_\infty]) = R(s_0) + \cdots + R(s_T)$$

   ⇝ nonstationary policy: $\pi(s)$ depends on time left.

2. If there are absorbing states: for any policy π agent eventually "dies" with probability 1 ⇝ expected utility of every state is finite.
3. Discounting: assuming $\gamma < 1$, $R(s) \leq R_{\mathrm{max}}$,

$$U([s_0,s_1,\ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\mathrm{max}} = R_{\mathrm{max}}/(1-\gamma)$$

   Smaller γ ⇝ shorter horizon.

We will only consider discounted rewards in this course

SOME RIGHTS RESERVED

Why discounted rewards?

Discounted rewards are both convenient and (often) realistic:

- stationary preferences imply (additive rewards or) discounted rewards anyway,
- discounted rewards lead to finite utilities for (potentially) infinite sequences of states (we can compute expected utilities for the entire future),
- discounted rewards lead to stationary policies, which are easier to compute and often more adequate (unless we know that remaining time matters),
- discounted rewards mean we value *short-term gains* over *long-term gains* (all else being equal), which is often realistic (e.g. the same amount of money gained *early* gives more opportunity to spend/invest ⇒ potentially more utility in the long run)
- we can interpret the discount factor as a measure of *uncertainty about future rewards* ⇒ more robust measure in uncertain environments.

SOME RIGHTS RESERVED

Utility of States

Remember: Given a sequence of states $S = s_0, s_1, s_2, \ldots$, and a discount factor $0 \leq \gamma < 1$, the utility of the sequence is

$$U(S) = \sum_{t=0}^{\infty} \gamma^t R(s_t)$$

Definition 2.3. Given a policy π and a starting state s_0, let $S_{s_0}^{\pi}$ be the random variable giving the sequence of states resulting from executing π at every state starting at s_0. (Since the environment is stochastic, we don't know the exact sequence.)
Then the expected utility obtained by executing π starting in s_0 is given by

$$U^{\pi}(s_0) := \mathrm{EU}(S_{s_0}^{\pi}).$$

We define the optimal policy $\pi_{s_0}^{*} := \underset{\pi}{\mathrm{argmax}}\ U^{\pi}(s_0)$.

SOME RIGHTS RESERVED

Utility of States

Remember: Given a sequence of states $S = s_0, s_1, s_2, \ldots$, and a discount factor $0 \leq \gamma < 1$, the utility of the sequence is

$$U(S) = \sum_{t=0}^{\infty} \gamma^t R(s_t)$$

Definition 2.4. Given a policy π and a starting state s_0, let $S^\pi_{s_0}$ be the random variable giving the sequence of states resulting from executing π at every state starting at s_0. (Since the environment is stochastic, we don't know the exact sequence.)

Then the expected utility obtained by executing π starting in s_0 is given by

$$U^\pi(s_0) := \mathrm{EU}(S^\pi_{s_0}).$$

We define the optimal policy $\pi^*_{s_0} := \operatorname*{argmax}_{\pi} U^\pi(s_0)$.

Note: This is perfectly well-defined, but almost always computationally infeasible. (requires considering *all possible (potentially infinite) sequences of states*)

SOME RIGHTS RESERVED

Utility of States (continued)

Observation 2.5. $\pi^*_{s_0}$ *is independent of the state* s_0*.*

Proof sketch: If π^*_a and π^*_b reach point c, then there is no reason to disagree from that point on – or with π^*_c, and we expect optimal policies to "meet at some state" sooner or later.

⚠ **??** does not hold for finite horizon policies!

SOME RIGHTS RESERVED

Utility of States (continued)

Observation 2.8. $\pi^*_{s_0}$ *is independent of the state* s_0*.*
Proof sketch: If π^*_a and π^*_b reach point c, then there is no reason to disagree from that point on – or with π^*_c, and we expect optimal policies to "meet at some state" sooner or later.
⚠ **??** does not hold for finite horizon policies!

Definition 2.9. We call $\pi^* := \pi^*_s$ for some s the optimal policy.
Definition 2.10. The utility $U(s)$ of a state s is $U^{\pi^*}(s)$.

SOME RIGHTS RESERVED

Utility of States (continued)

Observation 2.11. $\pi^*_{s_0}$ *is independent of the state* s_0.

Proof sketch: If π^*_a and π^*_b reach point c, then there is no reason to disagree from that point on – or with π^*_c, and we expect optimal policies to "meet at some state" sooner or later.

⚠ **??** does not hold for finite horizon policies!

Definition 2.12. We call $\pi^* := \pi^*_s$ for some s the optimal policy.

Definition 2.13. The utility $U(s)$ of a state s is $U^{\pi^*}(s)$.

Remark: $R(s) \widehat{=}$ "immediate reward", whereas $U \widehat{=}$ "long-term reward".

Given the utilities of the states, choosing the best action is just MEU: maximize the expected utility of the immediate successor states

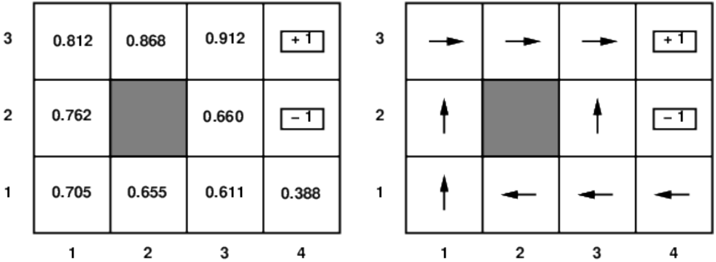
$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \left(\sum_{s'} P(s' \mid s,a) \cdot U(s') \right)$$

⇒ given the "true" utilities, we can compute the optimal policy and vice versa.

SOME RIGHTS RESERVED

Utility of States (continued)

- **Example 2.14 (Running Example Continued).**



Expected Utility

	1	2	3	4
3	0.812	0.868	0.912	+ 1
2	0.762	▨	0.660	− 1
1	0.705	0.655	0.611	0.388

Optimal Policy

	1	2	3	4
3	→	→	→	+ 1
2	↑	▨	↑	− 1
1	↑	←	←	←

- **Question:** Why do we go left in $(3,1)$ and not up? (follow the utility)

SOME RIGHTS RESERVED

25.3 Value/Policy Iteration

SOME RIGHTS RESERVED

Dynamic programming: the Bellman equation

- **Problem:** We have defined $U(s)$ via the optimal policy: $U(s) := U^{\pi^*}(s)$, but how to compute it without knowing π^*?
- **Observation:** A simple relationship among utilities of neighboring states:

expected sum of rewards = current reward $+ \gamma \cdot$ exp. reward sum after best action

- **Theorem 3.1 (Bellman equation (1957)).**

$$U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} U(s') \cdot P(s' \mid s, a)$$

We call this equation the Bellman equation

- **Example 3.2.** $U(1,1) = -0.04$

$$\begin{aligned} &+ \gamma \max\{0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && \textit{up}\\ &\quad 0.9U(1,1) + 0.1U(1,2) && \textit{left}\\ &\quad 0.9U(1,1) + 0.1U(2,1) && \textit{down}\\ &\quad 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\} && \textit{right}\end{aligned}$$

- **Problem:** One equation/state $\leadsto$ n nonlinear ($\max$ isn't) equations in n unknowns.

  $\leadsto$ cannot use linear algebra techniques for solving them.

SOME RIGHTS RESERVED

Value Iteration Algorithm

- **Idea:** We use a simple iteration scheme to find a fixpoint:
  1. start with arbitrary utility values,
  2. update to make them locally consistent with the Bellman equation,
  3. everywhere locally consistent ⇝ global optimality.
- **Definition 3.3.** The value iteration algorithm for utilitysutility function is given by

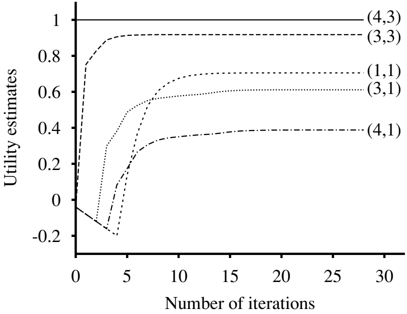
```
function VALUE−ITERATION (mdp,$\epsilon$) returns a utility fn.
  inputs: mdp, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s,a)$,
          rewards $R(s)$, and discount $\gamma$
        $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U$, $U'$, vectors of utilities for states in $S$, initially zero
          $\delta$, the maximum change in the utility of any state in an iteration
  repeat
     $U := U'$; $\delta := 0$
     for each state $s$ in $S$ do
       $U'[s] := R(s) + \gamma \cdot \max_{a \in A(s)} (\sum_{s'} U[s'] \cdot P(s' \mid s,a))$
       if $|U'[s] - U[s]| > \delta$ then $\delta := |U'[s] - U[s]|$
     until $\delta < \epsilon(1-\gamma)/\gamma$
     return $U$
```

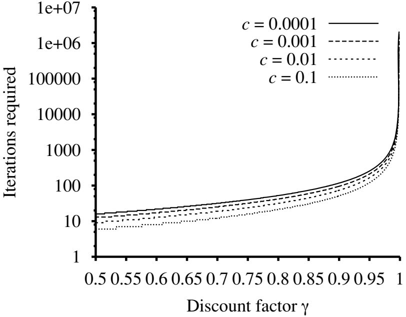
- **Remark:** Retrieve the optimal policy with
  $\pi[s] := \underset{a \in A(s)}{\operatorname{argmax}} (\sum_{s'} U[s'] \cdot P(s' \mid s,a))$

SOME RIGHTS RESERVED

Value Iteration Algorithm (Example)

- **Example 3.4 (Iteration on 4x3).**





(where $\varepsilon = c \cdot R_{max}$)

SOME RIGHTS RESERVED

Convergence

- **Definition 3.5.** The maximum norm is defined as $\|U\| = \max_s |U(s)|$, so $\|U - V\|$ = maximum difference between U and V.
- Let U^t and U^{t+1} be successive approximations to the true utility U during value iteration.
- **Theorem 3.6.** *For any two approximations* U^t *and* V^t

$$\left\|U^{t+1} - V^{t+1}\right\| \leq \gamma \left\|U^t - V^t\right\|$$

  I.e., any distinct approximations get closer to each other over time
  In particular, any approximation gets closer to the true U *over time*
  $\Rightarrow$ *value iteration converges to a unique, stable, optimal solution.*

- **Theorem 3.7.** *If* $\left\|U^{t+1} - U^t\right\| < \epsilon$, *then* $\left\|U^{t+1} - U\right\| < 2\epsilon\gamma/1 - \gamma$
  (*once the change in* U^t *becomes small, we are almost done.*)
- **Remark:** The policy resulting from U^t may be optimal long before the utilities convergence!

SOME RIGHTS RESERVED

Policy Iteration

- **Recap:** Value iteration computes utilities $\leadsto$ optimal policy by MEU.
- This even works if the utility estimate is inaccurate. ($\leftsquigarrow$ policy loss small)
- **Idea:** Search for optimal policy and utility values simultaneously [**Howard:dpmp60**]: Iterate
  - policy evaluation: given policy π_i, calculate $U_i = U^{\pi_i}$, the utility of each state were π_i to be executed.
  - policy improvement: calculate a new MEU policy π_{i+1} using 1 lookahead

  Terminate if policy improvement yields no change in computed utilities.
- **Observation 3.8.** *Upon termination U_i is a fixpoint of Bellman update $\leadsto$ Solution to Bellman equation $\leadsto$ π_i is an optimal policy.*
- **Observation 3.9.** *Policy improvement improves policy and policy space is finite $\leadsto$ termination.*

SOME RIGHTS RESERVED

Policy Iteration Algorithm

- **Definition 3.10.** The policy iteration algorithm is given by the following pseudocode:

```
function POLICY−ITERATION(mdp) returns a policy
  inputs: mdp, and MDP with states S, actions A(s), transition model $P(s' \mid s,a)$
  local variables: U a vector of utilities for states in S, initially zero
                   $\pi$ a policy indexed by state, initially random,
  repeat
     U := POLICY−EVALUATION($\pi$,U,mdp)
     unchanged? := true
     foreach state s in X do
        if $\max_{a \in A(s)} (\sum_{s'} P(s' \mid s,a) \cdot U(s')) > \sum_{s'} P(s' \mid s,\pi[s]) \cdot U(s')$ then do
           $\pi[s] := \operatorname{argmax}_{b \in A(s)} (\sum_{s'} P(s' \mid s,b) \cdot U(s'))$
           unchanged? := false
  until unchanged?
  return $\pi$
```

SOME RIGHTS RESERVED

Policy Evaluation

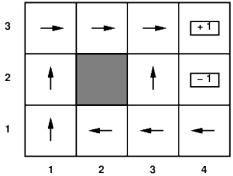
- **Problem:** How to implement the POLICY−EVALUATION algorithm?
- **Solution:** To compute utilities given a fixed π: For all s we have

$$U(s) = R(s) + \gamma(\sum_{s'} U(s') \cdot P(s' \mid s, \pi(s)))$$

(i.e. Bellman equation with the maximum replaced by the current policy π)

- **Example 3.11 (Simplified Bellman Equations for π).**

$$\begin{aligned} U_i(1,1) &= -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1) \\ U_i(1,2) &= -0.04 + 0.8U_i(1,3) + 0.1U_i(1,2) \\ &\vdots \end{aligned}$$



- **Observation 3.12.** *n simultaneous linear equations in n unknowns, solve in $\mathcal{O}(n^3)$ with standard linear algebra methods.*

SOME RIGHTS RESERVED

Modified Policy Iteration

- Value iteration requires many iterations, but each one is cheap.
- Policy iteration often converges in few iterations, but each is expensive.
- **Idea:** Use a few steps of value iteration (but with π fixed), starting from the value function produced the last time to produce an approximate value determination step.
- Often converges much faster than pure VI or PI.
- Leads to much more general algorithms where Bellman value updates and Howard policy updates can be performed locally in any order.
- **Remark:** Reinforcement learning algorithms operate by performing such updates based on the observed transitions made in an initially unknown environment.

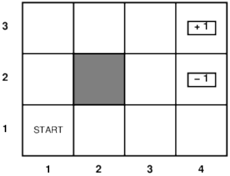
SOME RIGHTS RESERVED

25.4 Partially Observable MDPs

SOME RIGHTS RESERVED

Partial Observability

- **Definition 4.1.** A partially observable MDP (a POMDP for short) is a MDP together with an observation model O that has the sensor Markov property and is stationary: $O(s,e) = P(e \mid s)$.
- **Example 4.2 (Noisy 4x3 World).**

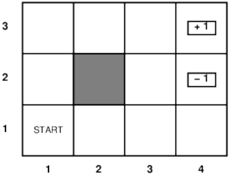


Add a partial and/or noisy sensor.
e.g. count number of adjacent walls ($1 \leq w \leq 2$)
with 0.1 error (noise)
If sensor reports 1, we are in $(3,?)$ (probably)

SOME RIGHTS RESERVED

Partial Observability

- **Definition 4.4.** A partially observable MDP (a POMDP for short) is a MDP together with an observation model O that has the sensor Markov property and is stationary: $O(s, e) = P(e \mid s)$.
- **Example 4.5 (Noisy 4x3 World).**

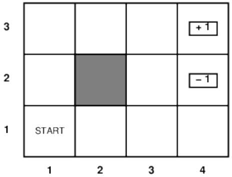


Add a partial and/or noisy sensor.
e.g. count number of adjacent walls $(1 \leq w \leq 2)$
with 0.1 error (noise)
If sensor reports 1, we are in $(3, ?)$ (probably)

- **Problem:** Agent does not know which state it is in ⇝ makes no sense to talk about policy $\pi(s)$!

Partial Observability

- **Definition 4.7.** A partially observable MDP (a POMDP for short) is a MDP together with an observation model O that has the sensor Markov property and is stationary: $O(s, e) = P(e \mid s)$.
- **Example 4.8 (Noisy 4x3 World).**



Add a partial and/or noisy sensor.
e.g. count number of adjacent walls $(1 \leq w \leq 2)$
with 0.1 error (noise)
If sensor reports 1, we are in $(3, ?)$ (probably)

- **Problem:** Agent does not know which state it is in $\rightsquigarrow$ makes no sense to talk about policy $\pi(s)$!
- **Theorem 4.9 (Astrom 1965).** *The optimal policy in a POMDP is a function $\pi(b)$ where b is the belief state (probability distribution over states).*
- **Idea:** Convert a POMDP into an MDP in belief state space, where $\mathcal{T}(b, a, b')$ is the probability that the new belief state is b' given that the current belief state is b and the agent does a. I.e., essentially a filtering update step.

SOME RIGHTS RESERVED

POMDP: Filtering at the Belief State Level

- **Recap:** Filtering updates the belief state for new evidence.
- For POMDPs, we also need to consider actions. (but the effect is the same)
- If b is the previous belief state and agent does action $A = a$ and then perceives $E = e$, then the new belief state is

  $$b' = \alpha(\mathbb{P}(E = e|s') \cdot (\sum_s \mathbb{P}(s'|S = s, A = a) \cdot b(s)))$$

  We write $b' = \mathrm{FORWARD}(b, a, e)$ in analogy to recursive state estimation.
- **Fundamental Insight for POMDPs:** The optimal action only depends on the agent's current belief state. (good, it does not know the state!)
- **Consequence:** The optimal policy can be written as a function $\pi^*(b)$ from belief states to actions.
- **Definition 4.10.** The POMDP decision cycle is to iterate over
  1. Given the current belief state b, execute the action $a = \pi^*(b)$
  2. Receive percept e.
  3. Set the current belief state to $\mathrm{FORWARD}(b, a, e)$ and repeat.
- **Intuition:** POMDP decision cycle is search in belief state space.

SOME RIGHTS RESERVED

Partial Observability contd.

- **Recap:** The POMDP decision cycle is search in belief state space.
- **Observation 4.11.** *Actions change the belief state, not just the (physical) state.*
- **Thus** POMDP solutions automatically include information gathering behavior.
- **Problem:** The belief state is continuous: If there are n states, b is an n-dimensional real-valued vector.
- **Example 4.12.** The belief state of the 4x3 world is a 11 dimensional continuous space. (11 states)
- **Theorem 4.13.** *Solving POMDPs is very hard!* (*actually,* PSPACE *hard*)
- **In particular,** none of the algorithms we have learned applies. (discreteness assumption)
- The real world is a POMDP (with initially unknown transition model T and sensor model O)

SOME RIGHTS RESERVED

Reducing POMDPs to Belief-State MDPs I

- **Idea:** Calculating the probability that an agent in belief state b reaches belief state b' after executing action a.
  - if we knew the action and the *subsequent* percept e, then $b' = \mathrm{FORWARD}(b, a, e)$.
    (deterministic update to the belief state)
  - but we don't, since b' depends on e. (let's calculate $P(e \mid a,b)$)
- **Idea:** To compute $P(e \mid a,b)$ — the probability that e is perceived after executing a in belief state b — sum up over all actual states the agent might reach:

$$\begin{aligned}
P(e \mid a,b) &= \sum_{s'} P(e \mid a,s',b) \cdot P(s' \mid a,b) \\
&= \sum_{s'} P(e \mid s') \cdot P(s' \mid a,b) \\
&= \sum_{s'} P(e \mid s') \cdot (\sum_{s} P(s' \mid s,a), b(s))
\end{aligned}$$

SOME RIGHTS RESERVED

Reducing POMDPs to Belief-State MDPs II

Write the probability of reaching b' from b, given action a, as $P(b' \mid b,a)$, then

$$\begin{aligned} P(b' \mid b,a) &= P(b' \mid a,b) = \sum_e P(b' \mid e,a,b) \cdot P(e \mid a,b) \\ &= \sum_e P(b' \mid e,a,b) \cdot (\sum_{s'} P(e \mid s') \cdot (\sum_s P(s' \mid s,a), b(s))) \end{aligned}$$

where $P(b' \mid e,a,b)$ is 1 if $b' = \mathrm{FORWARD}(b,a,e)$ and 0 otherwise.

- **Observation:** This equation defines a transition model for belief state space!
- **Idea:** We can also define a reward function for belief states:

$$\rho(b) := \sum_s b(s) \cdot R(s)$$

i.e., the expected reward for the actual states the agent might be in.

SOME RIGHTS RESERVED

Reducing POMDPs to Belief-State MDPs III

- Together, $P(b' \mid b,a)$ and $\rho(b)$ define an (observable) MDP on the space of belief states.
- **Theorem 4.14.** *An optimal policy* $\pi^*(b)$ *for this MDP, is also an optimal policy for the original POMDP.*
- **Upshot:** Solving a POMDP on a physical state space can be reduced to solving an MDP on the corresponding belief state space.
- **Remember:** The belief state is always observable to the agent, by definition.

SOME RIGHTS RESERVED

Ideas towards Value-Iteration on POMDPs

- **Recap:** The value iteration algorithm from ??? computes one utility value per state.
- **Problem:** We have infinitely many belief states $\rightsquigarrow$ be more creative!
- **Observation:** Consider an optimal policy π^*
  - applied in a specific belief state b: π^* generates an action,
  - for each subsequent percept, the belief state is updated and a new action is generated ...

  For this specific b: $\pi^* \widehat{=}$ a conditional plan!
- **Idea:** Think about conditional plans and how the expected utility of executing a fixed conditional plan varies with the initial belief state. (instead of optimal policies)

Definition 4.15. Given a set of percepts E and a set of actions A, a conditional plan is either an action $a \in A$, or a tuple $\langle a, E', p_1, p_2 \rangle$ such that $a \in A$, $E' \subseteq E$, and p_1, p_2 are conditional plans.
It represents the strategy "First execute a, If we subsequently perceive $e \in E'$, continue with p_1, otherwise continue with p_2."
The depth of a conditional plan p is the maximum number of actions in any path from p before reaching a single action plan.

SOME RIGHTS RESERVED

Expected Utilities of Conditional Plans on Belief States

- **Observation 1:** Let p be a conditional plan and $\alpha_p(s)$ the utility of executing p in state s.
  - the expected utility of p in belief state b is $\sum_s b(s) \cdot \alpha_p(s) \mathrel{\widehat{=}} b{\cdot}\alpha_p$ as vectors.
  - the expected utility of a fixed conditional plan varies linearly with b
  - $\leadsto$ the "*best* conditional plan to execute" corresponds to a hyperplane in belief state space.

SOME RIGHTS RESERVED

Expected Utilities of Conditional Plans on Belief States

- **Observation 1:** Let p be a conditional plan and $\alpha_p(s)$ the utility of executing p in state s.
  - the expected utility of p in belief state b is $\sum_s b(s) \cdot \alpha_p(s) \widehat{=} b{\cdot}\alpha_p$ as vectors.
  - the expected utility of a fixed conditional plan varies linearly with b
  - $\rightsquigarrow$ the "*best* conditional plan to execute" corresponds to a hyperplane in belief state space.
- **Observation 2:** We can replace the *original* actions by conditional plans on those actions!
  Let π^* be the subsequent optimal policy. At any given belief state b,
  - π^* will choose to execute the conditional plan with highest expected utility
  - the expected utility of b under the π^* is the utility of that plan:

$$U(b) = U^{\pi^*}(b) = \max_b (b{\cdot}\alpha_p)$$

  - If the optimal policy π^* chooses to execute p starting at b, then it is reasonable to expect that it might choose to execute p in belief states that are very close to b;
  - if we bound the depth of the conditional plans, then there are only finitely many such plans
  - the continuous space of belief states will generally be divided into regions, each corresponding to a particular conditional plan that is optimal in that region.

SOME RIGHTS RESERVED

Expected Utilities of Conditional Plans on Belief States

- **Observation 1:** Let p be a conditional plan and $\alpha_p(s)$ the utility of executing p in state s.
  - the expected utility of p in belief state b is $\sum_s b(s) \cdot \alpha_p(s) \widehat{=} b{\cdot}\alpha_p$ as vectors.
  - the expected utility of a fixed conditional plan varies linearly with b
  - $\leadsto$ the "*best* conditional plan to execute" corresponds to a hyperplane in belief state space.
- **Observation 2:** We can replace the *original* actions by conditional plans on those actions!
  Let π^* be the subsequent optimal policy. At any given belief state b,
  - π^* will choose to execute the conditional plan with highest expected utility
  - the expected utility of b under the π^* is the utility of that plan:

    $$U(b) = U^{\pi^*}(b) = \max_b (b{\cdot}\alpha_p)$$

  - If the optimal policy π^* chooses to execute p starting at b, then it is reasonable to expect that it might choose to execute p in belief states that are very close to b;
  - if we bound the depth of the conditional plans, then there are only finitely many such plans
  - the continuous space of belief states will generally be divided into regions, each corresponding to a particular conditional plan that is optimal in that region.
- **Observation 3 (conbined):** The utility function $U(b)$ on belief states, being the maximum of a collection of hyperplanes, is piecewise linear and convex.

SOME RIGHTS RESERVED

A simple Illustrating Example I

- **Example 4.16.** A world with states S_0 and S_1, where $R(S_0) = 0$ and $R(S_1) = 1$ and two actions:
  - "Stay" stays put with probability 0.9
  - "Go" switches to the other state with probability 0.9.
  - The sensor reports the correct state with probability 0.6.

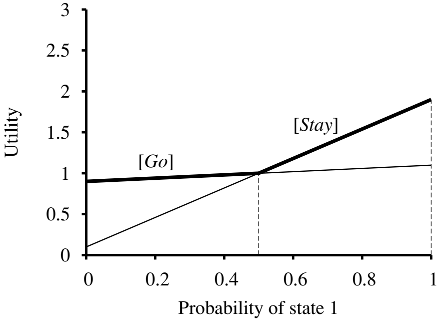
  Obviously, the agent should "Stay" when it thinks it's in state S_1 and "Go" when it thinks it's in state S_0.
- The belief state has dimension 1. (the two probabilities sum up to 1)
- Consider the one-step plans $[Stay]$ and $[Go]$ and their direct utilities:

$$\begin{aligned}
\alpha_{([Stay])}(S_0) &= 0.9R(S_0) + 0.1R(S_1) = 0.1\\
\alpha_{([stay])}(S_1) &= 0.9R(S_1) + 0.1R(S_0) = 0.9\\
\alpha_{([go])}(S_0) &= 0.9R(S_1) + 0.1R(S_0) = 0.9\\
\alpha_{([go])}(S_1) &= 0.9R(S_0) + 0.1R(S_1) = 0.1
\end{aligned}$$

SOME RIGHTS RESERVED

A simple Illustrating Example II

- Let us visualize the hyperplanes $b \cdot \alpha_{([Stay])}$ and $b \cdot \alpha_{([Go])}$.



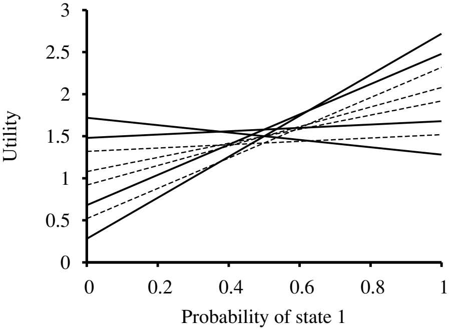
  - The maximum represents the utility function for the finite-horizon problem that allows just one action
  - in each "piece" the optimal action is the first action of the corresponding plan.
  - Here the optimal one-step policy is to "Stay" when $b(1) > 0.5$ and "Go" otherwise.

A simple Illustrating Example III

- compute the utilities for conditional plans of depth 2 by considering
  - each possible first action,
  - each possible subsequent percept, and then
  - each way of choosing a depth-1 plan to execute for each percept:

  There are eight of depth 2:

$$[Stay, \text{if } P = 0 \text{ then } Stay \text{ else } Stay \text{ fi}], [Stay, \text{if } P = 0 \text{ then } Stay \text{ else } Go \text{ fi}], \ldots$$



SOME RIGHTS RESERVED

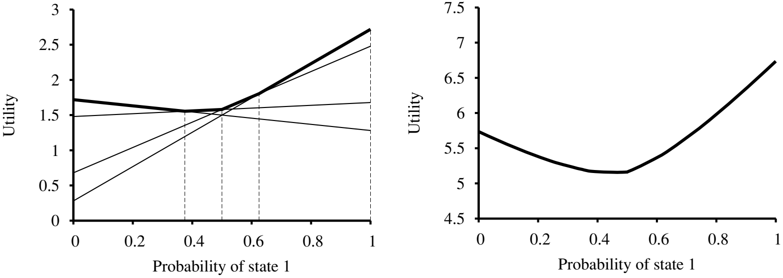
A simple Illustrating Example IV

Four of them (dashed lines) are suboptimal for the whole belief space
We call them dominated (they can be ignored)

SOME RIGHTS RESERVED

A simple Illustrating Example V

- There are four undominated plans, each optimal in their region



- **Idea:** Repeat for depth 3 and so on.
- **Theorem 4.17 (POMDP Plan Utility).** *Let p be a depth-d conditional plan whose initial action is a and whose depth-d* -1*-subplan for percept e is p.e, then*

$$\alpha_p(s) = R(s) + \gamma(\sum_{s'} P(s' \mid s,a)(\sum_{e} P(e \mid s') \cdot \alpha_{p.e}(s')))$$

- This recursion naturally gives us a value iteration algorithm,

A Value Iteration Algorithm for POMDPs

Definition 4.18. The POMDP value iteration algorithm for POMDPs is given by recursively updating

$$\alpha_p(s) = R(s) + \gamma(\sum_{s'} P(s' \mid s,a)(\sum_{e} P(e \mid s') \cdot \alpha_{p.e}(s')))$$

Observations: The complexity depends primarily on the generated plans:

- Given $|A|$ actions and $|E|$ possible observations, there are are $|A|^{|E|^{d-1}}$ distinct depth-d plans.
- Even for the example with $d = 8$, we have 2255 (144 undominated)
- The elimination of dominated plans is essential for reducing this doubly exponential growth (but they are already constructed)

Hopelessly inefficient in practice – even the 3x4 POMDP is too hard!

SOME RIGHTS RESERVED

25.5 Online Agents with POMDPs

SOME RIGHTS RESERVED

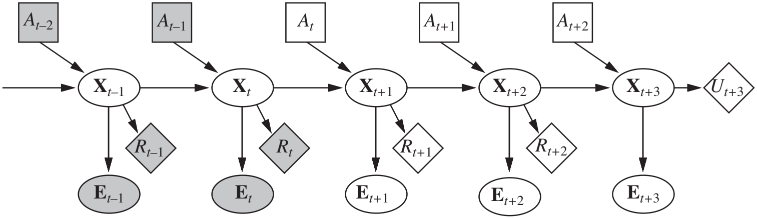
DDN: Decision Networks for POMDPs

- **Idea:** Let's try to use the computationally efficient representations (dynamic Bayesian networks and decision networks) for POMDPs.
- **Definition 5.1.** A dynamic decision network (DDN) is a graph-based representation of a POMDP, where
  - Transition and sensor model are represented as a DBN.
  - Action nodes and utility nodes are added as in decision networks.
- In a DDN, a filtering algorithm is used to incorporate each new percept and action and to update the belief state representation.
- Decisions are made in DDN by projecting forward possible action sequences and choosing the best one.
- DDNs – like the DBNs they are based on – are factored representations $\leadsto$ typically exponential complexity advantages!

SOME RIGHTS RESERVED

Structure of DDNs for POMDPs

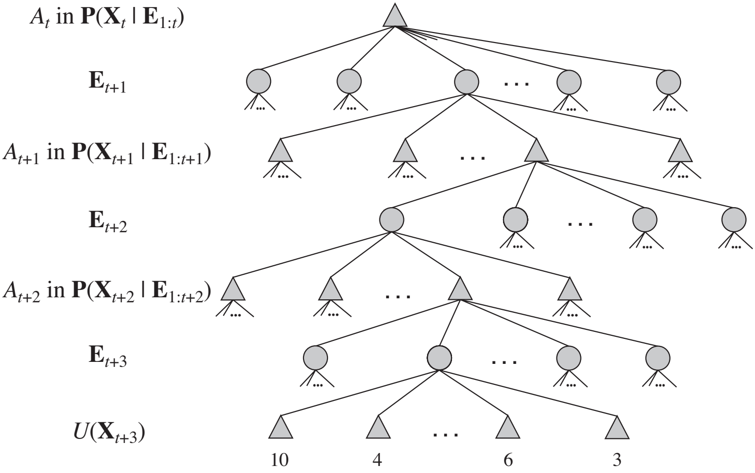
- **DDN for POMDPs:** The generic structure of a dymamic decision network at time t is



  - POMDP state S_t becomes a set of random variables X_t
  - there may be multiple evidence variables E_t
  - Action at time t denoted by A_t. agent must choose a value for A_t.
  - Transition model: $\mathbb{P}(\mathrm{X}_{t+1}|\mathrm{X}_t, A_t)$; sensor model: $\mathbb{P}(\mathrm{E}_t|\mathrm{X}_t)$.
  - Reward functions R_t and utility U_t of state S_t.
  - Variables with known values are gray, rewards for $t = 0, \ldots, t+2$, but utility for $t+3$ ($\widehat{=}$ discounted sum of rest)
- **Problem:** How do we compute with that?
- **Answer:** All POMDP algorithms can be adapted to DDNs! (only need CPTs)

Lookahead: Searching over the Possible Action Sequences

- **Idea:** Search over the tree of possible action sequences (like in game-play)
- Part of the lookahead solution of the DDN above (three steps lookahead)

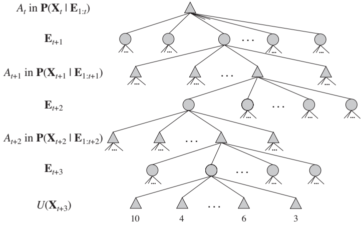


A_t in $\mathbf{P}(\mathbf{X}_t \mid \mathbf{E}_{1:t})$
$\mathbf{E}_{t+1}$
A_{t+1} in $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{E}_{1:t+1})$
$\mathbf{E}_{t+2}$
A_{t+2} in $\mathbf{P}(\mathbf{X}_{t+2} \mid \mathbf{E}_{1:t+2})$
$\mathbf{E}_{t+3}$
$U(\mathbf{X}_{t+3})$
...
10
4
6
3

- circle $\widehat{=}$ chance nodes (the environment decides)
- triangle $\widehat{=}$ belief state (each action decision is taken there)

SOME RIGHTS RESERVED

Designing Online Agents for POMDPs


A_t in $\mathbf{P}(\mathbf{X}_t \mid \mathbf{E}_{1:t})$
$\mathbf{E}_{t+1}$
A_{t+1} in $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{E}_{1:t+1})$
$\mathbf{E}_{t+2}$
A_{t+2} in $\mathbf{P}(\mathbf{X}_{t+2} \mid \mathbf{E}_{1:t+2})$
$\mathbf{E}_{t+3}$
$U(\mathbf{X}_{t+3})$
10
4
6
3

- Belief state at triangle computed by filtering with actions/percepts leading to it
  - for decision A_{t+i} will use percepts $\mathrm{E}_{t+1:t+i}$ (even if values at time t unknown)
  - thus a POMDP agent automatically takes into account the value of information and executes information gathering actions where appropriate.
- **Observation:** Time complexity for exhaustive search up to depth d is $\mathcal{O}(|A|^d \cdot |\mathrm{E}|^d)$ ($|A| \mathrel{\widehat{=}}$ number of actions, $|\mathrm{E}| \mathrel{\widehat{=}}$ number of percepts)
- **Upshot:** Much better than POMDP value iteration with $\mathcal{O}(|A|^{|E|^{d-1}})$.
- **Empirically:** For problems in which the discount factor γ is not too close to 1, a shallow search is often good enough to give near-optimal decisions.

SOME RIGHTS RESERVED

Summary

- Decision theoretic agents for sequential environments
- Building on temporal, probabilistic models/inference (dynamic Bayesian networks)
- MDPs for fully observable case.
- Value/Policy Iteration for MDPs $\leadsto$ optimal policies.
- POMDPs for partially observable case.
- POMDPs $\widehat{=}$ MDP on belief state space.
- The world is a POMDP with (initially) unknown transition and sensor models.

SOME RIGHTS RESERVED

Part 6
Machine Learning

SOME RIGHTS RESERVED

Chapter 26
Learning from Observations

Outline

- Learning agents
- Inductive learning
- Decision tree learning
- Measuring learning performance
- Computational Learning Theory
- Linear regression and classification
- Neural Networks
- Support Vector Machines

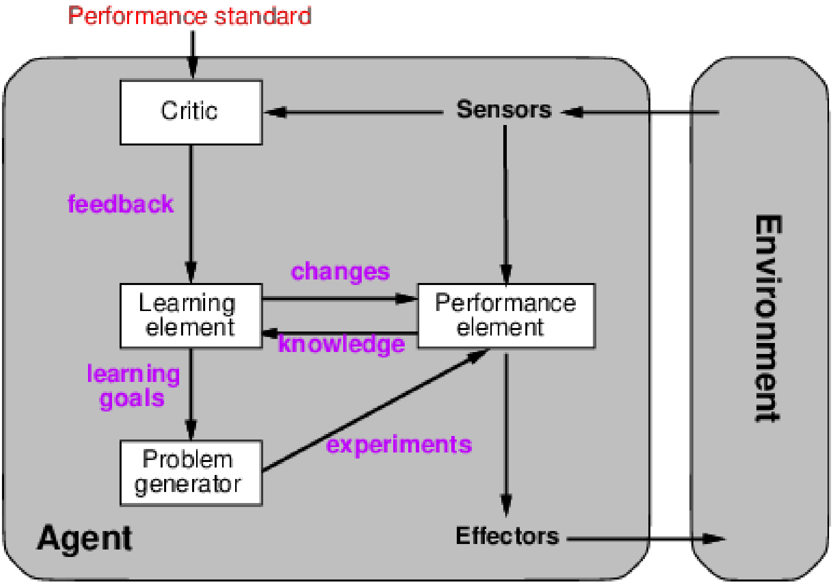
SOME RIGHTS RESERVED

26.1 Forms of Learning

Learning (why is this a good idea)

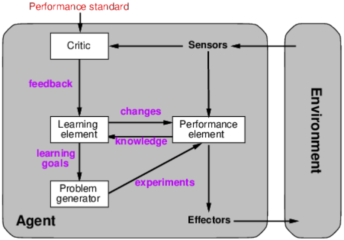
- Learning is essential for unknown environments:
  - i.e., when designer lacks omniscience.
  - The world is a POMDP with (initially) unknown transition and sensor models.
- Learning is useful as a system construction method.
  - i.e., expose the agent to reality rather than trying to write it down
- Learning modifies the agent's decision mechanisms to improve performance.

Recap: Learning Agents



SOME RIGHTS RESERVED

Recap: Learning Agents (continued)



- **Definition 1.1.** Performance element is what we called "agent" up to now.
- **Definition 1.2.** Critic/learning element/problem generator do the "improving".
- **Definition 1.3.** Performance standard is fixed; (outside the environment)
  - We can't adjust performance standard to flatter own behaviour!
  - No standard *in the environment*: e.g. ordinary chess and suicide chess look identical.
  - Essentially, certain kinds of percepts are "hardwired" as good/bad (e.g.,pain, hunger)
- **Definition 1.4.** Learning element may use knowledge already acquired in the performance element.
- **Definition 1.5.** Learning may require experimentation actions an agent might not normally consider such as dropping rocks from the Tower of Pisa.

SOME RIGHTS RESERVED

Ways of Learning

- Supervised learning: There's an unknown function $f\colon A \to B$ called the target function. We do know a set of pairs $T := \{\langle a_i, f(a_i)\rangle\}$ of examples. The goal is to find a hypothesis $h \in \mathcal{H} \subseteq A \to B$ based on T, that is "approximately" equal to f. (Most of the techniques we will consider)
- Unsupervised learning: Given a set of data A, find a *pattern* in the data; i.e. a function $f\colon A \to B$ for some predetermined B. (Primarily *clustering/dimensionality reduction*)
- Reinforcement learning: The agent receives a reward for each action performed. T he goal is to iteratively adapt the action function to maximize the total reward. (Useful in e.g. game play)

SOME RIGHTS RESERVED

26.2 Supervised Learning

Supervised learning a.k.a. inductive learning (a.k.a. Science)

Definition 2.1. A supervised (or inductive) learning problem consists of the following data:

- A set of hypotheses $\mathcal{H}$ consisting of functions $A \rightarrow B$,
- a set of examples $T \subseteq A \times B$ called the training set, such that for every $a \in A$, there is at most one $b \in B$ with $\langle a, b \rangle \in T$, ($\Rightarrow$ T is a function on some subset of A)

We assume there is an *unknown* function $f : A \rightarrow B$ called the target function with $T \subseteq f$.

Definition 2.2. Inductive learning algorithms solve inductive learning problems by finding a hypothesis $h \in \mathcal{H}$ such that $h \sim f$ (for some notion of similarity).

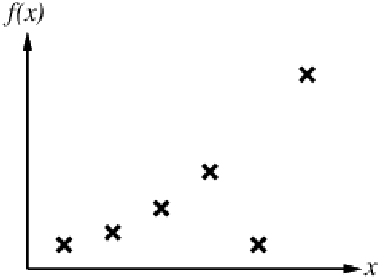
Definition 2.3. We call a supervised learning problem with target function $A \rightarrow B$ a classification problem if B is finite, and call the members of B classes.
We call it a regression problem if $B = \mathbb{R}$.

SOME RIGHTS RESERVED

Inductive Learning Method

- **Idea:** Construct/adjust hypothesis $h \in \mathcal{H}$ to agree with a training set T.
- **Definition 2.4.** We call h consistent with f (on a set $T \subseteq \mathrm{dom}(f)$), if it agrees with f (on all examples in T).
- **Example 2.5 (Curve Fitting).**

Training Set



- **Ockham's-razor:** maximize a combination of consistency and simplicity.

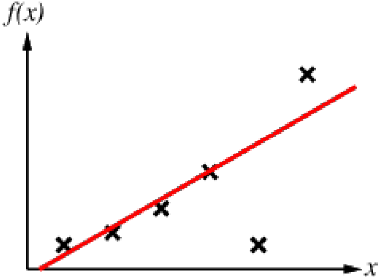
SOME RIGHTS RESERVED

Inductive Learning Method

- **Idea:** Construct/adjust hypothesis $h \in \mathcal{H}$ to agree with a training set T.
- **Definition 2.6.** We call h consistent with f (on a set $T \subseteq \mathrm{dom}(f)$), if it agrees with f (on all examples in T).
- **Example 2.7 (Curve Fitting).**

Linear Hypothesis

partially, approximatively consistent



- **Ockham's-razor:** maximize a combination of consistency and simplicity.

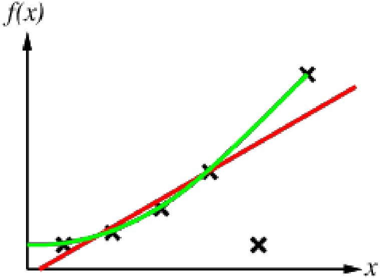
SOME RIGHTS RESERVED

Inductive Learning Method

- **Idea:** Construct/adjust hypothesis $h \in \mathcal{H}$ to agree with a training set T.
- **Definition 2.8.** We call h consistent with f (on a set $T \subseteq \mathrm{dom}(f)$), if it agrees with f (on all examples in T).
- **Example 2.9 (Curve Fitting).**

Quadratic Hypothesis

partially consistent



- **Ockham's-razor:** maximize a combination of consistency and simplicity.

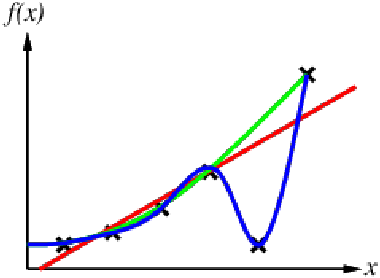
SOME RIGHTS RESERVED

Inductive Learning Method

- **Idea:** Construct/adjust hypothesis $h \in \mathcal{H}$ to agree with a training set T.
- **Definition 2.10.** We call h consistent with f (on a set $T \subseteq \mathrm{dom}(f)$), if it agrees with f (on all examples in T).
- **Example 2.11 (Curve Fitting).**

Degree-4 Hypothesis

consistent



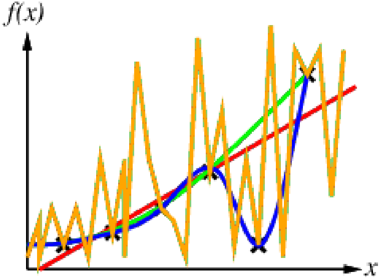
- **Ockham's-razor:** maximize a combination of consistency and simplicity.

Inductive Learning Method

- **Idea:** Construct/adjust hypothesis $h \in \mathcal{H}$ to agree with a training set T.
- **Definition 2.12.** We call h consistent with f (on a set $T \subseteq \mathrm{dom}(f)$), if it agrees with f (on all examples in T).
- **Example 2.13 (Curve Fitting).**

High-degree Hypothesis

consistent



- **Ockham's-razor:** maximize a combination of consistency and simplicity.

Choosing the Hypothesis Space

- **Observation:** Whether we can find a consistent hypothesis for a given training set depends on the chosen hypothesis space.
- **Definition 2.14.** We say that an supervised learning problem is realizable, iff there is a hypothesis $h \in \mathcal{H}$ consistent with the training set T.
- **Problem:** We do not always know whether a given learning problem is realizable, unless we have prior knowledge. (depending on the hypothesis space)
- **Solution:** Make $\mathcal{H}$ large, e.g. the class of all Turing machines.
- **Tradeoff:** The computational complexity of the supervised learning problem is tied to the size of the hypothesis space. E.g. consistency is not even decidable for general Turing machines.
- Much of the research in machine learning has concentrated on simple hypothesis spaces.
- **Preview:** We will concentrate on propositional logic and related languages first.

SOME RIGHTS RESERVED

Independent and Identically Distributed

- **Problem:** We want to learn a hypothesis that fits the future data best.
- **Intuition:** This only works, if the training set is "representative" for the underlying process.
- **Idea:** We think of examples (seen and unseen) as a sequence, and express the "representativeness" as a *stationarity assumption* for the probability distribution.
- **Method:** Each example before we see it is a random variable E_j, the observed value $e_j = (x_j, y_j)$ samples its distribution.
- **Definition 2.15.** A sequence of $E_1, \ldots, E_n$ of random variables is independent and identically distributed (short IID), iff they are
  - independent, i.e. $\mathbb{P}(E_j | E_{(j-1)}, E_{(j-2)}, \ldots) = \mathbb{P}(E_j)$ and
  - identically distributed, i.e. $\mathbb{P}(E_i) = \mathbb{P}(E_j)$ for all i and j.
- **Example 2.16.** A sequence of die tosses is IID. (fair or loaded does not matter)
- **Stationarity Assumption:** We assume that the set $\mathcal{E}$ of examples is IID in the future.

SOME RIGHTS RESERVED

26.3 Learning Decision Trees

SOME RIGHTS RESERVED

Attribute-based Representations

- **Definition 3.1.** In attribute-based representations, examples are described by
  - attributes: (simple) functions on input samples, (think pre classifiers on examples)
  - their values, and (classify by attributes)
  - classifications. (Boolean, discrete, continuous, etc.)
- **Example 3.2 (In a Restaurant).** Situations where I will/won't wait for a table:

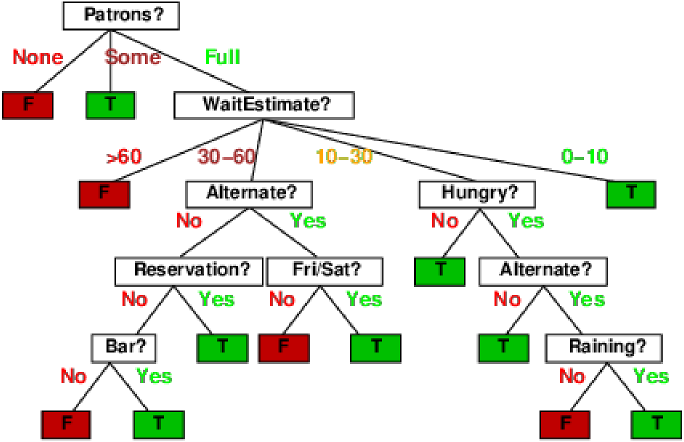
Example	*Attributes*										*Target*
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	WillWait
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

- **Definition 3.3.** For a boolean classification we say that an example is positive (T) or negative (F) depending on its class.

SOME RIGHTS RESERVED

Decision Trees

- Decision trees are one possible representation for hypotheses.
- **Example 3.4 (Restaurant continued).** Here is the "true" tree for deciding whether to wait:



SOME RIGHTS RESERVED

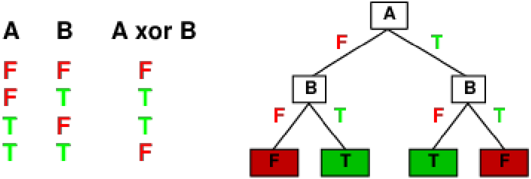
Decision Trees (Definition)

- **Definition 3.5.** A decision tree for a given attribute-based representation is a tree, where the non-leaf nodes are labeled by attributes, their outgoing edges by disjoint sets of attribute values, and the leaf nodes are labeled by the classifications.
- **Definition 3.6.** We call an attribute together with a set of attribute values (an inner node) with outgoing edge label an attribute test.
- the target function is a function $A_1 \times \ldots \times A_n \to C$, where A_i are the domains of the attributes and C is the set of classifications.

SOME RIGHTS RESERVED

Expressiveness

- Decision trees can express any function of the input attributes
  $\Rightarrow \mathcal{H} = A_1 \times \ldots \times A_n$
- **Example 3.7.** For Boolean functions, a path from the root to a leaf corresponds to a row in a truth table:



A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F

  $\Rightarrow$ a decision tree corresponds to a truth table (Formula in DNF)
- Trivially, for any training set there is a consistent hypothesis with one path to a leaf for each example, but it probably won't generalize to new examples.
- **Solution:** Prefer to find more *compact* decision trees.

Decision Tree learning

- **Aim:** Find a small decision tree consistent with the training examples.
- **Idea:** (recursively) choose "most significant" attribute as root of (sub)tree.
- **Definition 3.8.** The following algorithm performs decision tree learning (DTL)

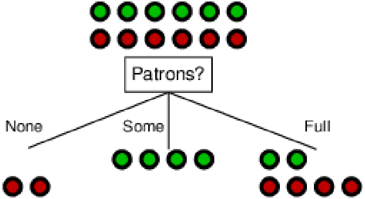
```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
   else if all examples have the same classification then return the classification
   else if attributes is empty then return MODE(examples)
   else
      best := Choose−Attribute(attributes, examples)
      tree := a new decision tree with root test best
      m := MODE(examples)
      for each value $v_i$ of best do
         $examples_i$ := {elements of examples with best = $v_i$}
         subtree := DTL($examples_i$, attributes \ best, m)
         add a branch to tree with label $v_i$ and subtree subtree
    return tree
```

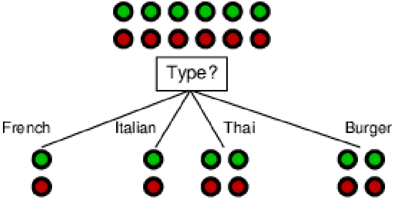
MODE(*examples*)= most frequent value in *example*.

SOME RIGHTS RESERVED

Choosing an Attribute

- **Idea:** A good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative".
- **Example 3.9.**





  Attribute "Patrons?" is a better choice, it gives gives information about the classification.

- Can we make this more formal? ⇝ Use information theory! (up next)

SOME RIGHTS RESERVED

26.4 Using Information Theory

SOME RIGHTS RESERVED

Information Entropy

Intuition: Information answers questions – the less I know initially, the more Information is contained in an answer.

Definition 4.1. Let $\langle p_1, \ldots, p_n \rangle$ the distribution of a random variable P. The information (also called entropy) of P is

$$I(\langle p_1, \ldots, p_n \rangle) := \sum_{i=1}^{n} -p_i \cdot \log_2(p_i)$$

Note: For $p_i = 0$, we consider $p_i \cdot \log_2(p_i) = 0$ ($\log_2(0)$ is undefined)

The unit of information is a bit, where $1\mathrm{b} := I(\langle \frac{1}{2}, \frac{1}{2} \rangle) = 1$

Example 4.2 (Information of a Coin Toss).

- For a fair coin toss we have $I(\langle \frac{1}{2}, \frac{1}{2} \rangle) = -\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1\mathrm{b}$.
- With a loaded coin (99% heads) we have $I(\langle \frac{1}{100}, \frac{99}{100} \rangle) = 0.08\mathrm{b}$.

Rightarrow Information goes to 0 as head probability goes to 1.

"How likely is the outcome actually going to tell me something informative?"

SOME RIGHTS RESERVED

Information Gain in Decision Trees

Idea: Suppose we have p examples classified as positive and n examples as negative. We can then estimate the probability distribution of the classification C with $\mathbb{P}(C) = \langle\frac{p}{p+n}, \frac{n}{p+n}\rangle$, and need $I(\mathbb{P}(C))$ bits to correctly classify a new example.

Example 4.3. For 12 restaurant examples and $p = n = 6$, we need $I(\mathbb{P}(\mathrm{WillWait})) = I(\langle\frac{6}{12}, \frac{6}{12}\rangle) = 1\mathrm{b}$ of information. (i.e. exactly the information which of the two classes)

Treating attributes also as random variables, we can compute how much information is needed *after* knowing the value for one attribute:

Example 4.4. If we know $\mathrm{Pat} = \mathrm{Full}$, we only need $I(\mathbb{P}(\mathrm{WillWait}|\mathrm{Pat} = \mathrm{Full})) = I(\langle\frac{4}{6}, \frac{2}{6}\rangle) \approxeq 0.9$ bits of information.

Note: The expected number of bits needed after an attribute test on A is

$$\sum_a P(A = a) \cdot I(\mathbb{P}(C|A = a))$$

Definition 4.5. The information gain from an attribute test A is

$$\mathrm{Gain}(A) := I(\mathbb{P}(C)) - \sum_a P(A = a) \cdot I(\mathbb{P}(C|A = a))$$

SOME RIGHTS RESERVED

Information Gain (continued)

- **Definition 4.6.** Assume we know the results of some attribute tests $b := B_1 = b_1 \wedge \ldots \wedge B_n = b_n$. Then the conditional information gain from an attribute test A is

$$\mathrm{Gain}(A|b) := I(\mathbb{P}(C|b)) - \sum_a P(A = a \mid b) \cdot I(\mathbb{P}(C|a, b))$$

- **Example 4.7.** If the classification C is Boolean and we have p positive and n negative examples, the information gain is

$$\mathrm{Gain}(A) = I(\langle \frac{p}{p+n}, \frac{n}{p+n} \rangle) - \sum_a \frac{p_a + n_a}{p+n} I(\langle \frac{p_a}{p_a + n_a}, \frac{n_a}{p_a + n_a} \rangle)$$

  where p_a and n_a are the positive and negative examples with $A = a$.
- **Example 4.8.**

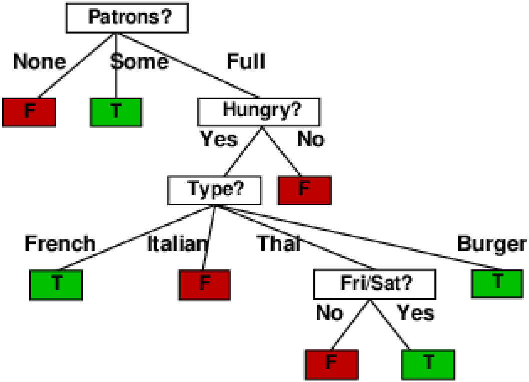
$$\begin{aligned} \mathrm{Gain}(Patrons?) &= 1 - (\frac{2}{12} I(\langle 0, 1 \rangle) + \frac{4}{12} I(\langle 1, 0 \rangle) + \frac{6}{12} I(\langle \frac{2}{6}, \frac{4}{6} \rangle)) \\ &\approx 0.541\mathrm{b} \\ \mathrm{Gain}(Type) &= 1 - (\frac{2}{12} I(\langle \frac{1}{2}, \frac{1}{2} \rangle) + \frac{2}{12} I(\langle \frac{1}{2}, \frac{1}{2} \rangle) + \frac{4}{12} I(\langle \frac{2}{4}, \frac{2}{4} \rangle) + \frac{4}{12} I(\langle \frac{2}{4}, \frac{2}{4} \rangle)) \\ &\approx 0\mathrm{b} \end{aligned}$$

- **Idea:** Choose the attribute that maximizes information gain.

SOME RIGHTS RESERVED

Restaurant Example contd.

- **Example 4.9.** Decision tree learned by DTL from the 12 examples using information gain maximization for Choose−Attribute:



- **Result:** Substantially simpler than "true" tree – a more complex hypothesis isn't justified by small amount of data.

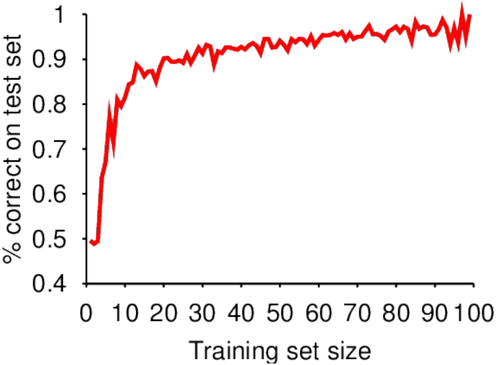
SOME RIGHTS RESERVED

26.5 Evaluating and Choosing the Best Hypothesis

SOME RIGHTS RESERVED

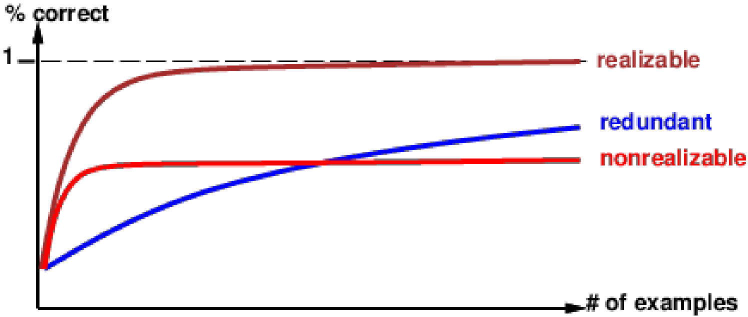
Performance measurement

- **Question:** How do we know that $h \approxeq f$? (Hume's *Problem of Induction*)
    1. Use theorems of computational/statistical learning theory.
    2. Try h on a new test set of examples. (use *same distribution over example space* as training set)
- **Definition 5.1.** The learning curve $\widehat{=}$ percentage correct on test set as a function of training set size.
- **Example 5.2.** Restaurant data; graph averaged over 20 trials



Performance measurement contd.

- **Observation 5.3.** *The learning curve depends on*
  - *realizable (can express target function) vs. non-realizable*
    non-realizability can be due to missing attributes or restricted hypothesis class (e.g., thresholded linear function)
  - *redundant expressiveness (e.g., lots of irrelevant attributes)*



Generalization and Overfitting

- **Observation:** Sometimes a learned hypothesis is more specific than the experiments warrant.
- **Definition 5.4.** We speak of overfitting, if a hypothesis *h* describes random error in the (limited) training set rather than the underlying relationship. Underfitting occurs when *h* cannot capture the underlying trend of the data.
- **Qualitatively:** Overfitting increases with the size of hypothesis space and the number of attributes, but decreases with number of examples.
- **Idea:** Combat overfitting by "generalizing" decision trees computed by DTL.

SOME RIGHTS RESERVED

Decision Tree Pruning

- **Idea:** Combat overfitting by "generalizing" decision trees ⇝ prune "irrelevant" nodes.
- **Definition 5.5.** For decision tree pruning repeat the following on a learned decision tree:
  - Find a terminal test node n (only result leaves as children)
  - If test is irrelevant, i.e. has low information gain, prune it by replacing n by with a leaf node.
- **Question:** How big should the information gain be to split (⇝ keep) a node?
- **Idea:** Use a statistical significance test.
- **Definition 5.6.** A result has statistical significance, if the probability they could arise from the null hypothesis (i.e. the assumption that there is no underlying pattern) is very low (usually 5%).

SOME RIGHTS RESERVED

Determining Attribute Irrelevance

- For decision tree pruning, the null hypothesis is that the attribute is irrelevant.
- Compute the probability that the example distribution (p positive, n negative) for a terminal node deviates from the expected distribution under the null hypothesis.
- For an attribute A with d values, compare the actual numbers p_k and n_k in each subset s_k with the expected numbers (expected if A is irrelevant) $\widehat{p}_k = p \cdot \frac{p_k+n_k}{p+n}$ and $\widehat{n}_k = n \cdot \frac{p_k+n_k}{p+n}$.
- A convenient measure of the total deviation is (sum of squared errors)

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \widehat{p}_k)^2}{\widehat{p}_k} + \frac{(n_k - \widehat{n}_k)^2}{\widehat{n}_k}$$

- **Lemma 5.7 (Neyman-Pearson).** *Under the null hypothesis, the value of Δ is distributed according to the χ^2 distribution with $d-1$ degrees of freedom.* [**NeyPea:pmtsh33**]
- **Definition 5.8.** Decision tree pruning with Pearson's χ^2 with $d-1$ degrees of freedom for Δ is called χ^2 pruning. (χ^2 values from stats library.)
- **Example 5.9.** The *type* attribute has four values, so three degrees of freedom, so $\Delta = 7.82$ would reject the null hypothesis at the 5% level.

SOME RIGHTS RESERVED

Error Rates and Cross-Validation

- **Recall:** We want to learn a hypothesis that fits the future data best.
- **Definition 5.10.** Given an inductive learning problem with a set of examples $T \subseteq AB$, we define the error rate of a hypothesis $h \in \mathcal{H}$ as the fraction of errors:

$$\frac{|\{\langle x, y\rangle \in T \mid h(x) \neq y\}|}{|T|}$$

- **Caveat:** A low error rate on the training set does not mean that a hypothesis generalizes well.
- **Idea:** Do not use homework questions in the exam.
- **Definition 5.11.** The practice of splitting the data available for learning into
  1. a training set from which the learning algorithm produces a hypothesis h and
  2. a test set, which is used for evaluating h

  is called holdout cross validation. (no peeking at test set allowed)

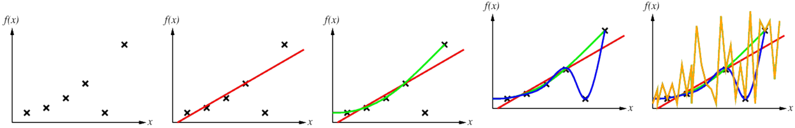
SOME RIGHTS RESERVED

Error Rates and Cross-Validation

- **Question:** What is a good ratio between training set and test set size?
  - small training set ⇝ poor hypothesis.
  - small test set ⇝ poor estimate of the accuracy.
- **Definition 5.12.** In k fold cross validation, we perform k rounds of learning, each with $1/k$ of the data as test set and average over the k error rates.
- **Intuition:** Each example does double duty: for training and testing.
- $k = 5$ and $k = 10$ are popular ⇝ good accuracy at k times computation time.
- **Definition 5.13.** If $k = |\mathrm{dom}(f)|$, then k fold cross validation is called leave one out cross validation (LOOCV).

Model Selection

- **Definition 5.14.** The model selection problem is to determine – given data – a good hypothesis space.
- **Example 5.15.** What is the best polynomial degree to fit the data



- **Observation 5.16.** *We can solve the problem of "learning from observations f" in a two-part process:*
  1. *model selection determines a hypothesis space $\mathcal{H}$,*
  2. *optimization solves the induced inductive learning problem.*
- **Idea:** Solve the two parts together by iteration over "size". (they inform each other)
- **Problem:** Need a notion of "size" ⇜ e.g. number of nodes in a decision tree.
- **Concrete Problem:** Find the "size" that best balances overfitting and underfitting to optimize test set accuracy.

SOME RIGHTS RESERVED

Model Selection Algorithm (Wrapper)

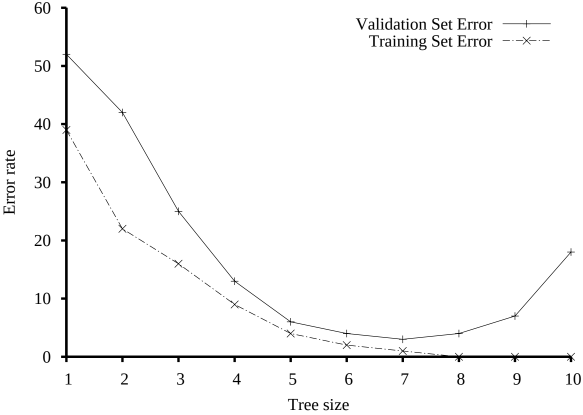
- **Definition 5.17.** The model selection algorithm (MSA) jointly optimizes model selection and optimization by partitioning and cross-validation:

```
function CROSS−VALIDATION−WRAPPER($Learner$,$k$,$examples$) returns a hypothesis
  local variables: $errT$, an array, indexed by size, storing training−set error rates
                   $errV$, an array, indexed by size, storing validation−set error rates
  for size = 1 to $\infty$ do
    $errT[size]$, $errV[size]$ := CROSS−VALIDATION($Learner$,$size$,$k$,$examples$)
    if $errT$ has converged then do
      $best\_size$ := the value of size with minimum $errV[size]$
      return Learner($best\_size$,$examples$)

function CROSS−VALIDATION($Learner$,$size$,$k$,$examples$) returns two values:
        average training set error rate, average validation set error rate
  $fold\_errT$ := 0; $fold\_errV$ := 0
  for fold = 1 to $k$ do
      $training\_set$, $validation\_set$ := PARTITION($examples$,$fold$,$k$)
      $h$ := Learner($size$,$training\_set$)
      $fold\_errT$ := $fold\_errT$ + ERROR−RATE($h$,$training\_set$)
      $fold\_errV$ := $fold\_errV$ + ERROR−RATE($h$,$validation\_set$)
  return $fold\_errT/k$, $fold\_errV/k$

function PARTITION($examples$,$fold$,$k$) returns two sets:
  a validation set of size $|examples|/k$ and the rest; the split is different for each $fold$ value
```

SOME RIGHTS RESERVED

Error Rates on Training/Validation Data

- **Example 5.18 (An Error Curve for Restaurant Decision Trees).**
  Modify DTL to be breadth-first, information gain sorted, stop after *k* nodes.



Stops when training set error rate converges, choose optimal tree for validation curve. (here a tree with 7 nodes)

SOME RIGHTS RESERVED

From Error Rates to Loss Functions

- **So far** we have been minimizing error rates. (better than maximizing ☺)
- **Example 5.19 (Classifying Spam).** It is much worse to classify ham (legitimate mails) as spam than vice versa. (message loss)
- **Recall Rationality:** Decision-makers should maximize expected utility (MEU).
- **So:** Machine learning should maximize "utility". (not only minimize error rates)
- machine learning traditionally deals with utilities in form of "loss functions".
- **Definition 5.20.** The loss function L is defined by setting $L(x, y, \widehat{y})$ to be the amount of utility lost by prediction $h(x) = \widehat{y}$ instead of $f(x) = y$. If L is independent of x, we often use $L(y, \widehat{y})$.
- **Example 5.21.** $L(spam, ham) = 1$, while $L(ham, spam) = 10$.

Generalization Loss

- **Note:** $L(y,y)=0$. (no loss if you are exactly correct)
- **Definition 5.22 (Popular general loss functions).**

absolute value loss	$L_1(y,\widehat{y}):=\lvert y-\widehat{y}\rvert$	small errors are good
squared error loss	$L_2(y,\widehat{y}):=(y-\widehat{y})^2$	ditto, but differentiable
0/1 loss	$L_{0/1}(y,\widehat{y}):=0$, if $y=\widehat{y}$, else 1	error rate

- **Idea:** Maximize expected utility by choosing hypothesis h that minimizes expected loss over all $(x,y)\in f$.
- **Definition 5.23.** Let $\mathcal{E}$ be the set of all possible examples and $\mathbb{P}(X,Y)$ the prior probability distribution over its components, then the expected generalization loss for a hypothesis h with respect to a loss function L is

$$\mathrm{GenLoss}_L(h):=\sum_{(x,y)\in\mathcal{E}} L(y,h(x))\cdot P(x,y)$$

and the best hypothesis $h^* := \underset{h\in\mathcal{H}}{\mathrm{argmin}}\ \mathrm{GenLoss}_L(h)$.

SOME RIGHTS RESERVED

Empirical Loss

- **Problem:** $\mathrm{P}(X, Y)$ is unknown ⇝ learner can only estimate generalization loss:
- **Definition 5.24.** Let L be a loss function and E a set of examples with $\#(E) = N$, then we call

$$\mathrm{EmpLoss}_{L,E}(h) := \frac{1}{N}\left(\sum_{(x,y)\in E} L(y, h(x))\right)$$

  the empirical loss and $\widehat{h}^* := \underset{h\in\mathcal{H}}{\mathrm{argmin}}\ \mathrm{EmpLoss}_{L,E}(h)$ the estimated best hypothesis.
- There are four reasons why $\widehat{h}^*$ may differ from f:
  1. Realizablility: then we have to settle for an approximation $\widehat{h}^*$ of f.
  2. Variance: different subsets of f give different $\widehat{h}^*$ ⇝ more examples.
  3. Noise: if f is non deterministic, then we cannot expect perfect results.
  4. Computational complexity: if $\mathcal{H}$ is too large to systematically explore, we make due with subset and get an approximation.

SOME RIGHTS RESERVED

Regularization

- **Idea:** Directly use empirical loss to solve model selection. (finding a good $\mathcal{H}$) Minimize the weighted sum of empirical loss and hypothesis complexity.(to avoid overfitting).
- **Definition 5.25.** Let $\lambda \in \mathbb{R}$, $h \in \mathcal{H}$, and E a set of examples, then we call

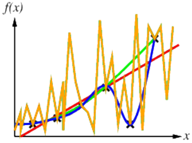
  $$\mathrm{Cost}_{L,E}(h) := \mathrm{EmpLoss}_{L,E}(h) + \lambda \mathrm{Complexity}(h)$$

  the total cost of h on E.
- **Definition 5.26.** The process of finding a total cost minimizing hypothesis

  $$\widehat{h}^* := \underset{h \in \mathcal{H}}{\mathrm{argmin}}\, \mathrm{Cost}_{L,E}(h)$$

  is called regularization; Complexity is called the regularization function or hypothesis complexity.
- **Example 5.27 (Regularization for Polynomials).**

  A good regularization function for polynomials is the sum of squares of exponents. ⇝ keep away from wriggly curves!



SOME RIGHTS RESERVED

Minimal Description Length

- **Remark:** In regularization, empirical loss and hypothesis complexity are not measured in the same scale $\rightsquigarrow$ λ mediates between scales.
- **Idea:** Measure both in the same scale $\rightsquigarrow$ use information content, i.e. in bits.
- **Definition 5.28.** Let $h \in \mathcal{H}$ be a hypothesis and E a set of examples, then the description length of (h,E) is computed as follows:
  1. encode the hypothesis as a Turing machine program, count bits.
  2. count data bits:
     - correctly predicted example $\rightsquigarrow$ $0\mathrm{b}$
     - incorrectly predicted example $\rightsquigarrow$ according to size of error.

  The minimum description length or MDL hypothesis minimizes the total number of bits required.
- This works well in the limit, but for smaller problems there is a difficulty in that the choice of encoding for the program affects the outcome.
  - e.g., how best to encode a decision tree as a bit string?

The Scale of Machine Learning

- Traditional methods in statistics and early machine learning concentrated on small-scale learning (50-5000 examples)
  - Generalization error mostly comes from
    - approximation error of not having the true f in the hypothesis space
    - estimation error of too few training examples to limit variance.
- In recent years there has been more emphasis on large-scale learning. (millions of examples)
  - Generalization error is dominated by limits of computation
    - there is enough data and a rich enough model that we could find an h that is very close to the true f,
    - but the computation to find it is too complex, so we settle for a sub-optimal approximation.
  - Hardware advances (GPU farms, Amazon EC2, Google Data Centers, ...) help.

SOME RIGHTS RESERVED

26.6 Computational Learning Theory

SOME RIGHTS RESERVED

A (General) Theory of Learning?

- **Main Question:** How can we be sure that our learning algorithm has produced a hypothesis that will predict the correct value for previously unseen inputs?
- **Formally:** How do we know that the hypothesis h is close to the target function f if we don't know what f is?
- **Other - more recent - Questions:**
  - How many examples do we need to get a good h?
  - What hypothesis space $\mathcal{H}$ should we use?
  - If the $\mathcal{H}$ is very complex, can we even find the best h, or do we have to settle for a local maximum in $\mathcal{H}$.
  - How complex should h be?
  - How do we avoid overfitting?
- "Computational Learning Theory" tries to answer these using concepts from AI, statistics, and theoretical CS.

SOME RIGHTS RESERVED

PAC Learning

- **Basic idea of Computational Learning Theory:**
  - Any hypothesis h that is seriously wrong will almost certainly be "found out" with high probability after a small number of examples, because it will make an incorrect prediction.
  - Thus, if h is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong.
  - $\leadsto$ h is probably approximately correct.
- **Definition 6.1.** Any learning algorithm that returns hypotheses that are probably approximately correct is called a PAC learning algorithm.
- Derive performance bounds for PAC learning algorithms in general, using the
- **Stationarity Assumption (again):** We assume that the set $\mathcal{E}$ of possible examples is IID $\leadsto$ we have a fixed distribution $P(E) = P(X, Y)$ on examples.
- **Simplifying Assumptions:** f is a function (deterministic) and $f \in \mathcal{H}$.

SOME RIGHTS RESERVED

PAC Learning

- Start with PAC theorems for Boolean functions, for which $L_{0/1}$ is appropriate.
- **Definition 6.2.** The error rate $\mathrm{error}(h)$ of a hypothesis h is the probability that h misclassifies a new example.

$$\mathrm{error}(h) := \mathrm{GenLoss}_{L_{0/1}}(h) = \sum_{(x,y)\in\mathcal{E}} L_{0/1}(y, h(x)) \cdot P(x, y)$$

- **Intuition:** $\mathrm{error}(h)$ is the probability that h misclassifies a new example.
- This is the same quantity as measured in the learning curves above.
- **Definition 6.3.** A hypothesis h is called approximatively correct, iff $\mathrm{error}(h) \leq \epsilon$ for some small $\epsilon > 0$.
  We write $\mathcal{H}_b := \{h \in \mathcal{H} \mid \mathrm{error}(h) > \epsilon\}$ for the "seriously bad" hypotheses.

SOME RIGHTS RESERVED

Sample Complexity

- Let's compute the probability that $h_b \in \mathcal{H}_b$ is consistent with the first N examples.
- We know $\mathrm{error}(h_b) > \epsilon$
  $\rightsquigarrow$ $P(h_b$ agrees with N examples$) \leq (1-\epsilon)^N$. (independence)
  $\rightsquigarrow$ $P(\mathcal{H}_b$ contains consistent hyp.$) \leq |\mathcal{H}_b| \cdot (1-\epsilon)^N \leq |\mathcal{H}| \cdot (1-\epsilon)^N$. ($\mathcal{H}_b \subseteq \mathcal{H}$)
  $\rightsquigarrow$ to bound this by a small δ, show the algorithm $N \geq \frac{1}{\epsilon} \cdot (\log_2(\frac{1}{\delta}) + \log_2(|\mathcal{H}|))$ examples.
- **Definition 6.4.** The number of required examples as a function of ϵ and δ is called the sample complexity of $\mathcal{H}$.
- **Example 6.5.** If $\mathcal{H}$ is the set of n-ary Boolean functions, then $|\mathcal{H}| = 2^{2^n}$.
  $\rightsquigarrow$ sample complexity grows with $\mathcal{O}(\log_2(2^{2^n})) = \mathcal{O}(2^n)$.
  There are 2^n possible examples,
  $\rightsquigarrow$ PAC learning for Boolean functions needs to see (nearly) all examples.

SOME RIGHTS RESERVED

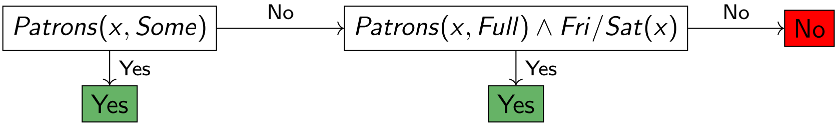
Escaping Sample Complexity

- **Problem:** PAC learning for Boolean functions needs to see (nearly) all examples.
  - $\mathcal{H}$ contains enough hypotheses to classify any given set of examples in all possible ways.
  - In particular, for any set of N examples, the set of hypotheses consistent with those examples contains equal numbers of hypotheses that predict x_{N+1} to be positive and hypotheses that predict x_{N+1} to be negative.
- **Idea/Problem:** restrict the $\mathcal{H}$ in some way (but we may lose realizability)
- **Three Ways out of this Dilemma:**
  1. bring prior knowledge into the problem. (???)
  2. prefer simple hypotheses. (e.g. decision tree pruning)
  3. focus on "learnable subsets" of $\mathcal{H}$. (next)

SOME RIGHTS RESERVED

PAC Learning: Decision Lists

- **Idea:** Apply PAC learning to a "learnable hypothesis space".
- **Definition 6.6.** A decision list consists of a sequence of tests, each of which is a conjunction of literals.
  - If a test succeeds when applied to an example description, the decision list specifies the value to be returned.
  - If the test fails, processing continues with the next test in the list.
- **Remark:** Like decision trees, but restricted branching, but more complex tests.
- **Example 6.7 (A decision list for the Restaurant Problem).**



- **Lemma 6.8.** *Given arbitrary size conditions, decision lists can represent arbitrary Boolean functions.*
- This directly defeats our purpose of finding a "learnable subset" of $\mathcal{H}$.

SOME RIGHTS RESERVED

Decision Lists: Learnable Subsets (Size-Restricted Cases)

- **Definition 6.9.** The set of decision lists where tests are of conjunctions of at most k literals is denoted by $k{-}\mathrm{DL}$.
- **Example 6.10.** The decision list from **??** is in $2{-}\mathrm{DL}$.
- **Observation 6.11.** *$k{-}\mathrm{DL}$ contains $k{-}\mathrm{DT}$, the set of decision trees of depth at most k.*
- **Definition 6.12.** We denote the set of $k{-}\mathrm{DL}$ decision lists with at most n Boolean attributes with $k{-}\mathrm{DL}(n)$. The set of conjunctions of at most k literals over n attributes is written as $\mathrm{Conj}(k,n)$.
- Decision lists are constructed of optional yes/no tests, so there are at most $3^{|\mathrm{Conj}(k,n)|}$ distinct sets of component tests. Each of these sets of tests can be in any order, so $|k{-}\mathrm{DL}(n)| \leq 3^{|\mathrm{Conj}(k,n)|} \cdot |\mathrm{Conj}(k,n)|!$

SOME RIGHTS RESERVED

Decision Lists: Learnable Subsets (Sample Complexity)

- The number of conjunctions of k literals from n attributes is given by

$$|\mathrm{Conj}(k,n)| = \sum_{i=1}^{k} \binom{2n}{i}$$

thus $|\mathrm{Conj}(k,n)|=\mathcal{O}(n^k)$. Hence, we obtain (after some work)

$$|k{-}\mathrm{DL}(n)|=2^{\mathcal{O}(n^k \log_2(n^k))}$$

- Plug this into the equation for the sample complexity: $N \geq \frac{1}{\epsilon} \cdot (\log_2(\frac{1}{\delta}) + \log_2(|\mathcal{H}|))$ to obtain

$$N \geq \frac{1}{\epsilon} \cdot (\log_2(\frac{1}{\delta}) + \log_2(\mathcal{O}(n^k \log_2(n^k))))$$

- **Intuitively:** Any algorithm that returns a consistent decision list will PAC learn a $k{-}\mathrm{DL}$ function in a reasonable number of examples, for small k.

SOME RIGHTS RESERVED

Decision Lists Learning

- **Idea:** Use a greedy search algorithm that repeats
  1. **find** test that agrees exactly with some subset E of the training set,
  2. **add** it to the decision list under construction and removes E,
  3. **construct** the remainder of the DL using just the remaining examples,

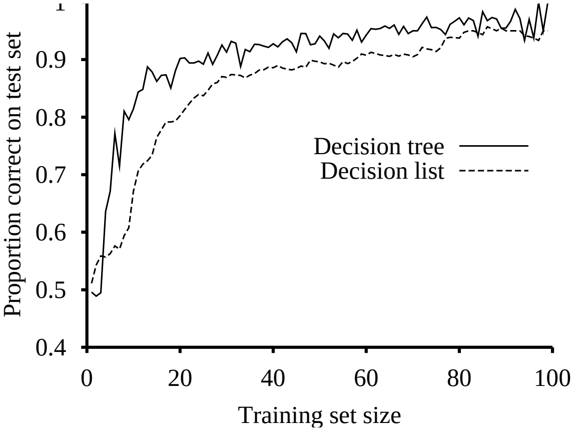
  until there are no examples left.
- **Definition 6.13.** The following algorithm performs decision list learning

```
function DLL($E$) returns a decision list, or failure
  if $E$ is empty then return (the trivial decision list) No
  $t$ := a test that matches a nonempty subset $E_t$ of $E$
            such that the members of $E_t$ are all positive or all negative
  if there is no such $t$ then return failure
  if the examples in $E_t$ are positive then $o$ := Yes else $o$ := No
  return a decision list with initial test $t$ and outcome $o$ and remaining tests given by
      DLL($E\backslash E_t$)
```

SOME RIGHTS RESERVED

Decision Lists Learning in Comparison

- **Learning curves:** for DLL (and DTL for comparison)



- **Upshot:** The simpler DLL works quite well!

SOME RIGHTS RESERVED

26.7 Regression and Classification with Linear Models

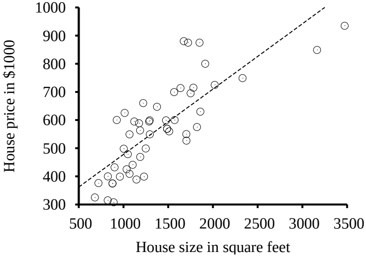
SOME RIGHTS RESERVED

Univariate Linear Regression

- **Definition 7.1.** A univariate or unary function is a function with one argument.
- **Recall:** A mapping f between vector spaces is called linear, iff it preserves rmodule/plus and rmodule/scalar multiplication, i.e. $f(\alpha \cdot v_1 + v_2) = \alpha \cdot f(v_1) + f(v_2)$.
- **Observation 7.2.** *A univariate, linear function* $f\colon \mathbb{R} \to \mathbb{R}$ *is of the form* $f(x) = w_1 x + w_0$ *for some* $w_i \in \mathbb{R}$*.*
- **Definition 7.3.** Given a vector $w := (w_0,w_1)$, we define $h_w(x) := w_1 x + w_0$.
- **Definition 7.4.** Given a set of examples $E \subseteq \mathbb{R}\times\mathbb{R}$, the task of finding h_w that best fits E is called linear regression.
- **Example 7.5.**

  Examples of house price vs. square feet in houses sold in Berkeley in July 2009.

  Also: linear function hypothesis that minimizes squared error loss $y = 0.232x + 246$.



SOME RIGHTS RESERVED

Univariate Linear Regression by Loss Minimization

- **Idea:** Minimize squared error loss over $\{(x_i,y_i)\,|\,i\leq N\}$(used already by Gauss)

$$\mathrm{Loss}(h_{\mathrm{w}})=\sum_{j=1}^{N}L_2(y_j,h_{\mathrm{w}}(x_j))=\sum_{j=1}^{N}(y_j-h_{\mathrm{w}}(x_j))^2=\sum_{j=1}^{N}(y_j-(\mathrm{w}_1x_j+\mathrm{w}_0))^2$$

  Task: find $\mathrm{w}^*:=\underset{\mathrm{w}}{\mathrm{argmin}}\,\mathrm{Loss}(h_{\mathrm{w}})$.

- **Recall:** $\sum_{j=1}^{N}(y_j-(\mathrm{w}_1x_j+\mathrm{w}_0))^2$ is minimized, when the partial derivatives wrt. the w_i are zero, i.e. when

$$\frac{\partial}{\partial \mathrm{w}_0}(\sum_{j=1}^{N}(y_j-(\mathrm{w}_1x_j+\mathrm{w}_0))^2)=0 \quad\text{and}\quad \frac{\partial}{\partial \mathrm{w}_1}(\sum_{j=1}^{N}(y_j-(\mathrm{w}_1x_j+\mathrm{w}_0))^2)=0$$

- **Observation:** These equations have a unique solution:

$$\mathrm{w}_1=\frac{N(\sum_j x_jy_j)-(\sum_j x_j)(\sum_j y_j)}{N(\sum_j {x_j}^2)-(\sum_j x_j)^2} \qquad \mathrm{w}_0=\frac{(\sum_j y_j)-\mathrm{w}_1(\sum_j x_j)}{N}$$

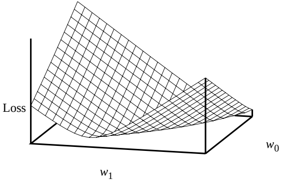
- **Remark:** Closed-form solutions only exist for linear regression, for other (differentiable) hypothesis spaces use gradient descent methods for adjusting/learning weights.

SOME RIGHTS RESERVED

A Picture of the Weight Space

- **Remark:** Many forms of learning involve adjusting weights to minimize loss.
- **Definition 7.6.** The weight space of a parametric model is the space of all possible combinations of parameters (called the weights). Loss minimization in a weight space is called weight fitting.

The weight space of univariate linear regression is $\mathbb{R}^2$.
⇝ graph the loss function over $\mathbb{R}^2$.
Note: it is convex.



-
- **Observation 7.7.** *The squared error loss function is convex for any linear regression problem ⇝ there are no local minima.*

SOME RIGHTS RESERVED

Gradient Descent Methods

- If we do not have closed form solutions for minimizing loss, we need to search.
- **Idea:** Use local search (hill climbing) methods.
- **Definition 7.8.** The gradient descent algorithm for finding a minimum of a continuous function F is hill climbing in the direction of the steepest descent, which can be computed by the partial derivatives of F.

```
function gradient−descent($F$,$w$,$\alpha$) returns a local minimum of $F$
  inputs: a differentiable function $F$ and initial weights $w$.
  loop until $w$ converges do
      for each $w_i$ do
        $w_i \longleftarrow w_i - \alpha\frac{\partial}{\partial w_i}F(w)$
      end for
  end loop
```

The parameter α is called the learning rate. It can be a fixed constant or it can decay as learning proceeds.

Gradient-Descent for Loss

- Let's try gradient descent for Loss.
- Work out the partial derivatives for one example (x,y):

$$\frac{\partial \mathrm{Loss}(\mathrm{w})}{\partial \mathrm{w}_i} = \frac{\partial (y - h_{\mathrm{w}}(x))^2}{\partial \mathrm{w}_i} = 2(y - h_{\mathrm{w}}(x))\frac{\partial (y - (\mathrm{w}_1 x + \mathrm{w}_0))}{\partial \mathrm{w}_i}$$

and thus

$$\frac{\partial \mathrm{Loss}(\mathrm{w})}{\partial \mathrm{w}_0} = -2(y - h_{\mathrm{w}}(x)) \qquad \frac{\partial \mathrm{Loss}(\mathrm{w})}{\partial \mathrm{w}_1} = -2(y - h_{\mathrm{w}}(x))x$$

Plug this into the gradient descent updates:

$$\mathrm{w}_0 \longleftarrow \mathrm{w}_0 - \alpha \cdot (-2(y - h_{\mathrm{w}}(x))) \qquad \mathrm{w}_1 \longleftarrow \mathrm{w}_1 - \alpha \cdot -2((y - h_{\mathrm{w}}(x))) \cdot x$$

SOME RIGHTS RESERVED

Gradient-Descent for Loss (continued)

- Analogously for n training examples (x_j,y_j):
- **Definition 7.9.**

$$w_0 \longleftarrow w_0 - \alpha(\sum_j -2(y_j - h_w(x_j))) \quad w_1 \longleftarrow w_1 - \alpha(\sum_j -2(y_j - h_w(x_n))x_n)$$

  These updates constitute the batch gradient descent learning rule for univariate linear regression.
- Convergence to the unique global loss minimum is guaranteed (as long as we pick α small enough) but may be very slow.
- Doing batch gradient descent on random subsets of the examples of fixed batch size n is called stochastic gradient descent (SGD). (More computationally efficient than updating for every example)

SOME RIGHTS RESERVED

Multivariate Linear Regression

- **Definition 7.10.** A multivariate or n-ary function is a function with one or more arguments.
- We can use it for multivariate linear regression.
- **Idea:** Every example $\vec{x_j}$ is an n element vector and the hypothesis space is the set of functions

$$h_{sw}(\vec{x_j}) = \mathrm{w}_0 + \mathrm{w}_1 x_{j,1} + \ldots + \mathrm{w}_n x_{j,n} = \mathrm{w}_0 + \sum_i \mathrm{w}_i x_{j,i}$$

- **Trick:** Invent $x_{j,0} := 1$ and use matrix notation:

$$h_{sw}(\vec{x_j}) = \vec{w}\cdot\vec{x_j} = \vec{w}^t\vec{x_j} = \sum_i \mathrm{w}_i x_{j,i}$$

- **Definition 7.11.** The best vector of weights, w^*, minimizes squared-error loss over the examples: $\mathrm{w}^* := \underset{\mathrm{w}}{\operatorname{argmin}} \left(\sum_j L_2(y_j)(\mathrm{w}\cdot\vec{x_j})\right)$.
- Gradient descent will reach the (unique) minimum of the loss function; the update equation for each weight w_i is

$$\mathrm{w}_i \longleftarrow \mathrm{w}_i - \alpha\left(\sum_j x_{j,i}(y_j - h_{\mathrm{w}}(\vec{x_j}))\right)$$

SOME RIGHTS RESERVED

Multivariate Linear Regression (Analytic Solutions)

- We can also solve analytically for the w^* that minimizes loss.
- Let $\vec{y}$ be the vector of outputs for the training examples, and X be the data matrix, i.e., the matrix of inputs with one n-dimensional example per row. Then the solution $w^* = (X^T X)^{-1} X^T \vec{y}$ minimizes the squared error.

Multivariate Linear Regression (Regularization)

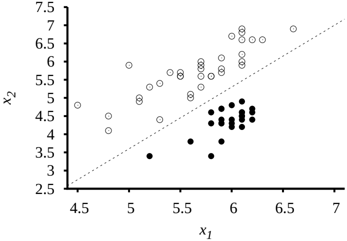
- **Remark:** Univariate linear regression does not overfit, but in the multivariate case there might be "redundant dimensions" that result in overfitting.
- **Idea:** Use regularization with a complexity function based on weights.
- **Definition 7.12.** $\mathrm{Complexity}(h_{\mathrm{w}}) = L_q(\mathrm{w}) = \sum_i |\mathrm{w}_i|^q$
- **Caveat:** Do not confuse this with the loss functions L_1 and L_2.
- **Problem:** Which q should we pick? (L_1 and L_2 minimize sum of absolute values/squares)
- **Answer:** It depends on the application.
- **Remark:** L_1-regularization tends to produce a sparse model, i.e. it sets many weights to 0, effectively declaring the corresponding attributes to be irrelevant. Hypotheses that discard attributes can be easier for a human to understand, and may be less likely to overfit. (see [**RusNor:AIMA03**])

SOME RIGHTS RESERVED

Linear Classifiers with a hard Threshold

- **Idea:** The result of linear regression can be used for classification.
- **Example 7.13 (Nuclear Test Ban Verification).**

Plots of seismic data parameters: body wave magnitude x_1 vs. surface wave magnitude x_2. White: earthquakes, black: underground explosions
Also: h_{w^*} as a decision boundary $x_2 = 17x_1 - 4.9$.



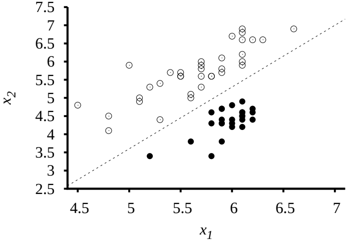
SOME RIGHTS RESERVED

Linear Classifiers with a hard Threshold

- **Idea:** The result of linear regression can be used for classification.
- **Example 7.16 (Nuclear Test Ban Verification).**

Plots of seismic data parameters: body wave magnitude x_1 vs. surface wave magnitude x_2. White: earthquakes, black: underground explosions

Also: h_{w^*} as a decision boundary $x_2 = 17x_1 - 4.9$.

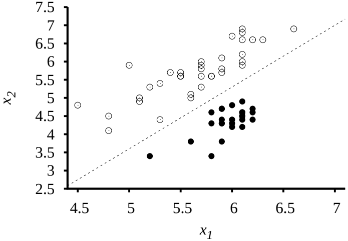


- **Definition 7.17.** A decision boundary is a line (or a surface, in higher dimensions) that separates two classes of points. A linear decision boundary is called a linear separator and data that admits one are called linearly separable.
- **Example 7.18 (Nuclear Tests continued).** The linear separator for **??**is defined by $-4.9 + 1.7x_1 - x_2 = 0$, explosions are characterized by $-4.9 + 1.7x_1 - x_2 > 0$, earthquakes by $-4.9 + 1.7x_1 - x_2 < 0$.

Linear Classifiers with a hard Threshold

- **Idea:** The result of linear regression can be used for classification.
- **Example 7.19 (Nuclear Test Ban Verification).**

Plots of seismic data parameters: body wave magnitude x_1 vs. surface wave magnitude x_2. White: earthquakes, black: underground explosions
Also: h_{w^*} as a decision boundary $x_2 = 17x_1 - 4.9$.



- **Definition 7.20.** A decision boundary is a line (or a surface, in higher dimensions) that separates two classes of points. A linear decision boundary is called a linear separator and data that admits one are called linearly separable.
- **Example 7.21 (Nuclear Tests continued).** The linear separator for **??**is defined by $-4.9 + 1.7x_1 - x_2 = 0$, explosions are characterized by $-4.9 + 1.7x_1 - x_2 > 0$, earthquakes by $-4.9 + 1.7x_1 - x_2 < 0$.
- **Useful Trick:** If we introduce dummy coordinate $x_0 = 1$, then we can write the classification hypothesis as $h_{\mathrm{w}}(\mathrm{x}) = 1$ if $\mathrm{w}\cdot\mathrm{x} > 0$ and 0 otherwise.

Linear Classifiers with a hard Threshold (Perceptron Rule)

- So $h_{\mathrm{w}}(\mathrm{x}) = 1$ if $\mathrm{w}\cdot\mathrm{x} > 0$ and 0 otherwise is well-defined, how to choose w?
- Think of $h_{\mathrm{w}}(\mathrm{x}) = \mathcal{T}(\mathrm{w}\cdot\mathrm{x})$, where $\mathcal{T}(z) = 1$, if $z > 0$ and $\mathcal{T}(z) = 0$ otherwise. We call $\mathcal{T}$ a threshold function.
- **Problem:** $\mathcal{T}$ is not differentiable and $\frac{\partial\mathcal{T}}{\partial z} = 0$ where defined $\leadsto$
  - No closed-form solutions by setting $\frac{\partial\mathcal{T}}{\partial z} = 0$ and solving.
  - Gradient-descent methods in weight-space do not work either.
- We can learn weights by iterating over the following rule:
- **Definition 7.22.** Given an example (x,y), the perceptron learning rule is

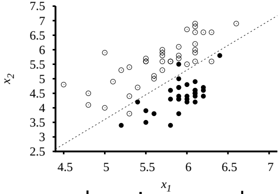
$$\mathrm{w}_i \longleftarrow \mathrm{w}_i + \alpha\cdot(y - h_{\mathrm{w}}(\mathrm{x}))\cdot x_i$$

- as we are considering 0/1 classification, there are three possibilities:
  1. If $y = h_{\mathrm{w}}(\mathrm{x})$, then w_i remains unchanged.
  2. If $y = 1$ and $h_{\mathrm{w}}(\mathrm{x}) = 0$, then w_i is in/decreased if x_i is positive/negative. (we want to make $\mathrm{w}\cdot\mathrm{x}$ bigger so that $\mathcal{T}(\mathrm{w}\cdot\mathrm{x}) = 1$)
  3. If $y = 0$ and $h_{\mathrm{w}}(\mathrm{x}) = 1$, then w_i is de/increased if x_i is positive/negative. (we want to make $\mathrm{w}\cdot\mathrm{x}$ smaller so that $\mathcal{T}(\mathrm{w}\cdot\mathrm{x}) = 0$)

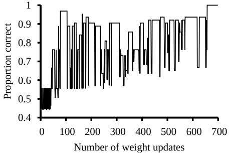
SOME RIGHTS RESERVED

Learning Curves for Linear Classifiers (Perceptron Rule)

- **Example 7.23.**
  Learning curves (plots of total training set accuracy vs. number of iterations) for the perceptron rule on the earthquake/explosions data.

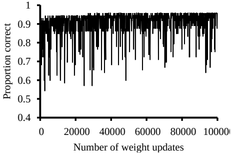


original data



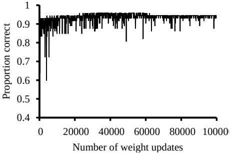
messy convergence
700 iterations

noisy, non-separable data



convergence failure
100,000 iterations

learning rate decay
$\alpha(t) = 1000/(1000 + t)$



slow convergence
100,000 iterations

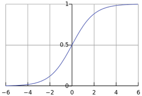
- **Theorem 7.24.** *Finding the minimal-error hypothesis is NP-hard, but possible with learning rate decay.*

SOME RIGHTS RESERVED

Linear Classification with Logistic Regression

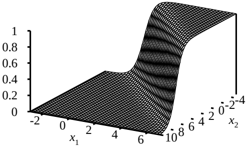
- **So far:** Passing the output of a linear function through a threshold function $\mathcal{T}$ yields a linear classifier.
- **Problem:** The hard nature of $\mathcal{T}$ brings problems:
  - $\mathcal{T}$ is not differentiable nor continuous $\rightsquigarrow$ learning via perceptron rule becomes unpredictable.
  - $\mathcal{T}$ is “overly precise” near the boundary $\leftsquigarrow$ need more graded judgments.
- **Idea:** Soften the threshold, approximate it with a differentiable function.

  We use the standard logistic function $l(x) = \frac{1}{1+e^{-x}}$
  So we have $h_{\mathbf{w}}(\mathbf{x}) = l(\mathbf{w}\cdot\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x})}}$



- **Example 7.25 (Logistic Regression Hypothesis in Weight Space).**

  Plot of a logistic regression hypothesis for the earthquake/explosion data. The value at $(\mathrm{w}_0,\mathrm{w}_1)$ is the probability of belonging to the class labeled 1.



We speak of the cliff in the classifier intuitively.

Logistic Regression

- **Definition 7.26.** The process of weight fitting in $h_{\mathrm{w}}(\mathrm{x}) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x})}}$ is called logistic regression.
- There is no easy closed form solution, but gradient descent is straightforward,
- As our hypotheses have continuous output, use the squared error loss function L_2.
- For an example (x,y) we compute the partial derivatives: (via chain rule)

$$\begin{aligned}
\frac{\partial}{\partial \mathrm{w}_i} L_2(\mathrm{w}) &= \frac{\partial}{\partial \mathrm{w}_i}((y - h_{\mathrm{w}}(\mathrm{x}))^2) \\
&= 2 \cdot h_{\mathrm{w}}(\mathrm{x}) \cdot \frac{\partial}{\partial \mathrm{w}_i}(y - h_{\mathrm{w}}(\mathrm{x})) \\
&= -2 \cdot h_{\mathrm{w}}(\mathrm{x}) \cdot l'(\mathrm{w}\cdot\mathrm{x}) \cdot \frac{\partial}{\partial \mathrm{w}_i}(\mathrm{w}\cdot\mathrm{x}) \\
&= -2 \cdot h_{\mathrm{w}}(\mathrm{x}) \cdot l'(\mathrm{w}\cdot\mathrm{x}) \cdot x_i
\end{aligned}$$

SOME RIGHTS RESERVED

Logistic Regression (continued)

- The derivative of the logistic function satisfies $l'(z) = l(z)(1 - l(z))$, thus

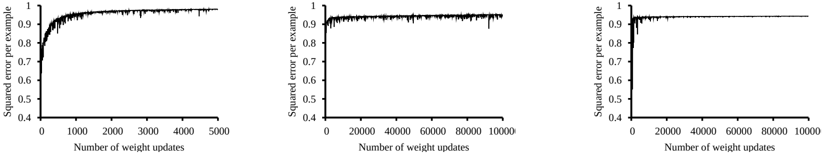
$$l'(\mathrm{w}\cdot\mathrm{x}) = l(\mathrm{w}\cdot\mathrm{x})(1 - l(\mathrm{w}\cdot\mathrm{x})) = h_{\mathrm{w}}(\mathrm{x})(1 - h_{\mathrm{w}}(\mathrm{x}))$$

- **Definition 7.27.** The rule for logistic update (weight update for minimizing the loss) is

$$\mathrm{w}_i \longleftarrow \mathrm{w}_i + \alpha \cdot (y - h_{\mathrm{w}}(\mathrm{x})) \cdot h_{\mathrm{w}}(\mathrm{x}) \cdot (1 - h_{\mathrm{w}}(\mathrm{x})) \cdot x_i$$

- **Example 7.28 (Redoing the Learning Curves).**

original data | noisy, non-separable data | learning rate decay $\alpha(t) = 1000/(1000 + t)$



messy convergence 5000 iterations | convergence failure 100,000 iterations | slow convergence 100,000 iterations

- **Upshot:** Logistic update seems to perform better than perceptron update.

SOME RIGHTS RESERVED

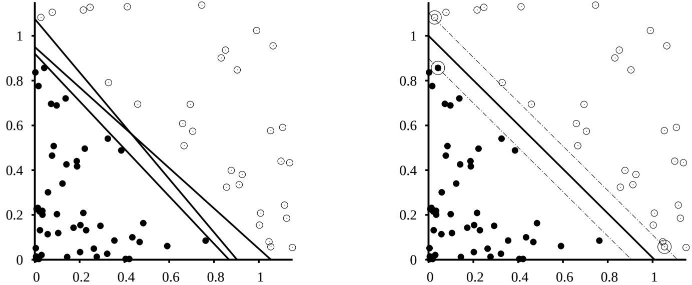
26.8 Support Vector Machines

SOME RIGHTS RESERVED

Support Vector Machines

Definition 8.1. Given a linearly separable data set E the maximum margin separator is the linear separator s that maximizes the margin, i.e. the distance of the E from s.

Example 8.2. All lines on the left are valid linear separators:



We expect the maximum margin separator on the right to generalize best

Note: To find the maximum margin separator, we only need to consider the innermost points (circled above).

Support Vector Machines (contd.)

Definition 8.3. Support-vector machines (SVMs; also support-vector networks) are supervised learning models for classification and regression.
SVMs construct a maximum margin separator by prioritizing critical examples (support vectors).

SVMs are still one of the most popular approaches for "off-the-shelf" supervised learning.

Setting:

- We have a training set $E = \{\langle x_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle\}$ where $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$ (instead of $\{1, 0\}$)
- The goal is to find a *hyperplane* in $\mathbb{R}^p$ that maximally separates the two classes (i.e. $y_i = -1$ from $y_i = 1$)

Remember A *hyperplane* can be represented as the set $\{x \mid (\mathrm{w} \cdot x) + b = 0\}$ for some vector w and scalar b. (w is orthogonal to the plane, b determines the offset from the origin)

SOME RIGHTS RESERVED

Finding the Maximum Margin Separator (Separable Case)

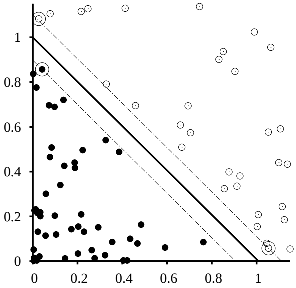
Idea: The margin is bounded by the two hyperplanes described by $\{\mathrm{x} \,|\, (\mathrm{w}\cdot\mathrm{x}) + b + 1 = 0\}$ (lower boundary) and $\{\mathrm{x} \,|\, (\mathrm{w}\cdot\mathrm{x}) + b - 1 = 0\}$ (upper boundary).

$\Rightarrow$ The distance between them is $\frac{2}{\|\mathrm{w}\|_2}$.

Constraints: To maximize the margin, minimize $\|\mathrm{w}\|_2$ while keeping x_i out of the margin: $(\mathrm{w}\cdot x_i) + b \geq 1$ for $y_i = 1$ and $(\mathrm{w}\cdot x_i) + b \leq -1$ for $y_i = -1$

$\rightsquigarrow$ $y_i((\mathrm{w}\cdot x_i) - b) \geq 1$ for $1 \leq i \leq n$.

$\rightsquigarrow$ This is an optimization problem.



Theorem 8.4 (SVM equation). *Let*

$$\alpha = \underset{\alpha}{\operatorname{argmax}} \left(\sum_j \alpha_j - \frac{1}{2}\left(\sum_{j,k} \alpha_j \alpha_k y_j y_k (x_j \cdot x_k)\right)\right)$$ *under the constraints* $\alpha_j \geq 0$ *and* $\sum_j \alpha_j y_j = 0$.

The maximum margin separator is given by $\mathrm{w} = \sum_j \alpha_j x_j$ *and* $b = \mathrm{w}\cdot x_i - y_i$ *for any* x_i *where* $\alpha_i \neq 0$.

Proof sketch: By the duality principle for optimization problems

Finding the Maximum Margin Separator (Separable Case)

$$\alpha = \underset{\alpha}{\operatorname{argmax}} \left(\sum_j \alpha_j - \frac{1}{2}\left(\sum_{j,k} \alpha_j \alpha_k y_j y_k (x_j \cdot x_k)\right)\right), \text{where } \alpha_j \geq 0, \quad \sum_j \alpha_j y_j = 0$$

Important Properties:

- The weights α_j associated with each data point are zero except at the support vectors (the points closest to the separator),
- The expression is convex ⇝ the single global maximum can found efficiently,
- Data enter the expression only in the form of dot products of point pairs ⇝ once the optimal α_i have been calculated, we have $h(x) = \operatorname{sign}(\sum_j \alpha_j y_j (x \cdot x_j) - b)$
- There are good software packages for solving such quadratic programming optimizations

SOME RIGHTS RESERVED

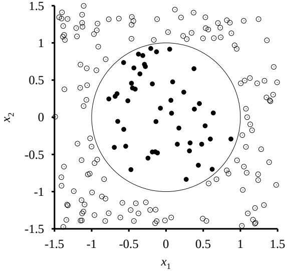
Support Vector Machines (Kernel Trick)

What if the data is not linearly separable?

Idea: Transform the data into a *feature space* where they are.

Definition 8.5. A feature for data in $\mathbb{R}^p$ is a function $\mathbb{R}^p \to \mathbb{R}^q$.

Example 8.6 (Projecting Up a Non-Separable Data Set).
The true decision boundary is ${x_1}^2 + {x_2}^2 \leq 1$.



$\rightsquigarrow$ use the feature "distance from center"

SOME RIGHTS RESERVED

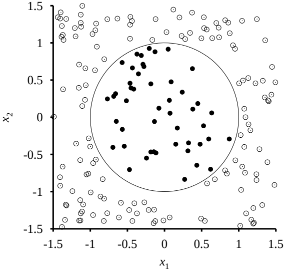
Support Vector Machines (Kernel Trick continued)

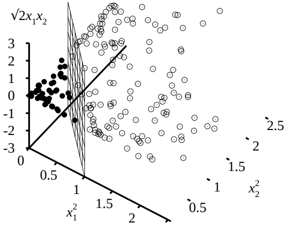
Idea: Replace $x_i \cdot x_j$ by some other product on the feature space in the SVM equation

Definition 8.7. A kernel function is a function $K \colon \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ of the form $K(x_1, x_2) = \langle F(x_1), F(x_2) \rangle$ for some feature F and inner product $\langle \cdot, \cdot \rangle$ on the codomain of F.

Smart choices for a kernel function often allow us to compute $K(x_i, x_j)$ without needing to compute F at all.

Example 8.8. If we encode the distance from the center as the feature $F(x) = \langle {x_1}^2, {x_2}^2, \sqrt{2} x_1 x_2 \rangle$ and define the kernel function as $K(x_i, x_j) = F(x_i) \cdot F(x_j)$, then this simplifies to $K(x_i, x_j) = (x_i \cdot x_j)^2$





Support Vector Machines (Kernel Trick continued)

Generally: We can learn non-linear separators by solving

$$\underset{\alpha}{\operatorname{argmax}}\,(\sum_j \alpha_j - \frac{1}{2}(\sum_{j,k} \alpha_j \alpha_k y_j y_k K(\mathrm{x}_j, \mathrm{x}_k)))$$

where K is a kernel function

Definition 8.9. Let $X = \{x_1, \ldots, x_n\}$. A symmetric function $K\colon X \times X \to \mathbb{R}$ is called positive definite iff the matrix $K_{i,j} = K(x_i, x_j)$ is a positive definite matrix.

Theorem 8.10 (Mercer's Theorem). *Every positive definite function K on X is a kernel function on X for some feature F.*

Definition 8.11. The function $K(\mathrm{x}_j, \mathrm{x}_k) = (1 + (\mathrm{x}_j \cdot \mathrm{x}_j))^d$ is a kernel function corresponding to a feature space whose dimension is exponential in d. It is called the polynomial kernel.

SOME RIGHTS RESERVED

26.9 Artificial Neural Networks

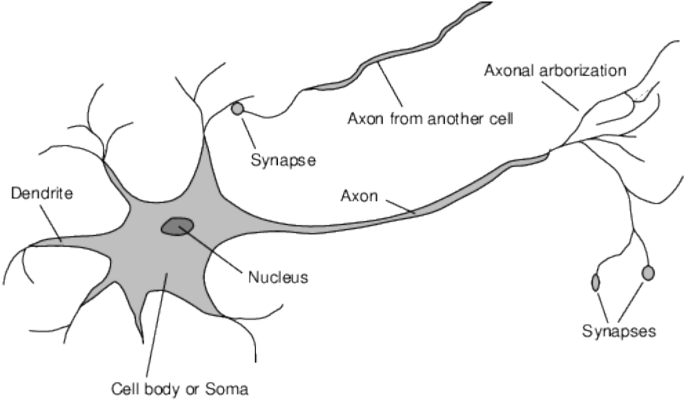
SOME RIGHTS RESERVED

Outline

- Brains
- Neural networks
- Perceptrons
- Multilayer perceptrons
- Applications of neural networks

Brains

- **Axiom 9.1 (Neuroscience Hypothesis).** *Mental activity consists consists primarily of electrochemical activity in networks of brain cells called neurons.*



- **Definition 9.2.** The animal brain is a biological neural network
  - with 10^{11} neurons of > 20 types, 10^{14} synapses, $(1\mathrm{ms}) - (10\mathrm{ms})$ cycle time.
  - Signals are noisy "spike trains" of electrical potential.

Neural Networks as an approach to Artificial Intelligence

- One approach to artificial intelligence is to model and simulate brains.(and hope that AI comes along naturally)
- **Definition 9.3.** The AI subfield of neural networks (also called connectionism, parallel distributed processing, and neural computation) studies computing systems inspired by the biological neural networks that constitute brains.
- Neural networks are attractive computational devices, since they perform important AI tasks – most importantly learning and distributed, noise-tolerant computation – naturally and efficiently.

SOME RIGHTS RESERVED

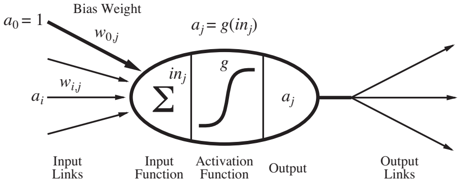
Neural Networks – McCulloch-Pitts “unit”

Definition 9.4. An artificial neural network is a directed graph such that every edge $a_i \to a_j$ is associated with a weight $w_{i,j} \in \mathbb{R}$, and each node a_j with parents $a_1, \ldots, a_n$ is associated with a function $f(w_{1,j}, \ldots, w_{n,j}, x_1, \ldots, x_n) \in \mathbb{R}$.
We call the output of a node's function its activation, the matrix $\mathrm{w}_{i,j}$ the weight matrix, the nodes units and the edges links.

In 1943 McCulloch and Pitts proposed a simple model for a neuron/brain:
Definition 9.5. A McCulloch-Pitts unit first computes a weighted sum of all inputs and then applies an activation function g to it.

$$\mathrm{in}_i = \sum_j \mathrm{w}_{j,i} a_j$$

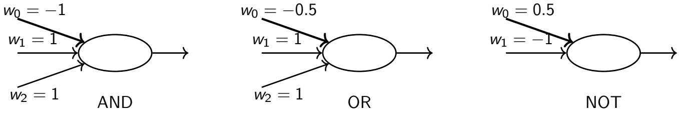
$$a_i \leftarrow g(\mathrm{in}_i) = g(+\sum_j \mathrm{w}_{j,i} a_j)$$



If g is a threshold function, we call the unit a perceptron unit, if g is a logistic function a sigmoid perceptron unit.
A McCulloch-Pitts network is a neural network with McCulloch-Pitts units.

SOME RIGHTS RESERVED

Implementing Logical Functions as Units

- McCulloch-Pitts units are a gross oversimplification of real neurons, but its purpose is to develop understanding of what neural networks of simple units can do.
- **Theorem 9.6 (McCulloch and Pitts).** *Every Boolean function can be implemented as McCulloch-Pitts networks.*
- *Proof:* by construction
  1. Recall that $a_i \longleftarrow g(\sum_j w_{j,i} a_j)$. Let $g(r) = 1$ iff $r > 0$, else 0.
  2. As for linear regression we use $a_0 = 1 \rightsquigarrow w_{0,i}$ as a bias weight (or intercept) (determines the threshold)
  3. 

  4. Any Boolean function can be implemented as a DAG of McCulloch-Pitts units.

□

SOME RIGHTS RESERVED

Network Structures: Feed-Forward Networks

- We have models for neurons ⇝ connect them to neural networks.
- **Definition 9.7.** A neural network is called a feed-forward network, if it is acyclic.
- **Intuition:** Feed-forward networks implement functions, they have no internal state.
- **Definition 9.8.** Feed-forward networks are usually organized in layers: a n layer network has a partition $\{L_0, \ldots, L_n\}$ of the nodes, such that edges only connect nodes from subsequent layer.

  L_0 is called the input layer and its members input units, and L_n the output layer and its members output units. Any unit that is not in the input layer or the output layer is called hidden.

SOME RIGHTS RESERVED

Network Structures: Recurrent Networks

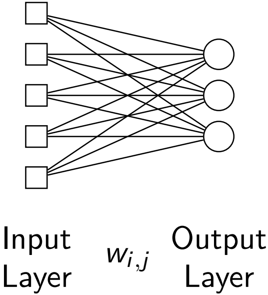
- **Definition 9.9.** A neural network is called recurrent (a RNNs), iff it has cycles.
  - Hopfield networks have symmetric weights ($w_{i,j} = w_{j,i}$) $g(x) = \mathrm{sign}(x)$, $a_i = \pm 1$; (holographic associative memory)
  - Boltzmann machines use stochastic activation functions.
- Recurrent neural networks have cycles with delay $\rightsquigarrow$ have internal state (like flip-flops), can oscillate etc.

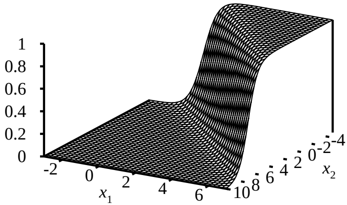
Recurrent neural networks follow largely the same principles as feed-forward networks, so we will not go into details here.

SOME RIGHTS RESERVED

Single-layer Perceptrons

- **Definition 9.10.** A perceptron network is a feed-forward network of perceptron units. A single layer perceptron network is called a perceptron.
- **Example 9.11.**

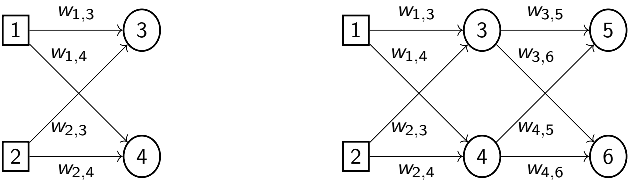




- All input units are directly connected to output units.
- Output units all operate separately, no shared weights $\rightsquigarrow$ treat as the combination of n perceptron units.
- Adjusting weights moves the location, orientation, and steepness of cliff.

Feed-forward Neural Networks (Example)

- Feed-forward network $\widehat{=}$ a parameterized family of nonlinear functions:
- **Example 9.12.** We show two feed-forward networks:



a) single layer (perceptron network) b) 2 layer feed-forward network

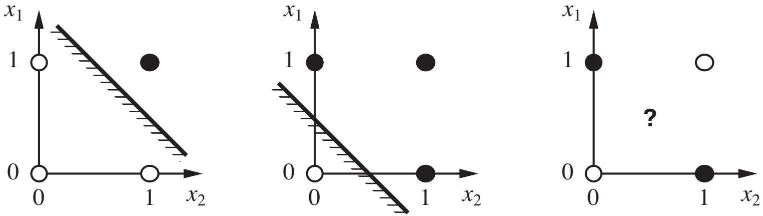
$$\begin{aligned} a_5 &= g(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4) \\ &= g(w_{3,5} \cdot g(w_{1,3} \cdot a_1 + w_{2,3}a_2) + w_{4,5} \cdot g(w_{1,4} \cdot a_1 + w_{2,4}a_2)) \end{aligned}$$

- **Idea:** Adjusting weights changes the function: do learning this way!

Expressiveness of Perceptrons

- Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)
- Can represent AND, OR, NOT, majority, etc., but not XOR(and thus no adders)
- Represents a linear separator in input space:

$$\sum_j w_j x_j > 0 \quad \text{or} \quad W, x \cdot > 0$$



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

- Minsky & Papert (1969) pricked the first neural network balloon!

SOME RIGHTS RESERVED

Perceptron Learning

For learning, we update the weights using gradient descent based on the generalization loss function.

Let e.g. $L(\mathrm{w}) = (y - h_{\mathrm{w}}(x))^2$ (the squared error loss).

We compute the gradient:

$$
\begin{aligned}
\frac{\partial L(\mathrm{w})}{\partial \mathrm{w}_{j,k}} &= 2 \cdot (y_k - h_{\mathrm{w}}(x)_k) \cdot \frac{\partial (y - h_{\mathrm{w}}(x))}{\partial \mathrm{w}_{j,k}} = 2 \cdot (y_k - h_{\mathrm{w}}(x)_k) \cdot \frac{\partial}{\partial \mathrm{w}_{j,k}} (y - g(\sum_{j=0}^{n} \\
&= -2 \cdot (y_k - h_{\mathrm{w}}(x)_k) \cdot g'(\mathrm{in}_k) \cdot x_j
\end{aligned}
$$

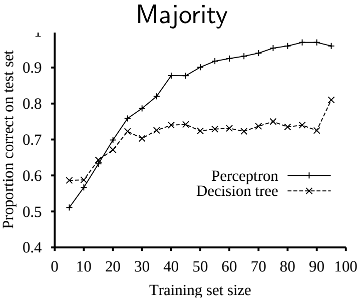
$\rightsquigarrow$ Replacing the constant factor -2 by a learning rate parameter α we get the update rule:

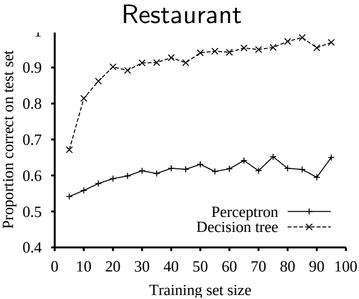
$$\mathrm{w}_{j,k} \leftarrow \mathrm{w}_{j,k} + \alpha \cdot (y_k - h_{\mathrm{w}}(x)_k) \cdot g'(\mathrm{in}_k) \cdot x_j$$

SOME RIGHTS RESERVED

Perceptron learning contd.

The perceptron learning rule converges to a consistent function – *for any linearly separable data set*

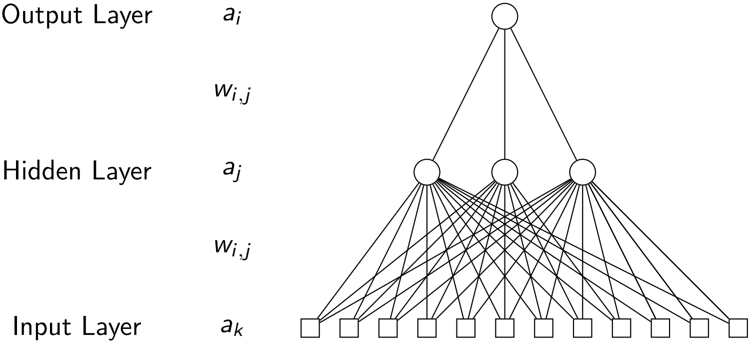




Perceptron learns the majority function easily, where DTL is hopeless.
Conversely, DTL learns the restaurant function easily, where a perceptron is hopeless. (not representable)

Multilayer perceptrons

- **Definition 9.13.** In multi layer perceptrons (MLPs), layers are usually fully connected;
  numbers of hidden units typically chosen by hand.

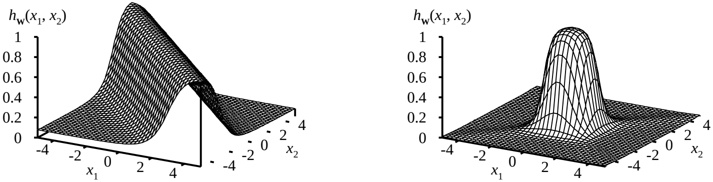


- **Definition 9.14.** Some MLPs have residual connections, i.e. connections that skip layers.

SOME RIGHTS RESERVED

Expressiveness of MLPs

- All continuous functions w/ 2 layers, all functions w/ 3 layers.



- Combine two opposite-facing threshold functions to make a ridge.
- Combine two perpendicular ridges to make a bump.
- Add bumps of various sizes and locations to fit any surface.
- Proof requires exponentially many hidden units. (cf. DTL proof)

SOME RIGHTS RESERVED

Learning in Multilayer Networks

Note: The *output layer* of a multilayer neural network is a single-layer perceptron whose input is the output of the last hidden layer.

$\leadsto$ We can use the perceptron learning rule to update the weights of the output layer; e.g. for a squared error loss function:

$w_{j,k} \leftarrow w_{j,k} + \alpha \cdot (y_k - h_w(x)_k) \cdot g'(\mathrm{in}_k) \cdot a_j$

What about the hidden layers?

Idea: The hidden node j is "responsible" for some fraction of the error proportional to the weight $w_{j,k}$.

$\leadsto$ Back-propagate the error $\Delta_k = (y_k - h_w(x)_k) \cdot g'(\mathrm{in}_j)$ from node k in the output layer to the hidden node j.

Let's justify this:

$$
\begin{aligned}
\frac{\partial L(w)_k}{\partial w_{i,j}} &= -2 \cdot \underbrace{(y_k - h_w(x)_k) \cdot g'(\mathrm{in}_k)}_{=:\Delta_k} \cdot \frac{\partial \mathrm{in}_k}{\partial w_{i,j}} \quad \text{(as before)} \\
&= -2 \cdot \Delta_k \cdot \frac{\partial (\sum_\ell w_{\ell,k} a_\ell)}{\partial w_{i,j}} = -2 \cdot \Delta_k \cdot w_{j,k} \cdot \frac{\partial a_j}{\partial w_{i,j}} = -2 \cdot \Delta_k \cdot w_{j,k} \cdot \frac{\partial g(}{\partial w} \\
&= -2 \cdot \underbrace{\Delta_k \cdot w_{j,k} \cdot g'(\mathrm{in}_j)}_{=:\Delta_{j,k}} \cdot a_i
\end{aligned}
$$

SOME RIGHTS RESERVED

Learning in Multilayer Networks (Hidden Layers)

$$\frac{\partial L(\mathrm{w})_k}{\partial \mathrm{w}_{i,j}} = -2 \cdot \underbrace{\Delta_k \cdot \mathrm{w}_{j,k} \cdot g'(\mathrm{in}_j)}_{=:\Delta_{j,k}} \cdot a_i$$

Idea: The total "error" of the hidden node j is the sum of all the connected nodes k in the next layer

Definition 9.15. The back-propagation rule for hidden nodes of a multilayer perceptron is $\Delta_j \leftarrow g'(\mathrm{in}_j) \cdot (\sum_i \mathrm{w}_{j,i}\Delta_i)$ And the update rule for weights in a hidden layer is $\mathrm{w}_{k,j} \leftarrow \mathrm{w}_{k,j} + \alpha \cdot a_k \cdot \Delta_j$

Remark: Most neuroscientists deny that back-propagation occurs in the brain.

The back-propagation process can be summarized as follows:

1. Compute the Δ values for the output units, using the observed error.
2. Starting with output layer, repeat the following for each layer in the network, until the earliest hidden layer is reached:
   2.1 Propagate the Δ values back to the previous (hidden) layer.
   2.2 Update the weights between the two layers.

Backprogagation Learning Algorithm

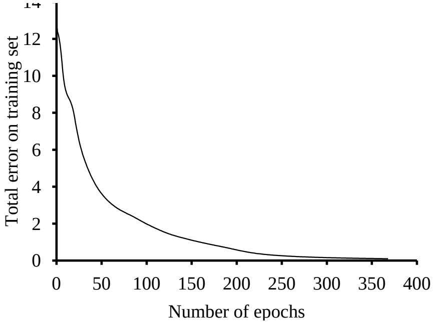
- **Definition 9.16.** The back-propagation learning algorithm is given the following pseudocode

```
function BACK−PROP−LEARNING(examples,network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with $L$ layers, weights $w_{i,j}$, activation function $g$
  local variables: $\Delta$, a vector of errors, indexed by network node
  foreach weight $w_{i,j}$ in network do
    $w_{i,j}$ := a small random number
  repeat
    foreach example (x, y) in examples do
      /* Propagate the inputs forward to compute the outputs */
      foreach node $i$ in the input layer do $a_i := x_i$
        for $l = 2$ to $L$ do
          foreach node $j$ in layer $l$ do
            $\mathrm{in}_j := \sum_i w_{i,j} a_i$
            $a_j := g(\mathrm{in}_j)$
      /* Propagate deltas backward from output layer to input layer */
      foreach node $j$ in the output layer do $\Delta[j] := g'(\mathrm{in}_j) \cdot (y_j - a_j)$
      for $l = L - 1$ to 1 do
        foreach node $i$ in layer $l$ do $\Delta[i] := g'(\mathrm{in}_i) \cdot (\sum_j w_{i,j} \Delta[j])$
      /* Update every weight in network using deltas */
      foreach weight $w_{i,j}$ in network do $w_{i,j} := w_{i,j} + \alpha \cdot a_i \cdot \Delta[j]$
  until some stopping criterion is satisfied
  return network
```

SOME RIGHTS RESERVED

Back-Propagation – Properties

- Sum gradient updates for all examples in some “batch” and apply gradient descent.
- Learning curve for 100 restaurant examples: finds exact fit.

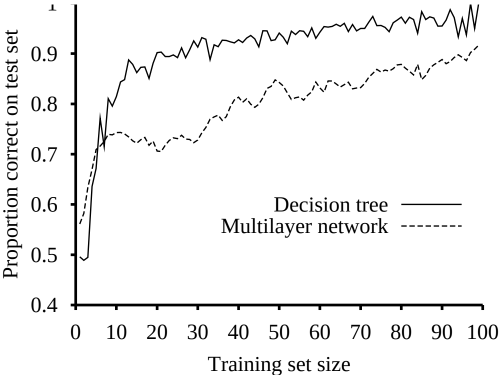


- **Typical problems:** slow convergence, local minima.

SOME RIGHTS RESERVED

Back-Propagation – Properties (contd.)

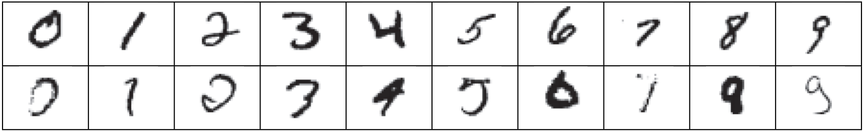
- **Example 9.17.** Learning curve for MLPs with 4 hidden units:



- **Experience shows:** MLPs are quite good for complex pattern recognition tasks,
  but resulting hypotheses cannot be understood easily.
- This makes MLPs ineligible for some tasks, such as credit card and loan approvals, where law requires clear unbiased criteria.

SOME RIGHTS RESERVED

Handwritten digit recognition



- 400–300–10 unit MLP = 1.6% error
- LeNet: 768–192–30–10 unit MLP = 0.9% error
- Current best (kernel machines, vision algorithms) ≈ 0.6% error

SOME RIGHTS RESERVED

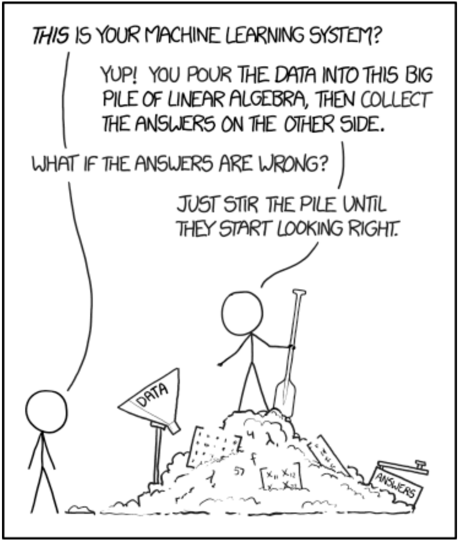
Summary

- neural networks can be extremely powerful (hypothesis space intractably complex)
- Perceptrons (one-layer networks) insufficiently expressive for most applications
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation
- Many applications: speech, driving, handwriting, fraud detection, etc.
- Engineering, cognitive modelling, and neural system modelling subfields have largely diverged
- Drawbacks: take long to converge, require large amounts of data, and are difficult to *interpret* (Why is the output what it is?)

SOME RIGHTS RESERVED

XKCD on Machine Learning

- **A Skepticists View:** see `https://xkcd.com/1838/`


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

SOME RIGHTS RESERVED

Summary of Inductive Learning

- Learning needed for unknown environments, lazy designers.
- Learning agent = performance element + learning element.
- Learning method depends on type of performance element, available feedback, type of component to be improved, and its representation.
- For supervised learning, the aim is to find a simple hypothesis that is approximately consistent with training examples
- Decision tree learning using information gain.
- Learning performance = prediction accuracy measured on test set
- PAC learning as a general theory of learning boundaries.
- Linear regression (hypothesis space of univariate linear functions).
- Linear classification by linear regression with hard and soft thresholds.

SOME RIGHTS RESERVED

Chapter 27
Statistical Learning

SOME RIGHTS RESERVED

Statistical Learning: Outline

- **Definition 0.1.** Statistical learning has the goal to learn the correct probability distribution of a random variable.
- **Example 0.2.**
  - Bayesian learning, i.e. learning probabilistic models (e.g. the CPTs in Bayesian networks) from observations.
  - Maximum *a posteriori* and maximum likelihood learning
  - Bayesian network learning
    - ML Parameter Learning with Complete Data
    - Naive Bayes Models/Learning

SOME RIGHTS RESERVED

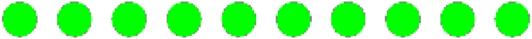
27.1 Full Bayesian Learning

SOME RIGHTS RESERVED

The Candy Flavors Example

- **Example 1.1.** Suppose there are five kinds of bags of candies:
  1. 10% are h_1: 100% cherry candies
  2. 20% are h_2: 75% cherry candies + 25% lime candies
  3. 40% are h_3: 50% cherry candies + 50% lime candies
  4. 20% are h_4: 25% cherry candies + 75% lime candies
  5. 10% are h_5: 100% lime candies

  Then we observe candies drawn from some bag:



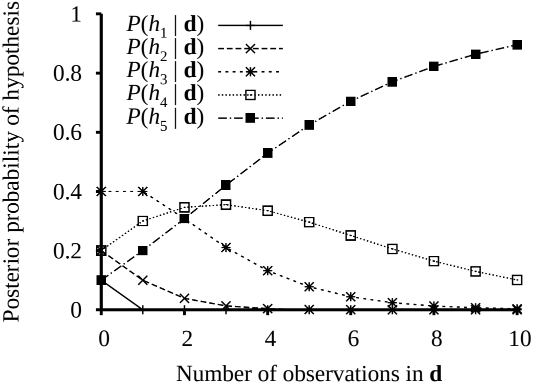
  What kind of bag is it? What flavour will the next candy be?

Note: Every hypothesis is itself a probability distribution over the random variable "flavour".

SOME RIGHTS RESERVED

Candy Flavors: Posterior probability of hypotheses

- **Example 1.2.** Let d_i be the event that the ith drawn candy is green. The probability of hypothesis h_i after n limes are observed ($\widehat{=}\ \mathrm{d}_{1:n} =: \mathrm{d}$) is

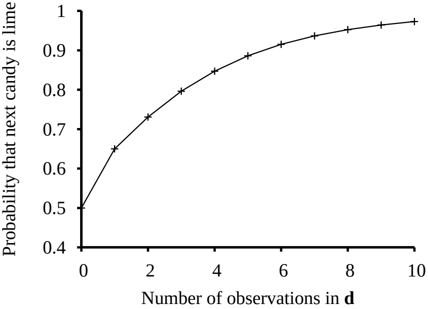


if the observations are IID, i.e. $P(\mathrm{d} \mid h_i) = \prod_j P(d_j \mid h_i)$ and the hypothesis prior is as advertised. (e.g. $P(\mathrm{d} \mid h_3) = 0.5^{10} = 0.1\%$)
The posterior probabilities start with the hypothesis priors, change with data.

Candy Flavors: Prediction Probability

- We calculate that the $n+1$-th candy is lime:

$$P(d_{n+1} = \mathrm{lime} \mid \mathrm{d}) = \sum_i P(d_{n+1} = \mathrm{lime} \mid h_i) \cdot P(h_i \mid \mathrm{d})$$



$\leadsto$ we compute the expected value of *the probability of the next candy being lime* over all hypotheses (i.e. distributions).

$\leadsto$ "meta-distribution"

SOME RIGHTS RESERVED

Full Bayesian Learning

- **Idea:** View learning as Bayesian updating of a probability distribution over the hypothesis space:
  - H is the hypothesis variable with values $h_1, h_2, \ldots$ and prior $\mathbb{P}(H)$.
  - jth observation d_j gives the outcome of random variable D_j.
  - $\mathrm{d} := d_1, \ldots, d_N$ constitutes the training set of a inductive learning problem.
- **Definition 1.3.** Bayesian learning calculates the probability of each hypothesis and makes predictions based on this:
  - Given the data so far, each hypothesis has a posterior probability:

    $$P(h_i \mid \mathrm{d}) = \alpha(P(\mathrm{d} \mid h_i) \cdot P(h_i))$$

    where $P(\mathrm{d} \mid h_i)$ is called the likelihood (of the data under each hypothesis) and $P(h_i)$ the hypothesis prior.
  - Bayesian predictions use a likelihood-weighted average over the hypotheses:

    $$\mathbb{P}(\mathrm{X}|\mathrm{d}) = \sum_i \mathbb{P}(\mathrm{X}|\mathrm{d}, h_i) \cdot P(h_i \mid \mathrm{d}) = \sum_i \mathbb{P}(\mathrm{X}|h_i) \cdot P(h_i \mid \mathrm{d})$$

- **Observation:** No need to pick one best-guess hypothesis for Bayesian predictions! (and that is all an agent cares about)

SOME RIGHTS RESERVED

Full Bayesian Learning: Properties

- **Observation:** The Bayesian prediction eventually agrees with the true hypothesis.
  - The probability of generating "uncharacteristic" data indefinitely is vanishingly small.
  - *Proof sketch:* Argument analogous to PAC learning.
- **Problem:** Summing over the hypothesis space is often intractable.
- **Example 1.4.** There are $2^{2^6} = 18,446,744,073,709,551,616$ Boolean functions of 6 arguments.
- **Solution:** Approximate the learning methods to simplify.

SOME RIGHTS RESERVED

27.2 Approximations of Bayesian Learning

SOME RIGHTS RESERVED

Maximum A Posteriori (MAP) Approximation

- **Goal:** Get rid of summation over the space of all hypotheses in predictions.
- **Idea:** Make predictions wrt. the "most probable hypothesis"!
- **Definition 2.1.** For maximum a posteriori learning (MAP learning) choose the MAP hypothesis h_{MAP} that maximizes $P(h_i \mid \mathrm{d})$.
  I.e., maximize $P(\mathrm{d} \mid h_i) \cdot P(h_i)$ or (even better) $\log_2(P(\mathrm{d} \mid h_i)) + \log_2(P(h_i))$.
- Predictions made according to a MAP hypothesis h_{MAP} are approximately Bayesian to the extent that $\mathrm{P}(X|\mathrm{d}) \approx \mathrm{P}(X|h_{\mathrm{MAP}})$.
- **Example 2.2.** In our candy example, $h_{\mathrm{MAP}} = h_5$ after three limes in a row
  - a MAP learner then predicts that candy 4 is lime with probability 1.
  - compare with Bayesian prediction of 0.8. (see prediction curves above)
- As more data arrive, the MAP and Bayesian predictions become closer, because the competitors to the MAP hypothesis become less and less probable.
- For deterministic hypotheses, $P(\mathrm{d} \mid h_i)$ is 1 if consistent, 0 otherwise
  $\leadsto$ MAP = simplest consistent hypothesis. (cf. science)
- **Remark:** Finding MAP hypotheses is often much easier than Bayesian learning, because it requires solving an optimization problem instead of a large summation (or integration) problem.

SOME RIGHTS RESERVED

Digression From MAP-learning to MDL-learning

- **Idea:** Reinterpret the log terms $\log_2(P(\mathrm{d} \mid h_i)) + \log_2(P(h_i))$ in MAP learning:
  - Maximizing $P(\mathrm{d} \mid h_i) \cdot P(h_i)$ $\widehat{=}$ minimizing $-\log_2(P(\mathrm{d} \mid h_i)) - \log_2(P(h_i))$.
  - $-\log_2(P(\mathrm{d} \mid h_i))$ $\widehat{=}$ number of bits to encode data given hypothesis.
  - $-\log_2(P(h_i))$ $\widehat{=}$ additional bits to encode hypothesis. (???)
- **Indeed** if hypothesis predicts the data exactly – e.g. h_5 in candy example – then $\log_2(1) = 0$ $\rightsquigarrow$ preferred hypothesis.
- This is more directly modeled by the following approximation to Bayesian learning:
- **Definition 2.3.** In minimum description length learning (MDL learning) the MDL hypothesis h_{MDL} minimizes the information entropy of the hypothesis likelihood.

SOME RIGHTS RESERVED

Maximum Likelihood (ML) approximation

- **Observation:** For large data sets, the prior becomes irrelevant. (we might not trust it anyways)
- **Idea:** Use this to simplify learning.
- **Definition 2.4.** Maximum likelihood learning (ML learning): choose the ML hypothesis h_{ML} maximizing $P(\mathrm{d} \mid h_i)$. (simply get the best fit to the data)
- **Remark:** ML learning $\widehat{=}$ MAP learning for a uniform prior. (reasonable if all hypotheses are of the same complexity)
- ML learning is the "standard" (non Bayesian) statistical learning method.

SOME RIGHTS RESERVED

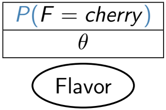
27.3 Parameter Learning for Bayesian Networks

SOME RIGHTS RESERVED

ML Parameter Learning in Bayesian Nets

Bayesian networks (with continuous random variables) often feature nodes with a particular *parametric* distribution $D(\theta)$ (e.g. normal, binomial, Poisson, etc.). How do we learn the parameters of these distributions from data?

Example 3.1. We get a candy bag from a new manufacturer; what is the fraction θ of cherry candies? (Note: We use the probability itself as the parameter. This is somewhat boring, but simple.)



$P(F = cherry)$
θ

New Facet: Any θ is possible: continuum of hypotheses h_θ
θ is a parameter for this simple (binomial) family of models; We call h_θ a MLP hypothesis and the process of learning θ MLP learning.

Example 3.2. Suppose we unwrap N candies, c cherries and $\ell = N - c$ limes. These are IID observations, so the likelihood is

$$P(\mathrm{d} \mid h_\theta) = \prod_{j=1}^{N} P(\mathrm{d}_j \mid h_\theta) = \theta^c \cdot (1-\theta)^\ell$$

ML Parameter Learning in Bayes Nets

Trick: When optimizing a product, optimize the logarithm instead! ($\log_2(!)$ is monotone and turns products into sums)

Definition 3.3. The log likelihood is the binary logarithm of the likelihood.
$L(\mathrm{d}|h) := \log_2(P(\mathrm{d} \mid h))$

Example 3.4. Compute the log likelihood as (using **??**)

$$L(\mathrm{d}|h_\theta) = \log_2(P(\mathrm{d} \mid h_\theta)) = \sum_{j=1}^{N} \log_2(P(\mathrm{d}_j \mid h_\theta)) = c\log_2(\theta) + \ell\log_2(1-\theta)$$

Maximize this w.r.t. θ

$$\frac{\partial}{\partial\theta}(L(\mathrm{d}|h_\theta)) = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \rightsquigarrow \theta = \frac{c}{c+\ell} = \frac{c}{N}$$

In English: h_θ asserts that the actual proportion of cherries in the bag is equal to the observed proportion in the candies unwrapped so far! (...exactly what we should expect!) (⇒ Generalize to more interesting parametric models later)

Warning: This causes problems with 0 counts!

SOME RIGHTS RESERVED

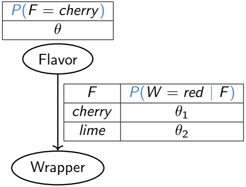
ML Learning for Multiple Parameters in Bayesian Networks

- **Cooking Recipe:**
  1. Write down an expression for the likelihood of the data as a function of the parameter(s).
  2. Write down the derivative of the log likelihood with respect to each parameter.
  3. Find the parameter values such that the derivatives are zero

SOME RIGHTS RESERVED

Multiple Parameters Example

- **Example 3.5.** Red/green wrapper depends probabilistically on flavour:



$P(F = cherry)$
θ

Flavor

F	$P(W = red \mid F)$
cherry	θ_1
lime	θ_2

Wrapper

- Likelihood for, e.g., cherry candy in green wrapper:

$$\begin{aligned} & P(F = cherry, W = green \mid h_{\theta,\theta_1,\theta_2}) \\ = \quad & P(F = cherry \mid h_{\theta,\theta_1,\theta_2}) \cdot P(W = green \mid F = cherry, h_{\theta,\theta_1,\theta_2}) \\ = \quad & \theta \cdot (1 - \theta_1) \end{aligned}$$

- **Ovservation:** For N candies, r_c red-wrapped cherry candies, etc. we have

$$P(\mathrm{d} \mid h_{\theta,\theta_1,\theta_2}) \quad = \quad \theta^c \cdot (1-\theta)^\ell \cdot {\theta_1}^{r_c} \cdot (1-\theta_1)^{g_c} \cdot {\theta_2}^{r_\ell} \cdot (1-\theta_2)^{g_\ell}$$

Multiple Parameters Example (contd.)

- Minimize the log likelihood:

$$\begin{aligned} L &= c\log_2(\theta) + \ell\log_2(1-\theta) \\ &+ r_c\log_2(\theta_1) + g_c\log_2(1-\theta_1) \\ &+ r_\ell\log_2(\theta_2) + g_\ell\log_2(1-\theta_2) \end{aligned}$$

- Derivatives of L contain only the relevant parameter:

$$\begin{aligned} \frac{\partial L}{\partial \theta} &= \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 & \leadsto & \quad \theta = \frac{c}{c+\ell} \\ \frac{\partial L}{\partial \theta_1} &= \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 & \leadsto & \quad \theta_1 = \frac{r_c}{r_c+g_c} \\ \frac{\partial L}{\partial \theta_2} &= \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 & \leadsto & \quad \theta_2 = \frac{r_\ell}{r_\ell+g_\ell} \end{aligned}$$

- **Upshot:** With complete data, *parameters can be learned separately* in Bayesian networks.
- **Remaining Problem:** Have to be careful with zero values! (division by zero)

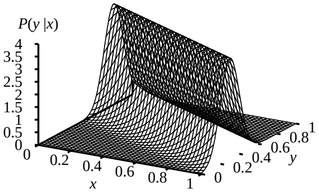
SOME RIGHTS RESERVED

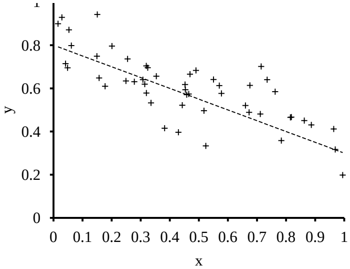
Example: Linear Gaussian Model

A continuous random variable Y has the *linear-Gaussian distribution* with respect to a continuous random variable X, if the outcome of Y is determined by a linear function of the outcome of X *plus gaussian noise with a fixed variance* σ, i.e.

$$P(y_1 \leq Y \leq y_2 \,|\, X = x) = \int_{y_1}^{y_2} N(y; \theta_1 x + \theta_2, \sigma^2)\,\mathrm{d}y = \int_{y_1}^{y_2} \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\cdot\left(\frac{y-(\theta_1 x+\theta_2)}{\sigma}\right)^2}$$

⤳ assuming σ given, we have two parameter θ_1 and θ_2 ⤳ Hypothesis space is $\mathbb{R} \times \mathbb{R}$





Example: Linear Gaussian Model

$$P(y_1 \leq Y \leq y_2 \mid X = x) = \int_{y_1}^{y_2} \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\cdot\left(\frac{y-(\theta_1 x+\theta_2)}{\sigma}\right)^2} \, \mathrm{d}y$$

$\rightsquigarrow$ Given observations $X = X$, $Y = y$, maximize $\prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i-(\theta_1 x_i+\theta_2))^2}{2\sigma^2}}$ w.r.t. θ_1, θ_2. (we can ignore the integral for this)

Using the log likelihood, this is equivalent to minimizing $\sum_{i=1}^{N}(y_i - (\theta_1 x_i + \theta_2))^2$
$\rightsquigarrow$ minimizing the sum of squared errors gives the ML solution

SOME RIGHTS RESERVED

Statistical Learning: Summary

- Full Bayesian learning gives best possible predictions but is intractable.
- MAP learning balances complexity with accuracy on training data.
- Maximum likelihood learning assumes uniform prior, OK for large data sets:
  1. Choose a parameterized family of models to describe the data.
     $\rightsquigarrow$ requires substantial insight and sometimes new models.
  2. Write down the likelihood of the data as a function of the parameters.
     $\rightsquigarrow$ may require summing over hidden variables, i.e., inference.
  3. Write down the derivative of the log likelihood w.r.t. each parameter.
  4. Find the parameter values such that the derivatives are zero.
     $\rightsquigarrow$ may be hard/impossible; modern optimization techniques help.
- Naive Bayes models as a fall-back solution for machine learning:
  - conditional independence of all attributes as simplifying assumption.

SOME RIGHTS RESERVED

Chapter 28
Reinforcement Learning

SOME RIGHTS RESERVED

28.1 Reinforcement Learning: Introduction & Motivation

SOME RIGHTS RESERVED

Unsupervised Learning

- **So far:** We have studied "learning from examples". (functions, logical theories, probability models)
- **Now:** How can agents learn "what to do" in the absence of labeled examples of "what to do". We call this problem unsupervised learning.
- **Example 1.1 (Playing Chess).** Learn transition models for own moves and maybe predict opponent's moves.
- **Problem:** The agent needs to have some feedback about what is good/bad $\leadsto$ cannot decide "what to do" otherwise. (recall: external performance standard for learning agents)
- **Example 1.2.** The ultimate feedback in chess is whether you win, lose, or draw.
- **Definition 1.3.** We call a learning situation where there are no labeled examples unsupervised learning and the feedback involved a reward or reinforcement.
- **Example 1.4.** In soccer, there are intermediate reinforcements in the shape of goals, penalties, . . .

SOME RIGHTS RESERVED

Reinforcement Learning as Policy Learning

- **Definition 1.5.** Reinforcement learning is a type of unsupervised learning where an agent learns how to behave in an environment by performing actions and seeing the results.
- **Recap:** In ??? we introduced rewards as parts of MDPs (Markov decision processes) to define optimal policies.
  - an optimal policy maximizes the expected total reward.
- **Idea:** The task of reinforcement learning is to use observed rewards to come up with an optimal policy.
- In MDPs, the agent has total knowledge about the environment *and the reward function*, in reinforcement learning we do not assume this. (⇝ POMDPs+reward-learning)
- **Example 1.6.** You play a game without knowing the rules, and at some time the opponent shouts "*you lose!*"

SOME RIGHTS RESERVED

Scope and Forms of Reinforcement Learning

- **Reinforcement Learning solves all of AI:** An agent is placed in an environment and must learn to behave successfully therein.
- **KISS:** We will only look at simple environments and simple agent designs:
  - A utility-based agent learns a utility function on states and uses it to select actions that maximize the expected outcome utility. (passive learning)
  - A Q-learning agent learns an action-utility function, or Q-function, giving the expected utility of taking a given action in a given state. (active learning)
  - A reflex agent learns a policy that maps directly from states to actions.

SOME RIGHTS RESERVED

28.2 Passive Learning

SOME RIGHTS RESERVED

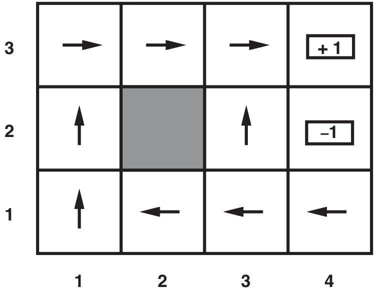
Passive Learning

- **Definition 2.1 (To keep things simple).** Agent uses a state-based representation in a fully observable environment:
  - In passive learning, the agent's policy π is fixed: in state s, it always executes the action $\pi(s)$.
  - Its goal is simply to learn how good the policy is – that is, to learn the utility function $U^{\pi}(s)$.
- The passive learning task is similar to the policy evaluation task (part of the policy iteration algorithm) but the agent does not know
  - the transition model $P(s' \mid s,a)$, which specifies the probability of reaching state s' from state s after doing action a,
  - the reward function $R(s)$, which specifies the reward for each state.

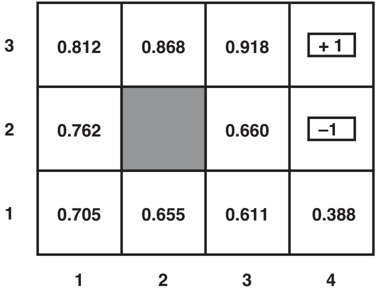
SOME RIGHTS RESERVED

Passive Learning by Example

- **Example 2.2 (Passive Learning).** We use the 4 × 3 world introduced above



Optimal Policy π



	1	2	3	4
3	0.812	0.868	0.918	+ 1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388

Utilities, given π

- The agent executes a set of trials in the environment using its policy π.
- In each trial, the agent starts in state (1,1) and experiences a sequence of state transitions until it reaches one of the terminal states, (4,2) or (4,3).
- Its percepts supply both the current state and the reward received in that state.

SOME RIGHTS RESERVED

Passive Learning by Example

- **Example 2.3.** Typical trials might look like this:
  1. $(1,1)_{-0.4} \rightsquigarrow (1,2)_{-0.4} \rightsquigarrow (1,3)_{-0.4} \rightsquigarrow (1,2)_{-0.4} \rightsquigarrow (1,3)_{-0.4} \rightsquigarrow (2,3)_{-0.4} \rightsquigarrow (3,3)_{-0.4} \rightsquigarrow (4,3)_{+1}$
  2. $(1,1)_{-0.4} \rightsquigarrow (1,2)_{-0.4} \rightsquigarrow (1,3)_{-0.4} \rightsquigarrow (2,3)_{-0.4} \rightsquigarrow (3,3)_{-0.4} \rightsquigarrow (3,2)_{-0.4} \rightsquigarrow (3,3)_{-0.4} \rightsquigarrow (4,3)_{+1}$
  3. $(1,1)_{-0.4} \rightsquigarrow (2,1)_{-0.4} \rightsquigarrow (3,1)_{-0.4} \rightsquigarrow (3,2)_{-0.4} \rightsquigarrow (4,2)_{-1}$.
- **Definition 2.4.** The utility is defined to be the expected sum of (discounted) rewards obtained if policy π is followed.

$$U^\pi(s) := E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

where $R(s)$ is the reward for a state, S_t (a random variable) is the state reached at time t when executing policy π, and $S_0 = s$. (for 4×3 we take the discount factor $\gamma = 1$)

SOME RIGHTS RESERVED

Direct Utility Estimation

- A simple method for direct utility estimation was invented in the late 1950s in the area of adaptive control theory.
- **Definition 2.5.** The utility of a state is the expected total reward from that state onward (called the expected reward to go).
- **Idea:** Each trial provides a sample of the reward to go for each state visited.
- **Example 2.6.** The first trial in ??? provides a sample total reward of 0.72 for state (1,1), two samples of 0.76 and 0.84 for (1,2), two samples of 0.80 and 0.88 for (1,3), . . .
- **Definition 2.7.** The direct utility estimation algorithm cycles over trials, calculates the reward to go for each state, and updates the estimated utility for that state by keeping the running average for that for each state in a table.
- **Observation 2.8.** *In the limit, the sample average will converge to the true expectation (utility) from ???.*
- *Remark 2.9.* Direct utility estimation is just supervised learning, where each example has the state as input and the observed reward to go as output.
- **Upshot:** We have reduced reinforcement learning to an inductive learning problem.

SOME RIGHTS RESERVED

Adaptive Dynamic Programming

- **Problem:** The utilities of states are not independent in direct utility estimation!
- The utility of each state equals its own reward plus the expected utility of its successor states.
- **So:** The utility values obey a Bellman equation for a fixed policy π.

$$U^{\pi}(s) = R(s) + \gamma \cdot (\sum_{s'} P(s' \mid s,\pi(s)) \cdot U^{\pi}(s'))$$

- **Observation 2.10.** *By ignoring the connections between states, direct utility estimation misses opportunities for learning.*
- **Example 2.11.** Recall trial 2 in ???; state (3,3) is new.

  2 $(1,1)_{-0.4} \rightsquigarrow (1,2)_{-0.4} \rightsquigarrow (1,3)_{-0.4} \rightsquigarrow (2,3)_{-0.4} \rightsquigarrow (3,3)_{-0.4} \rightsquigarrow (3,2)_{-0.4} \rightsquigarrow (3,3)_{-0.4} \rightsquigarrow (4,3)_{+1}$
  - The next transition reaches (3,3), (known high utility from trial 1)
  - Bellman equation: $\rightsquigarrow$ high $U^{\pi}(3,2)$ because $(3,2)_{-0.4} \rightsquigarrow (3,3)$
  - But direct utility estimation learns nothing until the end of the trial.
- **Intuition:** Direct utility estimation searches for U in a hypothesis space that too large $\leftsquigarrow$ many functions that violate the Bellman equations.
- Thus the algorithm often converges very slowly.

SOME RIGHTS RESERVED

Adaptive Dynamic Programming

- **Idea:** Take advantage of the constraints among the utilities of states by
  - learning the transition model that connects them,
  - solving the corresponding Markov decision process using a dynamic programming method.

  This means plugging the learned transition model $\mathrm{P}(s'|s,\pi(s))$ and the observed rewards $R(s)$ into the Bellman equations to calculate the utilities of the states.
- **As above:** These equations are linear (no maximization involved) (solve with any any linear algebra package).
- **Observation 2.12.** *Learning the model itself is easy, because the environment is fully observable.*
- **Corollary 2.13.** *We have a supervised learning task where the input is a state–action pair and the output is the resulting state.*
  - *In the simplest case, we can represent the transition model as a table of probabilities.*
  - *Count how often each action outcome occurs and estimate the transition probability $P(s' \mid s,a)$ from the frequency with which s' is reached by action a in s.*
- **Example 2.14.** In the 3 trials from ???, *Right* is executed 3 times in $(1,3)$ and 2 times the result is $(2,3)$, so $P((2,3) \mid (1,3),Right)$ is estimated to be $2/3$.

SOME RIGHTS RESERVED

Passive ADP Learning Algorithm

- **Definition 2.15.** The passive ADP algorithm is given by

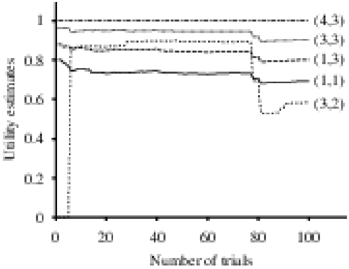
```
function PASSIVE−ADP−AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
  persistent: $\pi$ a fixed policy
      mdp, an MDP with model $P$, rewards $R$, discount $\gamma$
      $U$, a table of utilities, initially empty
      $N_{sa}$, a table of frequencies for state−action pairs, initially zero
      $N_{s'|sa}$, a table of outcome frequencies given state−action pairs, initially zero
      $s$, $a$, the previous state and action, initially null
    if $s'$ is new then $U[s'] := r'$; $R[s'] := r'$
    if $s$ is not null then
      increment $N_{sa}[s,a]$ and $N_{s'|sa}[s',s,a]$
      for each $t$ such that $N_{s]|sa}[t,s,a]$ is nonzero do
         $P(t|s,a) := N_{s'|sa}[t,s,a]/N_{sa}[s,a]$
    $U$ := POLICY−EVALUATION($\pi$,mdp)
    if $s'$.TERMINAL? then $s,a$ := null else $s,a := s',\pi[s']$
    return $a$
```

POLICY−EVALUATION computes $U^\pi(s):=E\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)\right]$ in a MDP.

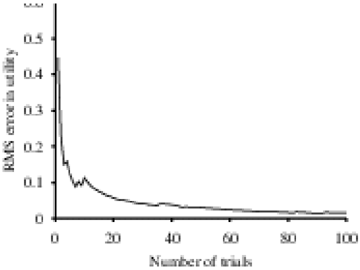
SOME RIGHTS RESERVED

Passive ADP Convergence

- **Example 2.16 (Passive ADP learning curves for the 4x3 world).** Given the optimal policy from ???



utility estimates/trials



error for $U(1,1)$: 20 runs of 100 trials

Note the large changes occurring around the 78th trial – this is the first time that the agent falls into the -1 terminal state at (4,2).

- **Observation 2.17.** *The ADP agent is limited only by its ability to learn the transition model.* *(intractable for large state spaces)*
- **Example 2.18.** In backgammon, roughly 10^{50} equations in 10^{50} unknowns.
- **Idea:** Use this as a baseline to compare passive learning algorithms

SOME RIGHTS RESERVED

28.3 Active Reinforcement Learning

SOME RIGHTS RESERVED

Active Reinforcement Learning

- **Recap:** A passive learning agent has a fixed policy that determines its behavior.
- An active agent must also decide what actions to take.
- **Idea:** Adapt the passive ADP algorithm to handle this new freedom.
  - learn a complete model with outcome probabilities for all actions, rather than just the model for the fixed policy. (use PASSIVE-ADP-AGENT)
  - choose actions; the utilities to learn are defined by the optimal policy, they obey the Bellman equation:

$$U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} (\sum_{s'} U(s') \cdot P(s' \mid s,a))$$

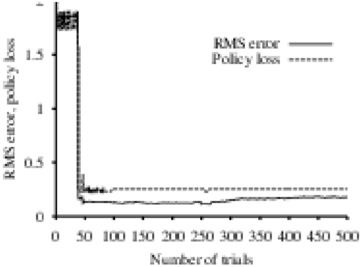
  - solve with value/policy iteration techniques from ???.
  - choose a good action, e.g.
    - by one-step lookahead to maximize expected utility, or
    - if agent uses policy iteration and has optimal policy, execute that.

  This agent/algorithm is greedy, since it only optimizes the next step!

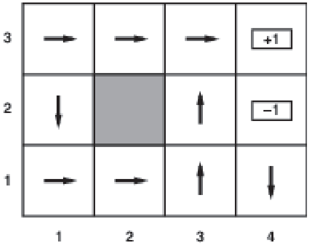
SOME RIGHTS RESERVED

Greedy ADP Learning (Evaluation)

- **Example 3.1 (Greedy ADP learning curves for the 4x3 world).**



average error/loss



suboptimal policy involved

The agent follows the optimal policy for the learned model at each step.

- It does not learn the true utilities or the true optimal policy!
- instead, in the 39th trial, it finds a policy that reaches the +1 reward along the lower route via (2,1), (3,1), (3,2), and (3,3).
- After experimenting with minor variations, from the 276th trial onward it sticks to that policy, never learning the utilities of the other states and never finding the optimal route via (1,2), (1,3), and (2,3).

SOME RIGHTS RESERVED

Exploration in Active Reinforcement Learning

- **Observation 3.2.** *Greedy active ADP learning agents very seldom converge against the optimal solution*
  - *The learned model is not the same as the true environment,*
  - *What is optimal in the learned model need not be in the true environment.*
- What can be done? The agent does not know the true environment.
- **Idea:** Actions do more than provide rewards according to the learned model
  - they also contribute to learning the true model by affecting the percepts received.
  - By improving the model, the agent may reap greater rewards in the future.
- **Observation 3.3.** *An agent must make a tradeoff between*
  - *exploitation to maximize its reward as reflected in its current utility estimates and*
  - *exploration to maximize its long term well-being.*

  Pure exploitation risks getting stuck in a rut. Pure exploration to improve one's knowledge is of no use if one never puts that knowledge into practice.
- Compare with the information gathering agent from ???.

SOME RIGHTS RESERVED

Chapter 29
Knowledge in Learning

29.1 Logical Formulations of Learning

SOME RIGHTS RESERVED

Knowledge in Learning: Motivation

- **Recap:** Learning from examples. (last chapter)
- **Idea:** Construct a function with the input/output behavior observed in data.
- **Method:** Search for suitable functions in the hypothesis space. (e.g. decision trees)
- **Observation 1.1.** *Every learning task begins from zero.* (*except for the choice of hypothesis space*)
- **Problem:** We have to forget everything before we can learn something new.
- **Idea:** Utilize prior knowledge about the world! (represented e.g. in logic)

A logical Formulation of Learning

- **Recall:** Examples are composed of descriptions (of the input sample) and classifications.
- **Idea:** Represent examples and hypotheses as logical formulae.
- **Example 1.2.** For attribute-based representations, we can use PL^1: we use predicate constants for Boolean attributes and classification and function constants for the other attributes.
- **Definition 1.3.** Logic based inductive learning tries to learn an hypothesis h that explains the classifications of the examples given their description, i.e. $h, \mathcal{D} \vDash \mathcal{C}$ (the explanation constraint), where
  - $\mathcal{D}$ is the conjunction of the descriptions, and
  - $\mathcal{C}$ the conjunction of their classifications.
- **Idea:** We solve the explanation constraint $h, \mathcal{D} \vDash \mathcal{C}$ for h where h ranges over some hypothesis space.
- **Refinement:** Use Occam's razor or additional constraints to avoid $h = \mathcal{C}$. (too easy otherwise/boring; see below)

SOME RIGHTS RESERVED

A logical Formulation of Learning (Restaurant Examples)

- **Example 1.4 (Restaurant Example again).** Descriptions are conjunctions of literals built up from
  - predicates Alt, Bar, $\mathrm{Fri/Sat}$, Hun, Rain, and res
  - equations about the functions Pat, Price, Type, and Est.

  For instance the first example X_1 from ???, can be described as

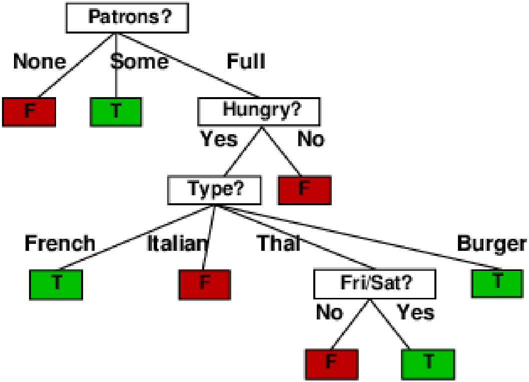
  $$\mathrm{Alt}(X_1) \wedge \neg\mathrm{Bar}(X_1) \wedge \mathrm{Fri/Sat}(X_1) \wedge \mathrm{Hun}(X_1) \wedge \ldots$$

  The classification is given by the goal predicate $\mathrm{WillWait}$, in this case $\mathrm{WillWait}(X_1)$ or $\neg\mathrm{WillWait}(X_1)$.

SOME RIGHTS RESERVED

A logical Formulation of Learning (Restaurant Tree)

- **Example 1.5 (Restaurant Example again; Tree).** The induced decision tree from ???



A logical Formulation of Learning (Restaurant Tree)

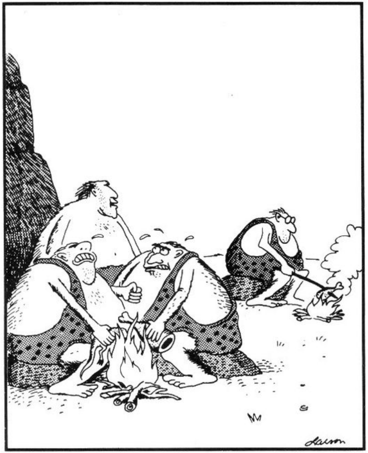
- **Example 1.6 (Restaurant Example again; Tree).** The induced decision tree from ??? can be represented as

$$\begin{aligned}\forall r.\mathrm{WillWait}(r) \quad \Leftrightarrow \quad & \mathrm{Pat}(r, \mathrm{Some})\\ \lor \quad & \mathrm{Pat}(r, \mathrm{Full}) \land \mathrm{Hun}(r) \land \mathrm{Type}(r, \mathrm{French})\\ \lor \quad & \mathrm{Pat}(r, \mathrm{Full}) \land \mathrm{Hun}(r) \land \mathrm{Type}(r, \mathrm{Thai}) \land \mathrm{Fri/Sat}(r)\\ \lor \quad & \mathrm{Pat}(r, \mathrm{Full}) \land \mathrm{Hun}(r) \land \mathrm{Type}(r, \mathrm{Burger})\end{aligned}$$

Method: Construct a disjunction of all the paths from the root to the positive leaves interpreted as conjunctions of the attributes on the path.
Note: The equivalence takes care of positive and negative examples.

Cumulative Development

- **Example 1.7.** Learning from very few examples using background knowledge:
  1. Caveman Zog and the fish on a stick:



“Hey! Look what Zog do!”

Cumulative Development

- **Example 1.9.** Learning from very few examples using background knowledge:
  1. Caveman Zog and the fish on a stick:
  2. Generalizing from one Brazilian:
     Upon meeting her first Brazilian – Fernando – who speaks Portugese, Sarah
     - learns/generalizes that all Brazilians speak Portugese,
     - but not that all Brazilians are called Fernando.

SOME RIGHTS RESERVED

Cumulative Development

- **Example 1.11.** Learning from very few examples using background knowledge:
  1. Caveman Zog and the fish on a stick:
  2. Generalizing from one Brazilian:
  3. General rules about effectiveness of antibiotics:

     When Sarah – gifted in diagnostics, but clueless in pharmacology – observes a doctor prescribing the antibiotic Proxadone for an inflamed foot, she learns/infers that Proxadone is effective against this ailment.
- **Observation:** The methods/algorithms from ??? cannot replicate this. (why?)

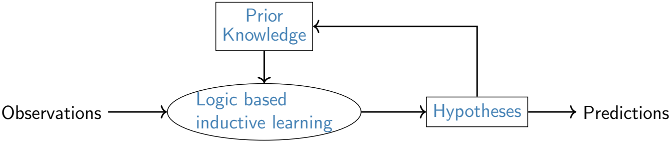
SOME RIGHTS RESERVED

Cumulative Development

- **Example 1.13.** Learning from very few examples using background knowledge:
  1. Caveman Zog and the fish on a stick:
  2. Generalizing from one Brazilian:
  3. General rules about effectiveness of antibiotics:
- **Observation:** The methods/algorithms from ??? cannot replicate this. (why?)
- **Missing Piece:** The background knowledge!
- **Problem:** To use background knowledge, need a method to obtain it. (use learning)
- **Question:** How to use knowledge to learn more efficiently?

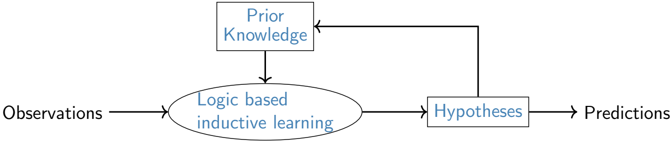
Cumulative Development

- **Example 1.15.** Learning from very few examples using background knowledge:
  1. Caveman Zog and the fish on a stick:
  2. Generalizing from one Brazilian:
  3. General rules about effectiveness of antibiotics:
- **Observation:** The methods/algorithms from ??? cannot replicate this. (why?)
- **Missing Piece:** The background knowledge!
- **Problem:** To use background knowledge, need a method to obtain it. (use learning)
- **Question:** How to use knowledge to learn more efficiently?
- **Answer:** Cumulative development: collect knowledge and use it in learning!



Cumulative Development

- **Example 1.17.** Learning from very few examples using background knowledge:
  1. Caveman Zog and the fish on a stick:
  2. Generalizing from one Brazilian:
  3. General rules about effectiveness of antibiotics:
- **Observation:** The methods/algorithms from ??? cannot replicate this. (why?)
- **Missing Piece:** The background knowledge!
- **Problem:** To use background knowledge, need a method to obtain it. (use learning)
- **Question:** How to use knowledge to learn more efficiently?
- **Answer:** Cumulative development: collect knowledge and use it in learning!



- **Definition 1.18.** We call the body of knowledge accumulated by (a group of) agents their background knowledge. It acts as prior knowledge in logic based learning processes.

SOME RIGHTS RESERVED

Adding Background Knowledge to Learning: Overview

- Explanation based learning (EBL)
- Relevance based learning (RBL)
- Knowledge based inductive learning (KBIL)

Three Principal Modes of Inference

- **Definition 1.19.** Deduction $\widehat{=}$ knowledge extension
- **Example 1.20.** $\dfrac{rains \Rightarrow wet\_street \quad rains}{wet\_street}\, D$

SOME RIGHTS RESERVED

Three Principal Modes of Inference

- **Definition 1.25.** Deduction $\widehat{=}$ knowledge extension
- **Example 1.26.** $\dfrac{rains \Rightarrow wet\_street \quad rains}{wet\_street}\, D$
- **Definition 1.27.** Abduction $\widehat{=}$ explanation
- **Example 1.28.** $\dfrac{rains \Rightarrow wet\_street \quad wet\_street}{rains}\, A$

SOME RIGHTS RESERVED

Three Principal Modes of Inference

- **Definition 1.31.** Deduction $\widehat{=}$ knowledge extension
- **Example 1.32.** $\dfrac{\mathit{rains} \Rightarrow \mathit{wet\_street} \quad \mathit{rains}}{\mathit{wet\_street}} D$
- **Definition 1.33.** Abduction $\widehat{=}$ explanation
- **Example 1.34.** $\dfrac{\mathit{rains} \Rightarrow \mathit{wet\_street} \quad \mathit{wet\_street}}{\mathit{rains}} A$
- **Definition 1.35.** Induction $\widehat{=}$ learning general rules from examples
- **Example 1.36.** $\dfrac{\mathit{wet\_street} \quad \mathit{rains}}{\mathit{rains} \Rightarrow \mathit{wet\_street}} I$

SOME RIGHTS RESERVED

29.2 Inductive Logic Programming

SOME RIGHTS RESERVED

Knowledge-based Inductive Learning

- **Idea:** Background knowledge and new hypothesis combine to explain the examples.
- **Example 2.1.** Inferring disease D from the symptoms is not enough to explain the prescription of medicine M.
  Need a new general rule: "*M is effective against D*" (induction from example)
- **Definition 2.2.** Knowledge based inductive learning (KBIL) replaces the explanation constraint by the KBIL constraint:

$$\mathit{Background} \wedge \mathit{Hypothesis} \wedge \mathit{Descriptions} \models \mathit{Classifications}$$

SOME RIGHTS RESERVED

Inductive Logic Programming

- **Definition 2.3.** Inductive logic programming (ILP) is logic based inductive learning method that uses logic programming as a uniform representation for examples, background knowledge and hypotheses.
  Given an encoding of the known background knowledge and a set of examples represented as a logical knowledge base of facts, an ILP system will derive a hypothesised logic program which entails all the positive and none of the negative examples.
- Main field of study for KBIL algorithms.
- Prior knowledge plays two key roles:
  1. The effective hypothesis space is reduced to include only those theories that are consistent with what is already known.
  2. Prior knowledge can be used to reduce the size of the hypothesis explaining the observations.
     - Smaller hypotheses are easier to find.
- **Observation:** ILP systems can formulate hypotheses in first-order logic.
  ⇝ Can learn in environments not understood by simpler systems.

SOME RIGHTS RESERVED

Inductive Logic Programming

- Combines inductive methods with the power of first-order representations.
- Offers a rigorous approach to the general KBIL problem.
- Offers complete algorithms for inducing general, first-order theories from examples.

SOME RIGHTS RESERVED

29.2.1 An Example

SOME RIGHTS RESERVED

ILP: An example

- General knowledge-based induction problem

$$\mathit{Background} \wedge \mathit{Hypothesis} \wedge \mathit{Descriptions} \models \mathit{Classifications}$$

- **Example 2.4 (Learning family relations from examples).**
  - Observations are an extended family tree
    - mother, father and married relations
    - male and female properties
  - Target predicates: grandparent, BrotherInLaw, Ancestor

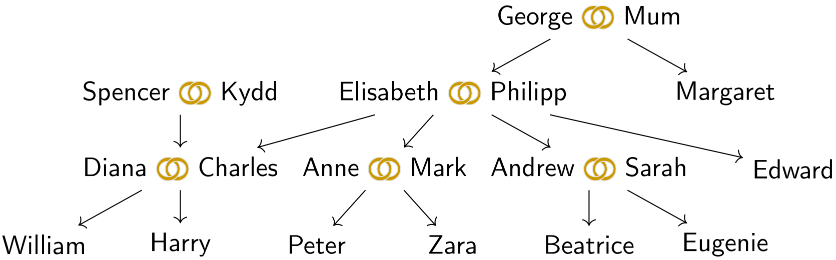
  ⇝ The goal is to find a logical formula that serves as a *definition* of the target predicates
  - equivalently: A Prolog program that *computes* the value of the target predicate

  ⇝ We obtain a perfectly comprehensible hypothesis

SOME RIGHTS RESERVED

British Royalty Family Tree (not quite not up to date)

- The facts about kinship and relations can be visualized as a family tree:



Example

- Descriptions include facts like
  - $\text{father}(\mathit{Philip}, \mathit{Charles})$
  - $\text{mother}(\mathit{Mum}, \mathit{Margaret})$
  - $\text{married}(\mathit{Diana}, \mathit{Charles})$
  - $\text{male}(\mathit{Philip})$
  - $\text{female}(\mathit{Beatrice})$
- Sentences in classifications depend on the target concept being learned (in the example: 12 positive, 388 negative)
  - $\text{grandparent}(\mathit{Mum}, \mathit{Charles})$
  - $\neg\text{grandparent}(\mathit{Mum}, \mathit{Harry})$
- **Goal:** Find a set of sentences for hypothesis such that the entailment constraint is satisfied.
- **Example 2.5.** Without background knowledge, define grandparent in terms of mother and father.

$$\text{grandparent}(x,y) \Leftrightarrow (\exists z.\text{mother}(x,z) \wedge \text{mother}(z,y)) \vee (\exists z.\text{mother}(x,z) \wedge \text{father}(z,y)) \vee \ldots \vee (\exists z.\text{f}$$

Why Attribute-based Learning Fails

- **Observation:** Decision tree learning will get nowhere!
  - To express Grandparent as a (Boolean) attribute, pairs of people need to be objects $Grandparent(\langle Mum, Charles\rangle)$.
  - But then the example descriptions can not be represented

$$FirstElementIsMotherOfElizabeth(\langle Mum, Charles\rangle)$$

  - A large disjunction of specific cases without any hope of generalization to new examples.
- **Generally:** Attribute-based learning algorithms are incapable of learning relational predicates.

SOME RIGHTS RESERVED

Background knowledge

- **Observation:** A little bit of background knowledge helps a lot.
- **Example 2.6.** If the background knowledge contains

  $$\mathrm{parent}(x,y)\Leftrightarrow\mathrm{mother}(x,y)\vee\mathrm{father}(x,y)$$

  then Grandparent can be reduced to

  $$\mathrm{grandparent}(x,y)\Leftrightarrow(\exists z.\mathrm{parent}(x,z)\wedge\mathrm{parent}(z,y))$$

- **Definition 2.7.** A constructive induction algorithm creates new predicates to facilitate the expression of explanatory hypotheses.
- **Example 2.8.** Use constructive induction to introduce a predicate parent to simplify the definitions of the target predicates.

SOME RIGHTS RESERVED

29.2.2 Top-Down Inductive Learning: FOIL

SOME RIGHTS RESERVED

Top-Down Inductive Learning

- Bottom-up learning; e.g. Decision-tree learning: start from the observations and work backwards.
  - Decision tree is gradually grown until it is consistent with the observations.
- Top-down learning method
  - start from a general rule and specialize it on every example.

SOME RIGHTS RESERVED

Top-Down Inductive Learning: FOIL

- Split positive and negative examples
  - Positive: $\langle George, Anne\rangle$, $\langle Philip, Peter\rangle$, $\langle Spencer, Harry\rangle$
  - Negative: $\langle George, Elizabeth\rangle$, $\langle Harry, Zara\rangle$, $\langle Charles, Philip\rangle$
- Construct a set of Horn clauses with head $\mathrm{grandfather}(x,y)$ such that the positive examples are instances of the grandfather relationship.
  - Start with a clause with an empty body $\Rightarrow\mathrm{grandfather}(x,y)$.
  - All examples are now classified as positive, so specialize to rule out the negative examples: Here are 3 potential additions:
    1. $\mathrm{father}(x,y) \Rightarrow \mathrm{grandfather}(x,y)$
    2. $\mathrm{parent}(x,z) \Rightarrow \mathrm{grandfather}(x,y)$
    3. $\mathrm{father}(x,z) \Rightarrow \mathrm{grandfather}(x,y)$
  - The first one incorrectly classifies the 12 positive examples.
  - The second one is incorrect on a larger part of the negative examples.
  - Prefer the third clause and specialize to $\mathrm{father}(x,z) \wedge \mathrm{parent}(z,y) \Rightarrow \mathrm{grandfather}(x,y)$.

SOME RIGHTS RESERVED

FOIL

```
function Foil(examples,target) returns a set of Horn clauses
    inputs: examples, set of examples
             target, a literal for the goal predicate
    local variables: clauses, set of clauses, initially empty
    while examples contains positive examples do
      clause := New−Clause(examples,target)
      remove examples covered by clause from examples
      add clause to clauses
    return clauses
```

FOIL

```
function New−Clause(examples,target) returns a Horn clause
   local variables: clause, a clause with target as head and an empty body
       l, a literal to be added to the clause
       extendedExamples, a set of examples with values for new variables
   extendedExamples := examples
   while extendedExamples contains negative examples do
       l := Choose−Literal(New−Literals(clause),extendedExamples)
       append l to the body of clause
       extendedExamples := map Extend−Example over extendedExamples
   return clause

function Extend−Example(example,literal) returns a new example
  if example satisfies literal
  then return the set of examples created by extending example with each
           possible constant value for each new variable in literal
  else return the empty set

function New−Literals(clause) returns a set of possibly ‘‘useful’’ literals
function Choose−Literal(literals) returns the ‘‘best’’ literal from literals
```

SOME RIGHTS RESERVED

FOIL: Choosing Literals

- New-Literals: Takes a clause and constructs all possibly “useful” literals
- $\mathrm{father}(x, z) \Rightarrow \mathrm{grandfather}(x, y)$
- Add literals using predicates
  - Negated or unnegated
  - Use any existing predicate (including the goal)
  - Arguments must be variables
  - Each literal must include at least one variable from an earlier literal or from the head of the clause
    - Valid: $\mathit{Mother}(z, u), \mathit{Married}(z, z), \mathrm{grandfather}(v, x)$
    - Invalid: $\mathit{Married}(u, v)$
- Equality and inequality literals
  - E.g. $z \neq x$, empty list
- Arithmetic comparisons
  - E.g. $x > y$, threshold values

SOME RIGHTS RESERVED

FOIL: Choosing Literals

- The way New-Literal changes the clauses leads to a very large branching factor.
- Improve performance by using type information:
  - E.g., $\mathrm{parent}(x, n)$ where x is a person and n is a number
- Choose-Literal uses a heuristic similar to information gain.
- Ockham's razor to eliminate hypotheses.
  - If the clause becomes longer than the total length of the positive examples that the clause explains, this clause is not a valid hypothesis.
- Most impressive demonstration
  - Learn the correct definition of list-processing functions in Prolog from a small set of examples, using previously learned functions as background knowledge.

SOME RIGHTS RESERVED

29.2.3 Inverse Resolution

SOME RIGHTS RESERVED

Inverse Resolution

- **Definition 2.9.** Inverse resolution in a nutshell
  - Classifications follows from *Background* $\wedge$ *Hypothesis* $\wedge$ *Descriptions*.
  - This can be proven by resolution.
  - Run the proof backwards to find hypothesis.

SOME RIGHTS RESERVED

Inverse Resolution

- **Definition 2.10.** Inverse resolution in a nutshell
  - Classifications follows from *Background* $\wedge$ *Hypothesis* $\wedge$ *Descriptions*.
  - This can be proven by resolution.
  - Run the proof backwards to find hypothesis.
- **Problem:** How to run the resolution proof backwards?

SOME RIGHTS RESERVED

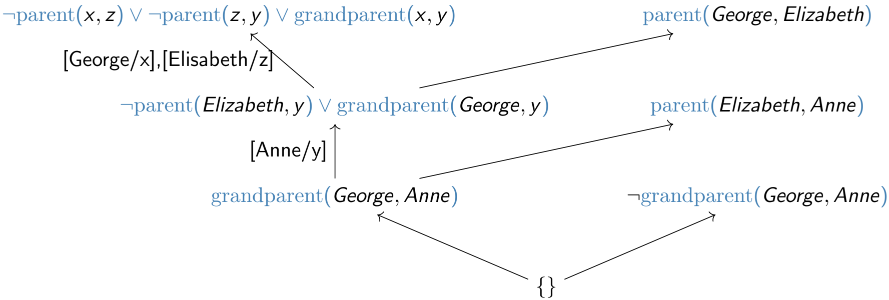
Inverse Resolution

- **Definition 2.11.** Inverse resolution in a nutshell
  - Classifications follows from *Background* $\wedge$ *Hypothesis* $\wedge$ *Descriptions*.
  - This can be proven by resolution.
  - Run the proof backwards to find hypothesis.
- **Problem:** How to run the resolution proof backwards?
- **Recap:** In ordinary resolution we take two clauses $C_1 = L \vee R_1$ and $C_2 = \neg L \vee R_2$ and resolve them to produce the resolvent $C = R_1 \vee R_2$.
- **Idea:** Two possible variants of inverse resolution:
  - Take resolvent C and produce two clauses C_1 and C_2.
  - Take C and C_1 and produce C_2.

SOME RIGHTS RESERVED

Generating Inverse Proofs (Example)

1. Start with an example classified as both positive and negative (Need a contradiction)
2. Invent clauses that resolve with a fact in our knowledge base



$\neg\mathrm{parent}(x,z) \vee \neg\mathrm{parent}(z,y) \vee \mathrm{grandparent}(x,y)$

$\mathrm{parent}(George, Elizabeth)$

[George/x],[Elisabeth/z]

$\neg\mathrm{parent}(Elizabeth,y) \vee \mathrm{grandparent}(George,y)$

$\mathrm{parent}(Elizabeth, Anne)$

[Anne/y]

$\mathrm{grandparent}(George, Anne)$

$\neg\mathrm{grandparent}(George, Anne)$

$\{\}$

$\neg\mathrm{parent}(x,z) \vee \neg\mathrm{parent}(z,y) \vee \mathrm{grandparent}(x,y)$ is equivalent to $\mathrm{parent}(x,z) \wedge \mathrm{parent}(z,y) \Rightarrow \mathrm{grandparent}(x,y)$

Generating Inverse Proofs

- Inverse resolution is a search algorithm: For any C and C_1 there can be several or even an infinite number of clauses C_2.
- **Example 2.12.** Instead of $\mathrm{parent}(George, Elizabeth)$ there were numerous alternatives we could have picked!
- The clauses C_1 that participate in each step can be chosen from Background, Descriptions, Classifications or from hypothesized clauses already generated.
- ILP needs restrictions to make the search manageable
  - Eliminate function symbols
  - Generate only the most specific hypotheses
  - Use Horn clauses
  - All hypothesized clauses must be consistent with each other
  - Each hypothesized clause must agree with the observations

SOME RIGHTS RESERVED

New Predicates and New Knowledge

- An inverse resolution procedure is a complete algorithm for learning first-order theories:
  - If some unknown hypothesis generates a set of examples, then an inverse resolution procedure can generate hypothesis from the examples.
- Can inverse resolution infer the law of gravity from examples of falling bodies?
  - Yes, given suitable background mathematics!
- Monkey and typewriter problem: How to overcome the large branching factor and the lack of structure in the search space?

SOME RIGHTS RESERVED

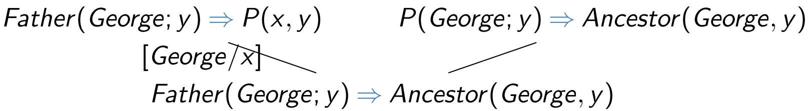
New Predicates and New Knowledge

- Inverse resolution is capable of generating new predicates:
    - Resolution of C_1 and C_2 into C eliminates a literal that C_1 and C_2 share.
    - This literal might contain a predicate that does not appear in C.
    - When working backwards, one possibility is to generate a new predicate from which to construct the missing literal.

SOME RIGHTS RESERVED

New Predicates and New Knowledge

- **Example 2.13.**



$Father(George; y) \Rightarrow P(x, y)$ $\quad$ $P(George; y) \Rightarrow Ancestor(George, y)$

$[George/x]$

$Father(George; y) \Rightarrow Ancestor(George, y)$

P can be used in later inverse resolution steps.

- **Example 2.14.** $\mathrm{mother}(x, y) \Rightarrow P(x, y)$ or $\mathrm{father}(x, y) \Rightarrow P(x, y)$ leading to the “Parent” relationship.
- Inventing new predicates is important to reduce the size of the definition of the goal predicate.
- Some of the deepest revolutions in science come from the invention of new predicates. (e.g. Galileo’s invention of acceleration)

SOME RIGHTS RESERVED

Applications of ILP

- ILP systems have outperformed knowledge free methods in a number of domains.
- Molecular biology: the GOLEM system has been able to generate high-quality predictions of protein structures and the therapeutic efficacy of various drugs.
- GOLEM is a completely general-purpose program that is able to make use of background knowledge about any domain.

SOME RIGHTS RESERVED

Part 7
Natural Language

SOME RIGHTS RESERVED

Fascination of (Natural) Language

- **Definition 2.15.** A natural language is any form of spoken or signed means of communication that has evolved naturally in humans through use and repetition without conscious planning or premeditation.
- **In other words:** the language you use all day long, e.g. English, German, . . .
- **Why Should we care about natural language?:**
  - Even more so than thinking, language is a skill that only humans have.
  - It is a miracle that we can express complex thoughts in a sentence in a matter of seconds.
  - It is no less miraculous that a child can learn tens of thousands of words and complex syntax in a matter of a few years.

SOME RIGHTS RESERVED

Natural Language and AI

- Without natural language capabilities (understanding and generation) no AI!
- Ca. 100.000 years ago, humans learned to speak, ca. 7.000 years ago, to write.
- Alan Turing based his test on natural language: (for good reason)
  - We want AI agents to be able to communicate with humans.
  - We want AI agents to be able to acquire knowledge from written documents.
- In this part, we analyze the problem with specific information-seeking tasks:
  - Language models (Which strings are English/Spanish/etc.)
  - Text classification (E.g. spam detection)
  - Information retrieval (aka. Search Engines)
  - Information extraction (finding objects and their relations in texts)

SOME RIGHTS RESERVED

Chapter 30
Natural Language Processing

SOME RIGHTS RESERVED

30.1 Introduction to NLP

SOME RIGHTS RESERVED

What is Natural Language Processing?

- **Generally:** Studying of natural languages and development of systems that can use/generate these.
- **Definition 1.1.** Natural language processing (NLP) is an engineering field at the intersection of computer science, AI, and linguistics which is concerned with the interactions between computers and human (natural) languages. Most challenges in NLP involve:
  - Natural language understanding (NLU) that is, enabling computers to derive meaning (representations) from human or natural language input.
  - Natural language generation (NLG) which aims at generating natural language or speech from meaning representation.
- For communication with/among humans we need both NLU and NLG.

SOME RIGHTS RESERVED

Language Technology

- Language Assistance:
    - written language: Spell/grammar/style-checking,
    - spoken language: dictation systems and screen readers,
    - multilingual text: machine-supported text and dialog translation, eLearning.

Language Technology

- Language Assistance:
  - written language: Spell/grammar/style-checking,
  - spoken language: dictation systems and screen readers,
  - multilingual text: machine-supported text and dialog translation, eLearning.
- Information management:
  - search and classification of documents, (e.g. Google/Bing)
  - information extraction, question answering. (e.g. `http://ask.com`)

SOME RIGHTS RESERVED

Language Technology

- Language Assistance:
  - written language: Spell/grammar/style-checking,
  - spoken language: dictation systems and screen readers,
  - multilingual text: machine-supported text and dialog translation, eLearning.
- Information management:
  - search and classification of documents, (e.g. Google/Bing)
  - information extraction, question answering. (e.g. `http://ask.com`)
- Dialog Systems/Interfaces:
  - information systems: at airport, tele-banking, e-commerce, call centers,
  - dialog interfaces for computers, robots, cars. (e.g. Siri/Alexa)

SOME RIGHTS RESERVED

Language Technology

- Language Assistance:
  - written language: Spell/grammar/style-checking,
  - spoken language: dictation systems and screen readers,
  - multilingual text: machine-supported text and dialog translation, eLearning.
- Information management:
  - search and classification of documents, (e.g. Google/Bing)
  - information extraction, question answering. (e.g. `http://ask.com`)
- Dialog Systems/Interfaces:
  - information systems: at airport, tele-banking, e-commerce, call centers,
  - dialog interfaces for computers, robots, cars. (e.g. Siri/Alexa)
- **Observation:** The earlier technologies largely rely on pattern matching, the latter ones need to compute the meaning of the input utterances, e.g. for database lookups in information systems.

SOME RIGHTS RESERVED

30.2 Natural Language and its Meaning

SOME RIGHTS RESERVED

What is (NL) Semantics? Answers from various Disciplines!

- **Observation:** Different (academic) disciplines specialize the notion of semantics (of natural language) in different ways.

SOME RIGHTS RESERVED

What is (NL) Semantics? Answers from various Disciplines!

- **Observation:** Different (academic) disciplines specialize the notion of semantics (of natural language) in different ways.
- **Philosophy:** has a long history of trying to answer it, e.g.
  - Platon ⇝ cave allegory, Aristotle ⇝ syllogisms.
  - Frege/Russell ⇝ sense vs. referent. ("*Michael Kohlhase*" vs. "*Odysseus*")

SOME RIGHTS RESERVED

What is (NL) Semantics? Answers from various Disciplines!

- **Observation:** Different (academic) disciplines specialize the notion of semantics (of natural language) in different ways.
- **Philosophy:** has a long history of trying to answer it, e.g.
  - Platon ⇝ cave allegory, Aristotle ⇝ syllogisms.
  - Frege/Russell ⇝ sense vs. referent. ("*Michael Kohlhase*" vs. "*Odysseus*")
- **Linguistics/Language Philosophy:** We need semantics e.g. in translation "*Der Geist ist willig aber das Fleisch ist schwach!*" vs. "*Der Schnaps ist gut, aber der Braten ist verkocht!*" (meaning counts)

SOME RIGHTS RESERVED

What is (NL) Semantics? Answers from various Disciplines!

- **Observation:** Different (academic) disciplines specialize the notion of semantics (of natural language) in different ways.
- **Philosophy:** has a long history of trying to answer it, e.g.
  - Platon ⇝ cave allegory, Aristotle ⇝ syllogisms.
  - Frege/Russell ⇝ sense vs. referent. ("*Michael Kohlhase*" vs. "*Odysseus*")
- **Linguistics/Language Philosophy:** We need semantics e.g. in translation "*Der Geist ist willig aber das Fleisch ist schwach!*" vs. "*Der Schnaps ist gut, aber der Braten ist verkocht!*" (meaning counts)
- **Psychology/Cognition:** Semantics $\widehat{=}$ "what is in our brains" (⇝ mental models)

SOME RIGHTS RESERVED

What is (NL) Semantics? Answers from various Disciplines!

- **Observation:** Different (academic) disciplines specialize the notion of semantics (of natural language) in different ways.
- **Philosophy:** has a long history of trying to answer it, e.g.
  - Platon ⇝ cave allegory, Aristotle ⇝ syllogisms.
  - Frege/Russell ⇝ sense vs. referent. ("*Michael Kohlhase*" vs. "*Odysseus*")
- **Linguistics/Language Philosophy:** We need semantics e.g. in translation "*Der Geist ist willig aber das Fleisch ist schwach!*" vs. "*Der Schnaps ist gut, aber der Braten ist verkocht!*" (meaning counts)
- **Psychology/Cognition:** Semantics $\widehat{=}$ "what is in our brains" (⇝ mental models)
- **Mathematics** has driven much of modern logic in the quest for foundations.
  - Logic as "foundation of mathematics" solved as far as possible
  - In daily practice syntax and semantics are not differentiated (much).

SOME RIGHTS RESERVED

What is (NL) Semantics? Answers from various Disciplines!

- **Observation:** Different (academic) disciplines specialize the notion of semantics (of natural language) in different ways.
- **Philosophy:** has a long history of trying to answer it, e.g.
  - Platon ⇝ cave allegory, Aristotle ⇝ syllogisms.
  - Frege/Russell ⇝ sense vs. referent. ("*Michael Kohlhase*" vs. "*Odysseus*")
- **Linguistics/Language Philosophy:** We need semantics e.g. in translation "*Der Geist ist willig aber das Fleisch ist schwach!*" vs. "*Der Schnaps ist gut, aber der Braten ist verkocht!*" (meaning counts)
- **Psychology/Cognition:** Semantics $\widehat{=}$ "what is in our brains" (⇝ mental models)
- **Mathematics** has driven much of modern logic in the quest for foundations.
  - Logic as "foundation of mathematics" solved as far as possible
  - In daily practice syntax and semantics are not differentiated (much).
- **Logic@AI/CS** tries to define meaning and compute with them. (applied semantics)
  - makes syntax explicit in a formal language (formulae, sentences)
  - defines truth/validity by mapping sentences into "world" (interpretation)
  - gives rules of truth-preserving reasoning (inference)

SOME RIGHTS RESERVED

Meaning of Natural Language; e.g. Machine Translation

- **Idea:** Machine translation is very simple! (we have good lexica)
- **Example 2.1.** "*Peter liebt Maria.*" $\leadsto$ "*Peter loves Mary.*"
- ⚠ this only works for simple examples!
- **Example 2.2.** "*Wirf der Kuh das Heu über den Zaun.*" $\not\leadsto$ "*Throw the cow the hay over the fence.*" (differing grammar; Google Translate)
- **Example 2.3.** ⚠ Grammar is not the only problem
  - "*Der Geist ist willig, aber das Fleisch ist schwach!*"
  - "*Der Schnaps ist gut, aber der Braten ist verkocht!*"
- **Observation 2.4.** *We have to understand the meaning for high-quality translation!*

SOME RIGHTS RESERVED

Language and Information

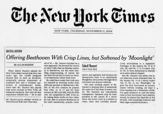
- **Observation:** Humans use words (sentences, texts) in natural languages to represent and communicate information.
- **But:** What really counts is not the words themselves, but the meaning information they carry.

SOME RIGHTS RESERVED

Language and Information

- **Observation:** Humans use words (sentences, texts) in natural languages to represent and communicate information.
- **But:** What really counts is not the words themselves, but the meaning information they carry.
- **Example 2.7 (Word Meaning).**

“*Newspaper*” ⇝



The New York Times

NEW YORK, THURSDAY, NOVEMBER 11, 2004

RECITAL REVIEW

Offering Beethoven With Crisp Lines, but Softened by 'Moonlight'

By ALLAN KOZINN

Soheil Nasseri

Alice Tully Hall

- For questions/answers, it would be very useful to find out what words (sentences/texts) mean.

SOME RIGHTS RESERVED

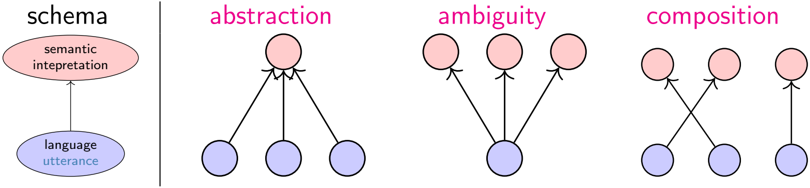
Language and Information

- **Observation:** Humans use words (sentences, texts) in natural languages to represent and communicate information.
- **But:** What really counts is not the words themselves, but the meaning information they carry.
- **Example 2.9 (Word Meaning).**

“*Newspaper*” ⇝

The New York Times
Offering Beethoven With Crisp Lines, but Softened by 'Moonlight'
The New York Times

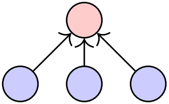
- For questions/answers, it would be very useful to find out what words (sentences/texts) mean.
- **Definition 2.10.** Interpretation of natural language utterances: three problems



SOME RIGHTS RESERVED

Language and Information (Examples)

- **Example 2.11 (Abstraction).**

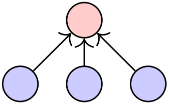


"*Car*" and "*automobile*" have the same meaning.

SOME RIGHTS RESERVED

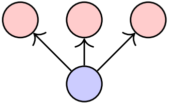
Language and Information (Examples)

- **Example 2.14 (Abstraction).**



"*Car*" and "*automobile*" have the same meaning.

- **Example 2.15 (Ambiguity).**

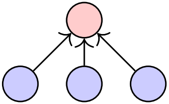


A "*bank*" can be a financial institution or a geographical feature.

SOME RIGHTS RESERVED

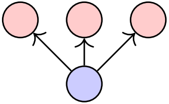
Language and Information (Examples)

- **Example 2.17 (Abstraction).**



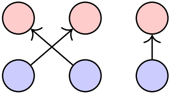
"*Car*" and "*automobile*" have the same meaning.

- **Example 2.18 (Ambiguity).**



A "*bank*" can be a financial institution or a geographical feature.

- **Example 2.19 (Composition).**

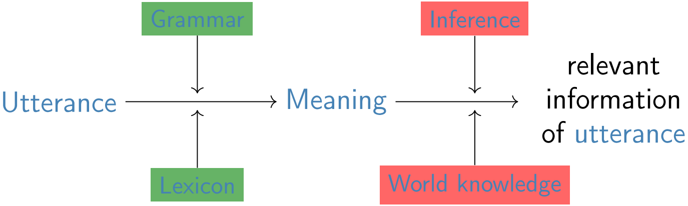


"*Every student sleeps*" $\rightsquigarrow \forall x.student(x) \Rightarrow sleep(x)$

SOME RIGHTS RESERVED

Context Contributes to the Meaning of NL Utterances

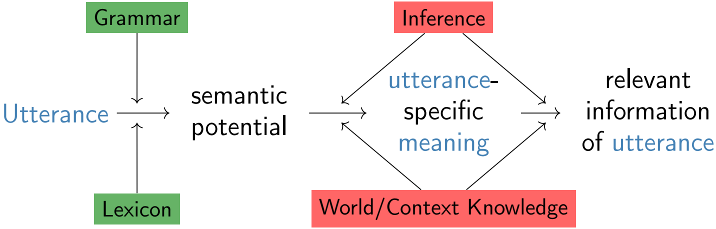
- **Observation:** Not all information conveyed is linguistically realized in an utterance.
- **Example 2.20.** "*The lecture begins at 11:00 am.*" What lecture? Today?
- **Definition 2.21.** We call a piece *i* of information linguistically realized in an utterance *U*, iff, we can trace *i* to a fragment of *U*.
- **Definition 2.22 (Possible Mechanism).** Inferring the missing pieces from the context and world knowledge:



We call this process semantic/pragmatic analysis.

Context Contributes to the Meaning of NL Utterances

- **Example 2.23.** "*It starts at eleven.*" What starts?
- Before we can resolve the time, we need to resolve the anaphor "*it*".
- **Possible Mechanism:** More Inference!


Grammar
Lexicon
Utterance → semantic potential → utterance-specific meaning → relevant information of utterance
Inference
World/Context Knowledge

⇝ Semantic/pragmatic analysis is quite complex! (prime topic of AI-2)

SOME RIGHTS RESERVED

30.3 Looking at Natural Language

SOME RIGHTS RESERVED

Fun with Diamonds (are they real?) [**Davidson:tam67**]

- **Example 3.1.** We study the truth conditions of adjectival complexes:
  - "*This is a diamond.*" ($\models$ *diamond*)

SOME RIGHTS RESERVED

Fun with Diamonds (are they real?) [**Davidson:tam67**]

- **Example 3.2.** We study the truth conditions of adjectival complexes:
  - "*This is a diamond.*" ($\models$ *diamond*)
  - "*This is a blue diamond.*" ($\models$ *diamond*, $\models$ *blue*)

Fun with Diamonds (are they real?) [**Davidson:tam67**]

- **Example 3.3.** We study the truth conditions of adjectival complexes:
  - "*This is a diamond.*" ($\models$ *diamond*)
  - "*This is a blue diamond.*" ($\models$ *diamond*, $\models$ *blue*)
  - "*This is a big diamond.*" ($\models$ *diamond*, $\not\models$ *big*)

Fun with Diamonds (are they real?) [**Davidson:tam67**]

- **Example 3.4.** We study the truth conditions of adjectival complexes:
  - "*This is a diamond.*" ($\models \mathit{diamond}$)
  - "*This is a blue diamond.*" ($\models \mathit{diamond}, \models \mathit{blue}$)
  - "*This is a big diamond.*" ($\models \mathit{diamond}, \not\models \mathit{big}$)
  - "*This is a fake diamond.*" ($\models \neg\mathit{diamond}$)

Fun with Diamonds (are they real?) [**Davidson:tam67**]

- **Example 3.5.** We study the truth conditions of adjectival complexes:
  - "*This is a diamond.*" ($\models$ *diamond*)
  - "*This is a blue diamond.*" ($\models$ *diamond*, $\models$ *blue*)
  - "*This is a big diamond.*" ($\models$ *diamond*, $\not\models$ *big*)
  - "*This is a fake diamond.*" ($\models \neg$*diamond*)
  - "*This is a fake blue diamond.*" ($\models$ *blue*?, $\models$ *diamond*?)

SOME RIGHTS RESERVED

Fun with Diamonds (are they real?) [**Davidson:tam67**]

- **Example 3.6.** We study the truth conditions of adjectival complexes:
  - "*This is a diamond.*" ($\models$ *diamond*)
  - "*This is a blue diamond.*" ($\models$ *diamond*, $\models$ *blue*)
  - "*This is a big diamond.*" ($\models$ *diamond*, $\not\models$ *big*)
  - "*This is a fake diamond.*" ($\models \neg$*diamond*)
  - "*This is a fake blue diamond.*" ($\models$ *blue*?, $\models$ *diamond*?)
  - "*Mary knows that this is a diamond.*" ($\models$ *diamond*)

SOME RIGHTS RESERVED

Fun with Diamonds (are they real?) [**Davidson:tam67**]

- **Example 3.7.** We study the truth conditions of adjectival complexes:
  - "*This is a diamond.*" ($\models$ *diamond*)
  - "*This is a blue diamond.*" ($\models$ *diamond*, $\models$ *blue*)
  - "*This is a big diamond.*" ($\models$ *diamond*, $\not\models$ *big*)
  - "*This is a fake diamond.*" ($\models \neg$*diamond*)
  - "*This is a fake blue diamond.*" ($\models$ *blue*?, $\models$ *diamond*?)
  - "*Mary knows that this is a diamond.*" ($\models$ *diamond*)
  - "*Mary believes that this is a diamond.*" ($\not\models$ *diamond*)

SOME RIGHTS RESERVED

Ambiguity: The dark side of Meaning

- **Definition 3.8.** We call an utterance ambiguous, iff it has multiple meanings, which we call readings.
- **Example 3.9.** All of the following sentences are ambiguous:
  - "*John went to the bank.*" (river or financial?)

Ambiguity: The dark side of Meaning

- **Definition 3.10.** We call an utterance ambiguous, iff it has multiple meanings, which we call readings.
- **Example 3.11.** All of the following sentences are ambiguous:
  - "*John went to the bank.*" (river or financial?)
  - "*You should have seen the bull we got from the pope.*" (three readings!)

SOME RIGHTS RESERVED

Ambiguity: The dark side of Meaning

- **Definition 3.12.** We call an utterance ambiguous, iff it has multiple meanings, which we call readings.
- **Example 3.13.** All of the following sentences are ambiguous:
  - "*John went to the bank.*" (river or financial?)
  - "*You should have seen the bull we got from the pope.*" (three readings!)
  - "*I saw her duck.*" (animal or action?)

SOME RIGHTS RESERVED

Ambiguity: The dark side of Meaning

- **Definition 3.14.** We call an utterance ambiguous, iff it has multiple meanings, which we call readings.
- **Example 3.15.** All of the following sentences are ambiguous:
  - "*John went to the bank.*" (river or financial?)
  - "*You should have seen the bull we got from the pope.*" (three readings!)
  - "*I saw her duck.*" (animal or action?)
  - "*John chased the gangster in the red sports car.*" (three-way too!)

SOME RIGHTS RESERVED

Quantifiers, Scope and Context

- **Example 3.16.** "*Every man loves a woman.*" (Keira Knightley or his mother!)

SOME RIGHTS RESERVED

Quantifiers, Scope and Context

- **Example 3.21.** "*Every man loves a woman.*" (Keira Knightley or his mother!)
- **Example 3.22.** "*Every car has a radio.*" (only one reading!)

Quantifiers, Scope and Context

- **Example 3.26.** “*Every man loves a woman.*” (Keira Knightley or his mother!)
- **Example 3.27.** “*Every car has a radio.*” (only one reading!)
- **Example 3.28.** “*Some student in every course sleeps in every class at least some of the time.*” (how many readings?)

SOME RIGHTS RESERVED

Quantifiers, Scope and Context

- **Example 3.31.** "*Every man loves a woman.*" (Keira Knightley or his mother!)
- **Example 3.32.** "*Every car has a radio.*" (only one reading!)
- **Example 3.33.** "*Some student in every course sleeps in every class at least some of the time.*" (how many readings?)
- **Example 3.34.** "*The president of the US is having an affair with an intern.*" (2002 or 2000?)

SOME RIGHTS RESERVED

Quantifiers, Scope and Context

- **Example 3.36.** "*Every man loves a woman.*" (Keira Knightley or his mother!)
- **Example 3.37.** "*Every car has a radio.*" (only one reading!)
- **Example 3.38.** "*Some student in every course sleeps in every class at least some of the time.*" (how many readings?)
- **Example 3.39.** "*The president of the US is having an affair with an intern.*" (2002 or 2000?)
- **Example 3.40.** "*Everyone is here.*" (who is everyone?)

SOME RIGHTS RESERVED

More Context: Anaphora – Challenge for Pragmatic Analysis

- **Example 3.41 (Anaphoric References).**
  - "*John is a bachelor. His wife is very nice.*" (Uh, what?, who?)

SOME RIGHTS RESERVED

More Context: Anaphora – Challenge for Pragmatic Analysis

- **Example 3.45 (Anaphoric References).**
  - "*John is a bachelor. His wife is very nice.*" (Uh, what?, who?)
  - "*John likes his dog Spiff even though he bites him sometimes.*" (who bites?)

SOME RIGHTS RESERVED

More Context: Anaphora – Challenge for Pragmatic Analysis

- **Example 3.49 (Anaphoric References).**
  - "*John is a bachelor. His wife is very nice.*" (Uh, what?, who?)
  - "*John likes his dog Spiff even though he bites him sometimes.*" (who bites?)
  - "*John likes Spiff. Peter does too.*" (what to does Peter do?)

SOME RIGHTS RESERVED

More Context: Anaphora – Challenge for Pragmatic Analysis

- **Example 3.53 (Anaphoric References).**
  - "*John is a bachelor. His wife is very nice.*" (Uh, what?, who?)
  - "*John likes his dog Spiff even though he bites him sometimes.*" (who bites?)
  - "*John likes Spiff. Peter does too.*" (what to does Peter do?)
  - "*John loves his wife. Peter does too.*" (whom does Peter love?)

SOME RIGHTS RESERVED

More Context: Anaphora – Challenge for Pragmatic Analysis

- **Example 3.57 (Anaphoric References).**
  - "*John is a bachelor. His wife is very nice.*" (Uh, what?, who?)
  - "*John likes his dog Spiff even though he bites him sometimes.*" (who bites?)
  - "*John likes Spiff. Peter does too.*" (what to does Peter do?)
  - "*John loves his wife. Peter does too.*" (whom does Peter love?)
  - "*John loves golf, and Mary too.*" (who does what?)

SOME RIGHTS RESERVED

More Context: Anaphora – Challenge for Pragmatic Analysis

- **Example 3.61 (Anaphoric References).**
  - "*John is a bachelor. His wife is very nice.*" (Uh, what?, who?)
  - "*John likes his dog Spiff even though he bites him sometimes.*" (who bites?)
  - "*John likes Spiff. Peter does too.*" (what to does Peter do?)
  - "*John loves his wife. Peter does too.*" (whom does Peter love?)
  - "*John loves golf, and Mary too.*" (who does what?)
- **Definition 3.62.** A word or phrase is called anaphoric (or an anaphor), if its interpretation depends upon another phrase in context. In a narrower sense, an anaphor refers to an earlier phrase (its antecedent), while a cataphor to a later one (its postcedent).

  Definition 3.63. The process of determining the antecedent or postcedent of an anaphoric phrase is called anaphor resolution.

  Definition 3.64. An anaphoric connection between anaphor and its antecedent or postcedent is called direct, iff it can be understood purely syntactically. An anaphoric connection is called indirect or a bridging reference if additional knowledge is needed.

SOME RIGHTS RESERVED

More Context: Anaphora – Challenge for Pragmatic Analysis

- **Example 3.65 (Anaphoric References).**
  - "*John is a bachelor. His wife is very nice.*" (Uh, what?, who?)
  - "*John likes his dog Spiff even though he bites him sometimes.*" (who bites?)
  - "*John likes Spiff. Peter does too.*" (what to does Peter do?)
  - "*John loves his wife. Peter does too.*" (whom does Peter love?)
  - "*John loves golf, and Mary too.*" (who does what?)
- **Definition 3.66.** A word or phrase is called anaphoric (or an anaphor), if its interpretation depends upon another phrase in context. In a narrower sense, an anaphor refers to an earlier phrase (its antecedent), while a cataphor to a later one (its postcedent).

  Definition 3.67. The process of determining the antecedent or postcedent of an anaphoric phrase is called anaphor resolution.

  Definition 3.68. An anaphoric connection between anaphor and its antecedent or postcedent is called direct, iff it can be understood purely syntactically. An anaphoric connection is called indirect or a bridging reference if additional knowledge is needed.
- Anaphora are another example, where natural languages use the inferential capabilities of the hearer/reader to "shorten" utterances.
- Anaphora challenge pragmatic analysis, since they can only be resolved from the context using world knowledge.

SOME RIGHTS RESERVED

Context is Personal and Keeps Changing

- **Example 3.69.** Consider the following sentences involving definite description:
  1. "*The king of America is rich.*" (true or false?)

How do the interact with your context and world knowledge?

SOME RIGHTS RESERVED

Context is Personal and Keeps Changing

- **Example 3.70.** Consider the following sentences involving definite description:
  1. "*The king of America is rich.*" (true or false?)
  2. "*The king of America isn't rich.*" (false or true?)

How do the interact with your context and world knowledge?

SOME RIGHTS RESERVED

Context is Personal and Keeps Changing

- **Example 3.71.** Consider the following sentences involving definite description:
  1. "*The king of America is rich.*" (true or false?)
  2. "*The king of America isn't rich.*" (false or true?)
  3. "*If America had a king, the king of America would be rich.*" (true or false!)

How do the interact with your context and world knowledge?

SOME RIGHTS RESERVED

Context is Personal and Keeps Changing

- **Example 3.72.** Consider the following sentences involving definite description:
  1. "*The king of America is rich.*" (true or false?)
  2. "*The king of America isn't rich.*" (false or true?)
  3. "*If America had a king, the king of America would be rich.*" (true or false!)
  4. "*The king of Buganda is rich.*" (Where is Buganda?)

  How do the interact with your context and world knowledge?

SOME RIGHTS RESERVED

Context is Personal and Keeps Changing

- **Example 3.73.** Consider the following sentences involving definite description:
  1. "*The king of America is rich.*" (true or false?)
  2. "*The king of America isn't rich.*" (false or true?)
  3. "*If America had a king, the king of America would be rich.*" (true or false!)
  4. "*The king of Buganda is rich.*" (Where is Buganda?)
  5. "*. . . Joe Smith. . . The CEO of Westinghouse announced budget cuts.*" (CEO=J.S.!)

  How do the interact with your context and world knowledge?

SOME RIGHTS RESERVED

Context is Personal and Keeps Changing

- **Example 3.74.** Consider the following sentences involving definite description:
  1. "*The king of America is rich.*" (true or false?)
  2. "*The king of America isn't rich.*" (false or true?)
  3. "*If America had a king, the king of America would be rich.*" (true or false!)
  4. "*The king of Buganda is rich.*" (Where is Buganda?)
  5. "*. . . Joe Smith. . . The CEO of Westinghouse announced budget cuts.*" (CEO=J.S.!)

  How do the interact with your context and world knowledge?
- The interpretation or whether they make sense at all dep
- **Note:** Last two examples feed back into the context or even world knowledge:
  - If 4. is uttered by an Africa expert, we add ""*Buganda exists and is a monarchy*" to our world knowledge
  - We add "*Joe Smith is the CEO of Westinghouse to the context/world knowledge*" (happens all the time in newpaper articles)

SOME RIGHTS RESERVED

30.4 Language Models

SOME RIGHTS RESERVED

Natural Languages vs. Formal Language

- **Recap:** A formal language is a set of strings.
- **Example 4.1.** Programming languages like `Java` or `C`$^{++}$ are formal languages.
- *Remark 4.2.* Natural languages like English, German, or Spanish are not.
- **Example 4.3.** Let us look at concrete examples
  - "*Not to be invited is sad!*" (definitely English)
  - "*To not be invited is sad!*" (controversial)
- **Idea:** Let's be lenient, instead of a hard set, use a probability distribution.
- **Definition 4.4.** A (statistical) language model is a probability distribution over sequences of characters or words.
- **Idea:** Try to learn/derive language models from text corpora.
- **Definition 4.5.** A text corpus (or simply corpus; plural corpora) is a large and structured collection of natural language texts called documents.
- **Definition 4.6.** In corpus linguistics, corpora are used to do statistical analysis and hypothesis testing, checking occurrences or validating linguistic rules within a specific natural language.

SOME RIGHTS RESERVED

N-gram Character Models

- Written text is composed of characters letters, digits, punctuation, and spaces.
- **Idea:** Let's study language models for sequences of characters.
- As for Markov processes, we write $P(c_{1:N})$ for the probability of a character sequence $c_1 \ldots c_n$ of length N.
- **Definition 4.7.** We call an character sequence of length n an n gram (unigram, bigram, trigram for $n = 1, 2, 3$).
- **Definition 4.8.** An n gram model is a Markov process of order $n - 1$.
- *Remark 4.9.* For a trigram model, $P(c_i \mid c_{1:i-1}) = P(c_i \mid c_{(i-2)}, c_{(i-1)})$. Factoring with the chain rule and then using the Markov property, we obtain

$$P(c_{1:N}) = \prod_{i=1}^{N} P(c_i \mid c_{1:i-1}) = \prod_{i=1}^{N} P(c_i \mid c_{(i-2)}, c_{(i-1)})$$

- **Thus,** a trigram model for a language with 100 characters, $P(c_i | c_{i-2:i-1})$ has 1.000.000 entries. It can be estimated from a corpus with 10^7 characters.

SOME RIGHTS RESERVED

Applications of N-Gram Models of Character Sequences

- What can we do with N gram models?
- **Definition 4.10.** The problem of language identification is given a text, determine the natural language it is written in.
- *Remark 4.11.* Current technology can classify even short texts like "*Hello, world*", or "*Wie geht es Dir*" correctly with more than 99% accuracy.
- **One approach:** Build a trigram language model $P(c_i|c_{i-2:i-1},\ell)$ for each candidate language ℓ by counting trigrams in a ℓ-corpus.
  Apply Bayes' rule and the Markov property to get the most likely language:

$$\begin{aligned}
\ell^* &= \operatorname*{argmax}_{\ell}\,(P(\ell \mid c_{1:N})) \\
&= \operatorname*{argmax}_{\ell}\,(P(\ell)\cdot P(c_{1:N} \mid \ell)) \\
&= \operatorname*{argmax}_{\ell}\,(P(\ell)\cdot (\prod_{i=1}^{N} P(c_i \mid c_{i-2:i-1},\ell)))
\end{aligned}$$

The prior probability $P(\ell)$ can be estimated, it is not a critical factor, since the trigram language models are extremely sensitive.

SOME RIGHTS RESERVED

Other Applications of Character N-Gram Models

- Spelling correction is a direct application of a single-language language model: Estimate the probability of a word and all off-by-one variants.
- **Definition 4.12.** Genre classification means deciding whether a text is a news story, a legal document, a scientific article, etc.
- *Remark 4.13.* While many features help make this classification, counts of punctuation and other character n-gram features go a long way [**KesNunSch:adtg97**].
- **Definition 4.14.** Named entity recognition (NER) is the task of finding names of things in a document and deciding what class they belong to.
- **Example 4.15.** In "*Mr. Sopersteen was prescribed aciphex.*" NER should recognize that "*Mr. Sopersteen*" is the name of a person and "*aciphex*" is the name of a drug.
- *Remark 4.16.* Character-level language models are good for this task because they can associate the character sequence "*ex*" with a drug name and "*steen*" with a person name, and thereby identify words that they have never seen before.

SOME RIGHTS RESERVED

N-Grams over Word Sequences

- **Idea:** n gram models apply to word sequences as well.
- **Problems:** The method works identically, but
  1. There are many more words than characters. (100 vs. 10^5 in Englisch)
  2. And what is a word anyways? (space/punctuation-delimited substrings?)
  3. Data sparsity: we do not have enough data! For a language model for 10^5 words in English, we have 10^{15} trigrams.
  4. Most training corpora do not have all words.

SOME RIGHTS RESERVED

Word N-Grams: Out-of-Vocab Words

- **Definition 4.17.** Out of vocabulary (OOV) words are unknown words that appear in the test corpus but not training corpus.
- *Remark 4.18.* OOV words are usually content words such as names and locations which contain information crucial to the success of NLP tasks.
- **Idea:** Model OOV words by
  1. adding a new word token, e.g. <UNK> to the vocabulary,
  2. in the training corpus, replacing the respective first occurrence of a previously unknown word by <UNK>,
  3. counting *n* grams as usual, treating <UNK> as a regular word.

  This trick can be refined if we have a word classifier, then use a new token per class, e.g. <EMAIL> or <NUM>.

SOME RIGHTS RESERVED

What can Word N-Gram Models do?

- **Example 4.19 (Test *n*-grams).** Build unigram, bigram, and trigram language models over the words [**RusNor:AIMA03**], randomly sample sequences from the models.
  1. Unigram: "*logical are as are confusion a may right tries agent goal the was ...*"
  2. Bigram: "*systems are very similar computational approach would be represented ...*"
  3. Trigram: "*planning and scheduling are integrated the success of naive bayes model ...*"
- **Clearly** there are differences, how can we measure them to evaluate the models?

SOME RIGHTS RESERVED

What can Word N-Gram Models do?

- **Example 4.23 (Test *n*-grams).** Build unigram, bigram, and trigram language models over the words **[RusNor:AIMA03]**, randomly sample sequences from the models.
  1. Unigram: "*logical are as are confusion a may right tries agent goal the was . . .*"
  2. Bigram: "*systems are very similar computational approach would be represented . . .*"
  3. Trigram: "*planning and scheduling are integrated the success of naive bayes model . . .*"
- **Clearly** there are differences, how can we measure them to evaluate the models?
- **Definition 4.24.** The perplexity of a sequence $c_{1:N}$ is defined as

$$\mathrm{Perplexity}(c_{1:N}) := P(c_{1:N})^{-\left(\frac{1}{N}\right)}$$

- **Intuition:** The reciprocal of probability, normalized by sequence length.
- **Example 4.25.** For a language with n characters or words and a language model that predicts that all are equally likely, the perplexity of any sequence is n. If some characters or words are more likely than others, and the model reflects that, then the perplexity of correct sequences will be less than n.
- **Example 4.26.** In ???, the perplexity was 891 for the unigram model, 142 for the bigram model and 91 for the trigram model.

SOME RIGHTS RESERVED

30.5 Part of Speech Tagging

SOME RIGHTS RESERVED

Language Models and Generalization

- **Recall:** *n*-grams can predict that a word sequence like "*a black cat*" is more likely than "*cat black a*". (as trigram 1. appears 0.000014% in a corpus and 2. never)
- **Native Speakers However:** Will tell you that "*a black cat*" matches a familiar pattern: article-adjective-noun, while "*cat black a*" does not!
- **Example 5.1.** Consider "*the fulvous kitten*" a native speaker reasons that it
  - follows the determiner-adjective-noun pattern
  - "*fulvous*" ($\widehat{=}$ brownish yellow) ends in "*ous*" $\leadsto$ adjective

  So by generalization this is (probably) correct English.
- **Observation:** The order of syntactical categories of words plays a role in English!
- **Problem:** How can we compute them? (up next)

SOME RIGHTS RESERVED

Part-of-Speech Tagging

- **Definition 5.2.** Part-of-speech tagging (also POS tagging, POST, or grammatical tagging) is the process of marking up a word in corpus with tags (called POS tags) as corresponding to a particular part of speech (a category of words with similar syntactic properties) based on both its definition and its context.
- **Example 5.3.** A sentence tagged with POS tags from the Penn treebank: (see below)

From	the	start	,	it	took	a	person	with	great	qualities	to	succeed
IN	DT	NN	,	PRP	VBD	DT	NN	IN	JJ	NNS	TO	VB

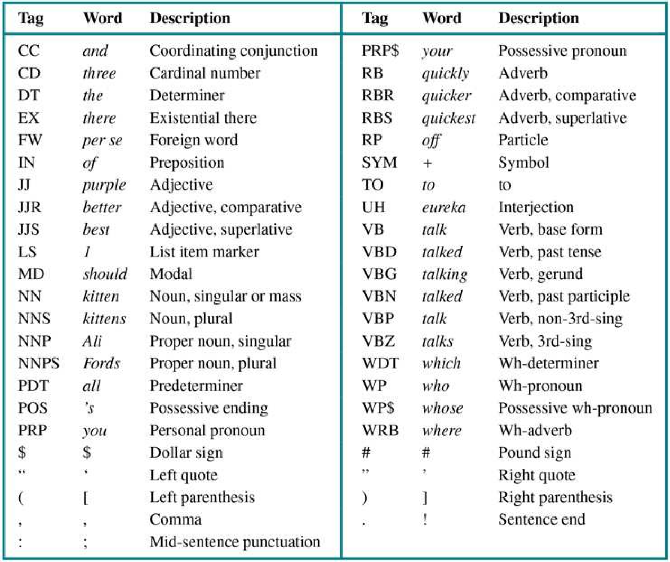
  1. "*From*" is tagged as a preposition (IN)
  2. "*the*" as a determiner (DT)
  3. . . .

- **Observation:** Even though POS tagging is uninteresting in its own right, it is useful as a first step in many NLP tasks.
- **Example 5.4.** In text-to-speech synthesis, a POS tag of "noun" for "*record*" helps determine the correct pronunciation (as opposed to the tag "verb")

SOME RIGHTS RESERVED

The Penn Treebank POS tags

- **Example 5.5.** The following 45 POS tags are used by the Penn treebank:



Tag	Word	Description	Tag	Word	Description
CC	*and*	Coordinating conjunction	PRP$	*your*	Possessive pronoun
CD	*three*	Cardinal number	RB	*quickly*	Adverb
DT	*the*	Determiner	RBR	*quicker*	Adverb, comparative
EX	*there*	Existential there	RBS	*quickest*	Adverb, superlative
FW	*per se*	Foreign word	RP	*off*	Particle
IN	*of*	Preposition	SYM	+	Symbol
JJ	*purple*	Adjective	TO	*to*	to
JJR	*better*	Adjective, comparative	UH	*eureka*	Interjection
JJS	*best*	Adjective, superlative	VB	*talk*	Verb, base form
LS	*1*	List item marker	VBD	*talked*	Verb, past tense
MD	*should*	Modal	VBG	*talking*	Verb, gerund
NN	*kitten*	Noun, singular or mass	VBN	*talked*	Verb, past participle
NNS	*kittens*	Noun, plural	VBP	*talk*	Verb, non-3rd-sing
NNP	*Ali*	Proper noun, singular	VBZ	*talks*	Verb, 3rd-sing
NNPS	*Fords*	Proper noun, plural	WDT	*which*	Wh-determiner
PDT	*all*	Predeterminer	WP	*who*	Wh-pronoun
POS	*'s*	Possessive ending	WP$	*whose*	Possessive wh-pronoun
PRP	*you*	Personal pronoun	WRB	*where*	Wh-adverb
$	$	Dollar sign	#	#	Pound sign
“	‘	Left quote	”	’	Right quote
(	[	Left parenthesis	)	]	Right parenthesis
,	,	Comma	.	!	Sentence end
:	;	Mid-sentence punctuation			

SOME RIGHTS RESERVED

Computing Part of Speech Tags

- **Idea:** Treat the POS tags in a sentence as state variables $C_{1:n}$ in a HMM: the words are the evidence variables $W_{1:n}$, use prediction for POS tagging.
- The HMM is a generative model that
  - starts in the tag predicted by the prior probability (usually IN) (problematic!)
  - and then, for each step makes two choices:
    - what word – e.g. "*From*" – should be emitted
    - what state – e.g. DT – should come next
- **This works, but** there are problems
  - the HMM does not consider context other than the current state (Markov property)
  - it does not have any idea what the sentence is trying to convey
- **Idea:** Use the Viterbi algorithm to find the most probable sequence of hidden states (POS tags)
- POS taggers based on the Viterbi algorithm can reach an F_1 score of up to 97%.

SOME RIGHTS RESERVED

The Viterbi algorithm for POS tagging – Details

- We need a transition model $P(\mathrm{C}_t \mid \mathrm{C}_{t-1})$: the probability of one POS tag following another.
- **Example 5.6.** $P(\mathrm{C}_t = VB \mid \mathrm{C}_{t-1} = MD) = 0.8$ means that given a modal verb (e.g. "*would*") the following word is a verb (e.g. "*think*") with probability 0.8.
- **Question:** Where does the number 0.8 come from?
- **Answer:** From counts in the corpus – with appropriate smoothing!
  There are 13124 instances of MD in the Penn treebank and 10471 are followed by a VB.
- For the sensor model $P(\mathrm{W}_t = would \mid \mathrm{C}_t = MD) = 0.1$ means that if we choose a modal verb, we will choose "*would*" 10% of the time.
- These numbers also come from the corpus with appropriate smoothing.
- **Limitations:** HMM models only know about the transition and sensor models
  In particular, we cannot take into account that e.g. words ending in "*ous*" are likely adjectives.
- We will see methods based on neural networks later.

SOME RIGHTS RESERVED

30.6 Text Classification

SOME RIGHTS RESERVED

Text Classification as a NLP Task

- **Problem:** Often we want to (ideally) automatically see who can best deal with a given document (e.g. e-mails in customer service)
- **Definition 6.1.** Given a set of categories the task of deciding which one a given document belongs to is called text classification or categorization.
- **Example 6.2.** Language identification and genre classification are examples of text classification.
- **Example 6.3.** Sentiment analysis – classifying a product review as positive or negative.
- **Example 6.4.** Spam detection – classifying an email message as spam or ham (i.e. non-spam).

SOME RIGHTS RESERVED

Spam Detection

- **Definition 6.5.** Spam detection – classifying an email message as spam or ham (i.e. non-spam)
- **General Idea:** Use NLP/machine learning techniques to learn the categories.
- **Example 6.6.** We have lots of examples of spam/ham, e.g.

Spam (from my spam folder)	Ham (in my inbox)
Wholesale Fashion Watches -57% today. Designer watches for cheap ...	The practical significance of hypertree width in identifying more ...
You can buy ViagraFr$1.85 All Medications at unbeatable prices! ...	Abstract: We will motivate the problem of social identity clustering: ...
WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US ...	Good to see you my friend. Hey Peter, It was good to hear from you. ...
Sta.rt earn*ing the salary yo,u d-eserve by o'btaining the prope,r crede'ntials!	PDS implies convexity of the resulting optimization problem (Kernel Ridge ...

- **Specifically:** What are good features to classify e-mails by?
  - n-grams like "*for cheap*" and "*You can buy*" indicate spam (but also occur in ham)
  - character-level features: capitalization, punctuation (e.g. in "*yo,u d-eserve*")
- **Note:** We have two complementary ways of talking about classification: (up next)
  - using language models
  - using machine learning

SOME RIGHTS RESERVED

Spam Detection as Language Modeling

- **Idea:** Define two n-gram language models:
  1. one for $\mathrm{P}(\mathrm{Message}|\mathrm{spam})$ by training on the spam folder
  2. one for $\mathrm{P}(\mathrm{Message}|\mathrm{ham})$ by training on the inbox

  Then we can classify a new message m with an application of Bayes' rule:

$$\underset{c\in\{\mathrm{spam},\mathrm{ham}\}}{\mathrm{argmax}}(P(c \mid m)) = \underset{c\in\{\mathrm{spam},\mathrm{ham}\}}{\mathrm{argmax}}(P(m \mid c)P(c))$$

  where $P(c)$ is estimated just by counting the total number of spam and ham messages.
- This approach works well for spam detection, just as it did for language identification.

SOME RIGHTS RESERVED

Classifier Success Measures: Precision, Recall, and F_1 score

- We need a way to measure success in classification tasks.
- **Definition 6.7.** Let $f_C\colon S \to \mathbb{B}$ be a binary classifier for a class $C \subseteq S$, then we call $a \in S$ with $f_C(a) = \mathsf{T}$ a false positive, iff $a \notin C$ and $f_C(a) = \mathsf{F}$ a false negative, iff $a \in C$. False positives and negatives are erros of f_C. True positives and negatives occur when f_C correctly indicates actual membership in S.
- **Definition 6.8.** The precision of f_C is defined as $\frac{\#(TP)}{\#(TP)+\#(FP)}$ and the recall is $\frac{\#(TP)}{\#(TP)+\#(FN)}$, where TP is the set of true positives and $FN/FP$the sets of false negatives and false positives of f_C.
- **Intuitively** these measure the rates of:
  - true positives in class C. (precision high, iff few false positives)
  - true positives in ${f_C}^{-1}(\mathsf{T})$. (recall high, iff few true positives forgotten, i.e. few false negatives)
- **Definition 6.9.** The F_1 score combines precision and recall into a single number: (harmonic mean)

$$2\frac{\text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})}$$

- **Observation:** Classifiers try to reach precision and recall $\leadsto$ F_1 score of 1.
  - if that is impossible, compromize on one $\leadsto$ F_β score . (application-dependent)
  - The F_β score generalizes the F_1 score by weighing the precision β times as important as recall.

SOME RIGHTS RESERVED

30.7 Information Retrieval

SOME RIGHTS RESERVED

Information Retrieval

-
- **Definition 7.1.** An information need is an individual or group's desire to locate and obtain information to satisfy a conscious or unconscious need.
- **Definition 7.2.** An information object is medium that is mainly used for its information content.
- **Definition 7.3.** Information retrieval (IR) deals with the representation, organization, storage, and maintenance of information objects that provide users with easy access to the relevant information and satisfy their various information needs.

  Observation (Hjørland 1997): Information need is closely related to relevance: If something is relevant for a person in relation to a given task, we might say that the person needs the information for that task.
- **Definition 7.4.** Relevance denotes how well an information object meets the information need of the user. Relevance may include concerns such as timeliness, authority or novelty of the object.
- **Observation:** We normally come in contact with IR in the form of web search.
- **Definition 7.5.** Web search is a fully automatic process that responds to a user query by returning a sorted document list relevant to the user requirements expressed in the query.
- **Example 7.6.** Google and Bing are web search engines, their query is a

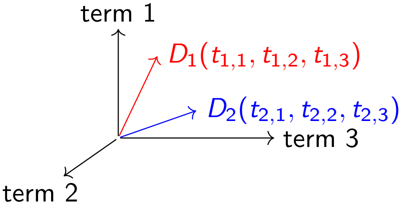
Vector Space Models for IR

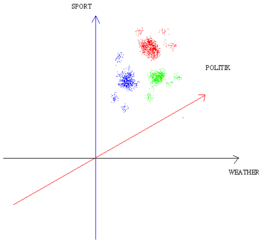
- **Idea:** For web search, we usually represent documents and queries as bags of words over a fixed vocabulary V. Given a query Q, we return all documents that are "similar".
- **Definition 7.7.** Given a vocabulary (a list) V of words, a word $w \in V$, and a document d, then we define the raw term frequency (often just called the term frequency) of w in d as the number of occurrences of w in d.
- **Definition 7.8.** A multiset of words in $V = \{t_1, \ldots, t_n\}$ is called a bag of words (BOW), and can be represented as a word frequency vectors in $\mathbb{N}^{|V|}$: the vector of raw word frequencies.
- **Example 7.9.** If we have two documents: $d_1 =$ "*Have a good day!*" and $d_2 =$ "*Have a great day!*", then we can use $V =$ "*Have*", "*a*", "*good*", "*great*", "*day*" and can represent "*good*" as $\langle 0, 0, 1, 0, 0\rangle$, "*great*" as $\langle 0, 0, 0, 1, 0\rangle$, and d_1 a $\langle 1, 1, 1, 0, 1\rangle$.
  Words outside the vocabulary are ignored in the BOW approach. So the document $d_3 =$ "*What a day, a good day*" is represented as $\langle 0, 2, 1, 0, 2\rangle$.

SOME RIGHTS RESERVED

Vector Space Models for IR

- **Idea:** Query and document are similar, iff the angle between their word frequency vectors is small.


term 1
$D_1(t_{1,1}, t_{1,2}, t_{1,3})$
$D_2(t_{2,1}, t_{2,2}, t_{2,3})$
term 3
term 2



- **Lemma 7.10 (Euclidean Dot Product Formula).** $A \cdot B = \|A\|_2 \, \|B\|_2 \cos\theta$, *where* θ *is the angle between* A *and* B*.*
- **Definition 7.11.** The cosine similarity of A and B is $\cos\theta = \frac{A \cdot B}{\|A\|_2 \|B\|_2}$.

SOME RIGHTS RESERVED

TF-IDF: Term Frequency/Inverse Document Frequency

- **Problem:** Word frequency vectors treat all the words equally.
- **Example 7.12.** In an query "*the brown cow*", the "*the*" is less important than "*brown cow*". (because "*the*" is less specific)
- **Idea:** Introduce a weighting factor for the word frequency vector that de-emphasizes the dimension of the more (globally) frequent words.
- We need to normalize the word frequency vectors first:
- **Definition 7.13.** Given a document d and a vocabulary word $t \in V$, the normalized term frequency (confusingly often called just term frequency) $\mathrm{tf}(t,d)$ is the raw term frequency divided by $|d|$.
- **Definition 7.14.** Given a document collection $D = \{d_1, \ldots, d_N\}$ and a word t the inverse document frequency is given by $\mathrm{idf}(t,D) := \log_{10}(\frac{N}{|\{d \in D \mid t \in d\}|})$.
- **Definition 7.15.** We define $\mathrm{tfidf}(t,d,D) := \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$.
- **Idea:** Use the tfidf-vector with cosine similarity for information retrieval instead.
- **Definition 7.16.** Let D be a document collection with vocabulary $V = \{t_1, \ldots, t_{|V|}\}$, then the tfidf-vector $\overline{\mathrm{tfidf}}(d,D) \in \mathbb{N}^{|V|}$ is defined by $\overline{\mathrm{tfidf}}(d,D)_i := \mathrm{tfidf}(t_i,d,D)$.

SOME RIGHTS RESERVED

TF-IDF Example

- Let $D := \{d_1, d_2\}$ be a document corpus over the vocabulary

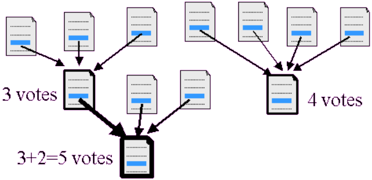
$$V = \{\text{“}\textit{this}\text{''}, \text{“}\textit{is}\text{''}, \text{“}\textit{a}\text{''}, \text{“}\textit{sample}\text{''}, \text{“}\textit{another}\text{''}, \text{“}\textit{example}\text{''}\}$$

  with word frequency vectors $\langle 1, 1, 1, 2, 0, 0\rangle$ and $\langle 1, 1, 0, 0, 2, 3\rangle$.
- Then we compute for the word “*this*”
  - $\mathrm{tf}(\text{“}\textit{this}\text{''}, d_1) = \frac{1}{5} = 0.2$ and $\mathrm{tf}(\text{“}\textit{this}\text{''}, d_2) = \frac{1}{7} \approxeq 0.14$,
  - idf is constant over D, we have $\mathrm{idf}(\text{“}\textit{this}\text{''}, D) = \log_{10}(\frac{2}{2}) = 0$,
  - thus $\mathrm{tfidf}(\text{“}\textit{this}\text{''}, d_1, D) = 0 = \mathrm{tfidf}(\text{“}\textit{this}\text{''}, d_2, D)$. (“*this*” occurs in both)
- The word “*example*” is more interesting, since it occurs only in d_2 (thrice)
  - $\mathrm{tf}(\text{“}\textit{example}\text{''}, d_1) = \frac{0}{5} = 0$ and $\mathrm{tf}(\text{“}\textit{example}\text{''}, d_2) = \frac{3}{7} \approxeq 0.429$.
  - $\mathrm{idf}(\text{“}\textit{example}\text{''}, D) = \log_{10}(\frac{2}{1}) \approxeq 0.301$,
  - thus $\mathrm{tfidf}(\text{“}\textit{example}\text{''}, d_1, D) = 0 \cdot 0.301 = 0$ and $\mathrm{tfidf}(\text{“}\textit{example}\text{''}, d_2, D) \approxeq 0.429 \cdot 0.301 = 0.129$.

SOME RIGHTS RESERVED

Ranking Search Hits: e.g. Google's Page Rank

- **Problem:** There are many hits, need to sort them (e.g. by importance)
- **Idea:** A web site is important, …if many other hyperlink to it.



- **Refinement:** …, if many important web pages hyperlink to it.
- **Definition 7.17.** Let A be a web page that is hyperlinked from web pages $S_1, \ldots, S_n$, then the page rank PR of A is defined as

$$\mathrm{PR}(A) = 1 - d + d\left(\frac{\mathrm{PR}(S_1)}{C(S_1)} + \cdots + \frac{\mathrm{PR}(S_n)}{C(S_n)}\right)$$

where $C(W)$ is the number of links in a page W and $d = 0.85$.
- *Remark 7.18.* $\mathrm{PR}(A)$ is the probability of reaching A by random browsing.

SOME RIGHTS RESERVED

30.8 Information Extraction

Information Extraction

- **Definition 8.1.** Information extraction is the process of acquiring information by skimming a text and looking for occurrences of a particular class of object and for relationships among objects.
- **Example 8.2.** Extracting instances of addresses from web pages, with attributes for street, city, state, and zip code;
- **Example 8.3.** Extracting instances of storms from weather reports, with attributes for temperature, wind speed, and precipitation.
- **Observation:** In a limited domain, this can be done with high accuracy.

SOME RIGHTS RESERVED

Attribute-Based Information Extraction

- **Definition 8.4.** In attribute-based information extraction we assume that the text refers to a single object and the task is to extract a factored representation.
- **Example 8.5 (Computer Prices).** Extracting from the text "*IBM ThinkBook 970. Our price: $399.00*" the attribute-based representation {Manufacturer=IBM, Model=ThinkBook970,Price=$399.00}.
- **Idea:** Try a template-based approach for each attribute.
- **Definition 8.6.** A template is a finite automaton that recognizes the information to be extracted. The template often consists of three sub-automata per attribute: the prefix pattern followed by the target pattern (it matches the attribute value) and the postfix pattern.
- **Example 8.7 (Extracing Prices with Regular Expressions).** When we want to extract computer price information, we could use regular expressions for the automata, concretely, the
  - prefix pattern: .∗price[:]?
  - target pattern: [$][0−9]+([.][0−9][0−9])?
  - postfix pattern: + shipping|
- **Alternative:** take all the target matches and choose among them.
- **Example 8.8.** For "*List price $99.00, special sale price $78.00, shipping $3.00.*" take the lowest price that is within 50% of the highest price. ⇝ "*$78.00*"

SOME RIGHTS RESERVED

Relational Information Extraction

- **Question:** Can we also do structured representations?
- **Answer:** That is the next step up from attribute-based information extraction.
- **Definition 8.9.** The task of a relational extraction system is to extract multiple objects and the relationships among them from a text.
- **Example 8.10.** When these systems see the text "*$249.99*," they need to determine not just that it is a price, but also which object has that price.
- **Example 8.11.** FASTUS is a typical relational extraction system, which handles news stories about corporate mergers and acquisitions. It can read the story

  > *Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.*

  and extract the relations:

$$e \in \mathit{JointVentures} \wedge \mathit{Product}(e, ''\mathit{golfclubs}) \wedge \mathit{Date}(e, ''\mathit{Friday}'')$$

$$\mathit{Member}(e, ''\mathit{BridgestoneSportsCo}'') \wedge \mathit{Member}(e, ''\mathit{alocalconcern}'')$$

$$\mathit{Member}(e, ''\mathit{aJapanesetradinghouse}'')$$

SOME RIGHTS RESERVED

Advertisement: Logic-Based Natural Language Semantics

- **Advanced Course:** "Logic-Based Natural Language Semantics" (next semester)
  - Wed. 10:15-11:50 and Thu 12:15-13:50 (expected: $\leq$ 10 Students)
- **Contents:** (Alternating Lectures and hands-on Lab Sessions)
  - Foundations of Natural Language Semantics (NLS)
  - Montague's Method of Fragments (Grammar, Semantics Constr., Logic)
  - Implementing Fragments in GLF (Grammatical Framework and MMT)
  - Inference Systems for Natural Language Pragmatics (tableau machine)
  - Advanced logical systems for NLS (modal, higher-order, dynamic Logics)
- **Grading:** Attendance & Wakefulness, Project/Homework, Oral Exam.
- **Course Intent:** Groom students for bachelor/master theses and as KWARC research assistants.

SOME RIGHTS RESERVED

Chapter 31
Deep Learning for NLP

SOME RIGHTS RESERVED

Deep Learning for NLP: Agenda

- **Observation:** Symbolic and statistical systems have demonstrated success on many NLP tasks, but their performance is limited by the endless complexity of natural language.
- **Idea:** Given the vast amount of text in machine-readable form, can data-driven machine-learning base approaches do better?
- In this chapter, we explore this idea, using – and extending – the methods from ???.
- Overview:
  1. Word embeddings
  2. Recurrent neural networks for NLP
  3. Sequence-to-sequence models
  4. Transformer Architecture
  5. Pretraining and transfer learning.

SOME RIGHTS RESERVED

31.1 Word Embeddings

Word Embeddings

- **Problem:** For ML methods in NLP, we need numerical data. (not words)
- **Idea:** Embed words or word sequences into real vector spaces.
- **Definition 1.1.** A word embedding is a mapping from words in context into a real vector space $\mathbb{R}^n$ used for natural language processing.
- **Definition 1.2.** A vector is called one hot, iff all components are 0 except for one 1. We call a word embedding one hot, iff all of its vectors are.
  One hot word embeddings are rarely used for actual tasks, but often used as a *starting point* for better word embeddings.
- **Example 1.3 (Vector Space Methods in Information Retrieval).**
  Word frequency vectors are induced by adding up one hot word embeddings.
- **Example 1.4.** Given a corpus D – the context – the tf idf word embedding is given by $\mathrm{tfidf}(t,d,D){:=}\mathrm{tf}(t,d)\cdot\log_{10}(\frac{|D|}{|\{d\in D\,|\,t\in d\}|})$, where $\mathrm{tf}(t,d)$ is the term frequency of word t in document d.
- **Intuition behind these two:** Words that occur in similar documents are similar.

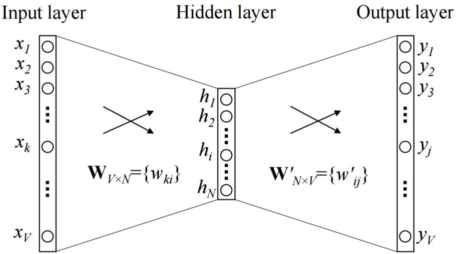
SOME RIGHTS RESERVED

Word2Vec

Idea: Use *feature extraction* to map words to vectors in $\mathbb{R}^N$:
Train a neural network on a "dummy task", throw away the output layer, use the previous layer's output (of size N) as the word embedding

First Attempt: Dimensionality Reduction: Train to predict the original one hot vector:

- For a vocabulary size V, train a network with a single hidden layer; i.e. three layers of sizes (V, N, V). The first two layers will compute our embeddings.
- Feed the one hot encoded input word into the network, and train it on the one hot vector itself, using a softmax activation function at the output layer. (softmax normalizes a vector into a probability distribution)



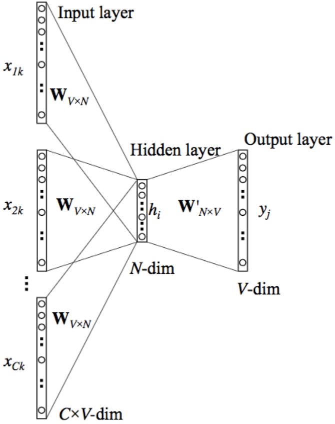
SOME RIGHTS RESERVED

Word2Vec: The Continuous Bag Of Words (CBOW) Algorithm

Distributional Semantics: "a word is characterized by the company it keeps".

Better Idea: Predict a word from its context:

- For a context window size n, take all sequences of $2n+1$ words in our corpus (e.g. *the brown cow jumps over the moon* for $n=3$) as training data. We call the word at the center (*jumps*) the *target word*, and the remaining words the *context words*.
- For every such sentence, pass all context words (one-hot encoded) through the first layer of the network, yielding $2n$ vectors.
- Pass their average into the output layer (*average pooling layer*) with a softmax activation function, and train network to predict the target



SOME RIGHTS RESERVED

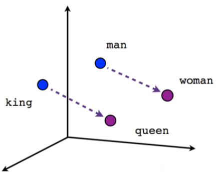
Properties

Vector embeddings like CBOW have interesting properties:

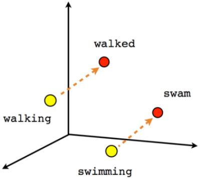
- *Similarity*: Using e.g. *cosine similarity* $(A \cdot B \cdot \cos(\theta))$ to compare vectors, we can find words with similar meanings.
- Semantic and syntactic relationships emerge as arithmetic relations:

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

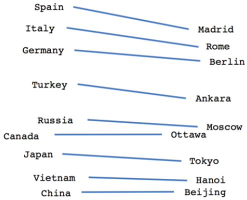
$$\text{germany} - \text{country} + \text{capitol} \approx \text{berlin}$$



Male-Female



Verb tense



Country-Capital

SOME RIGHTS RESERVED

Common Word Embeddings

- **Observation:** Word embeddings are crucial as first steps in any NN-based NLP methods.
- In practice it is often sufficient to use generic, pretrained word embeddings
- **Definition 1.5.** Common pretrained – i.e. trained for generic NLP applications word embeddings include
  - Word2vec: the original system that established the concept (see above)
  - GloVe (Global Vectors)
  - fastText (embeddings for 157 languages)
- But we can also train our own word embedding (together with main task) (up next)

SOME RIGHTS RESERVED

Learning POS tags and Word embeddings simultaneously

Specific word embeddings are trained on a carefully selected corpus and tend to emphasize the characteristics of the task.

Example 1.6. POS tagging – even though simple – is a good but non-trivial example.

Recall that many words can have multiple POS tags, e.g. “*cut*” can be

- a present tense verb (transitive or intransitive)
- a past tense verb
- a infinitive verb
- a past participle
- an adjective
- a noun.

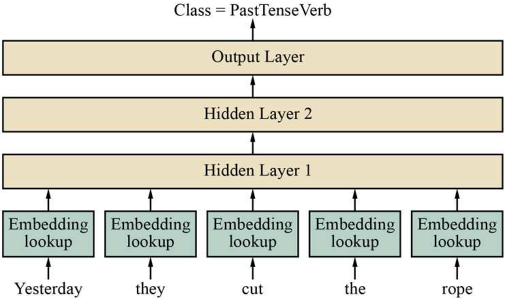
If a nearby temporal adverb refers to the past ⇝ this occurrence may be a past tense verb.

Note: CBOW treats all context words identically reagrdless of *order*, but in POS tagging the exact *positions* of the words matter.

SOME RIGHTS RESERVED

POS/Embedding Network

Idea: Start with a random (or pretrained) embedding of the words in the corpus and just concatenate them over some context window size



- Layer 1 has (in this case) $5 \cdot N$ inputs, Output layer is one hot over POS classes.
- The embedding layers treat all words the same, but the first hidden layer will treat them differently depending on the position.
- The embeddings will be finetuned for the POS task during training.

Note: Better *positional encoding* techniques exist (e.g. sinusoidal), but for fixed small context window sizes, this works well.

SOME RIGHTS RESERVED

31.2 Recurrent Neural Networks

SOME RIGHTS RESERVED

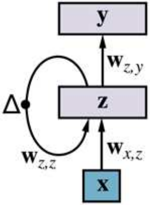
Recurrent Neural Networks in NLP

- word embeddings give a good representation of words in isolation.
- But natural language of word sequences ⇜ surrounding words provide context!
- For simple tasks like POS tagging, a fixed-size window of e.g. 5 words is sufficient.
- **Observation:** For advanced tasks like question answering we need more context!
- **Example 2.1.** In the sentence "*Eduardo told me that Miguel was very sick so I took him to the hospital*", the pronouns "*him*" refers to "*Miguel*" and not "*Eduardo*". (14 words of context)
- **Observation:** Language models with *n*-grams or *n*-word feed-forward networks have problems:
  Either the context is too small or the model has too many parameters! (or both)
- **Observation:** Feed-forward networks N also have the problem of asymmetry: whatever N learns about a word w at position n, it has to relearn about w at position $m \neq n$.
- **Idea:** What about recurrent neural networks – nets with cycles? (up next)

SOME RIGHTS RESERVED

RNNs for Time Series

- **Idea:** RNNs – neural networks with cycles – have memory
  ⇝ use that for more context in neural NLP.
- **Example 2.2 (A simple RNN).**
  It has an input layer x, a hidden layer z with recurrent connections and delay Δ, and an output layer y as shown on the right.
  Defining Equations for time step t:



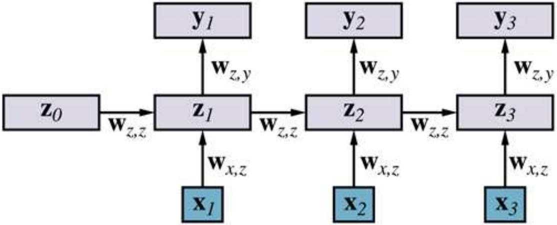
$$\begin{aligned} z_t &= g_z(W_{z,z}z_{t-1} + W_{x,z}x_t) \\ y_t &= g_y(W_{z,y}z_t) \end{aligned}$$

  where g_z and g_y are the activation functions for the hidden and output layers.
- **Intuition:** RNNs are a bit like HMMs and dynamic Bayesian Networks: They make a Markov assumption: the hidden state z suffices to capture the input from all previous inputs.
- **Side Benefit:** RNNs solve the asymmetry problem ⇜, the $W_{z,z}$ are the same at every step.

SOME RIGHTS RESERVED

Training RNNs for NLP

- **Idea:** For training, unroll a RNN into a feed-forward network ⇝ back-propagation.
- **Example 2.3.** The RNN from **??** unrolled three times.

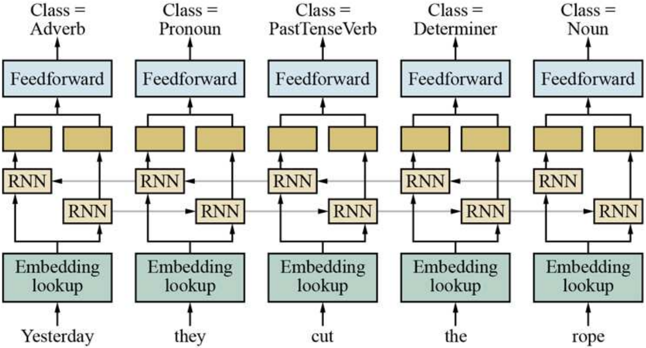


Problem: The weight matrices $\mathsf{W}_{\mathsf{x,z}}$, $\mathsf{W}_{\mathsf{z,z}}$, and $\mathsf{W}_{\mathsf{z,y}}$ are shared over all time slides.

- **Definition 2.4.** The back-propagation through time algorithm carefully maintains the identity of $\mathsf{W}_{\mathsf{z,z}}$ over all steps

Bidirectional RNN for more Context

- **Observation:** RNNs only take left context – i.e. words before – into account, but we may also need right context – the words after.
- **Example 2.5.** For "*Eduardo told me that Miguel was very sick so I took him to the hospital*" the pronoun "*him*" resolves to "*Miguel*" with high probability. If the sentence ended with "*to see Miguel*", then it should be "*Eduardo*".
- **Definition 2.6.** A bidirectional RNN concatenates a separate right-to-left model onto a left-to-right model
- **Example 2.7.** Bidirectional RNNs can be used for POS tagging, extending the network from ???



SOME RIGHTS RESERVED

Long Short-Term Memory RNNs

- **Problem:** When training a vanilla RNN using back-propagation through time, the long-term gradients which are back-propagated can "vanish" – tend to zero – or "explode" – tend to infinity.
- **Definition 2.8.** LSTMs provide a short-term memory for RNN that can last thousands of time steps, thus the name "long short-term memory". A LSTM can learn when to remember and when to forget pertinent information,
- **Example 2.9.** In NLP LSTMs can learn grammatical dependencies.
  An LSTM might process the sentence "*Dave, as a result of his controversial claims, is now a pariah*" by
  - remembering the (statistically likely) grammatical gender and number of the subject "*Dave*",
  - note that this information is pertinent for the pronoun "*his*" and
  - note that this information is no longer important after the verb "*is*".

SOME RIGHTS RESERVED

LSTM: Idea

Introduce a *memory vector* c in addition to the recurrent (short-term memory) vector z

- c is essentially copied from the previous time step, but can be modified by the *forget gate* f, the *input gate* i, and the *output gate* o.
- the *forget gate* f decides which components of c to retain or discard
- the *input gate* i decides which components of the *current* input to *add* to c (additive, not multiplicative $\leadsto$ no vanishing gradients)
- the *output gate* o decides which components of c to *output* as z

SOME RIGHTS RESERVED

31.3 Sequence-to-Sequence Models

SOME RIGHTS RESERVED

Neural Machine Translation

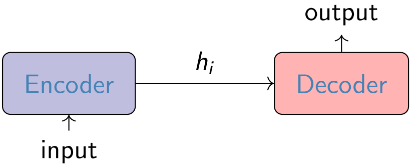
- **Question:** Machine translation (MT) is an important task in NLP, can we do it with neural networks?
- **Observation:** If there were a one-to-one correspondence between source words and target words MT would be a simple tagging task. But
  - the three Spanish words "*caballo de mar*" translate to the English "*seahorse*" and
  - the two Spanish words "*perro grande*" translate to English as "*big dog*".
  - in English, the subject is usually first and in Fijian last.
- **Idea:** For MT, generate one word at a time, but keep track of the context, so that
  - we can remember parts of the source we have not translated yet
  - we remember what we already translated so we do not repeat ourselves.

  We may have to process the whole source sentence before generating the target!
- **Remark:** This smells like we need LSTMs.

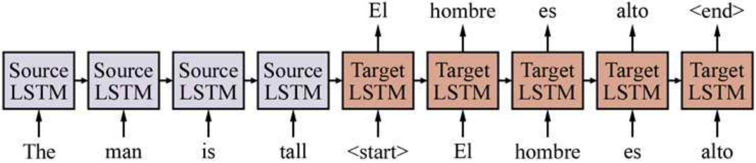
SOME RIGHTS RESERVED

Sequence-To-Sequence Models

- **Idea:** Use two coupled RNNs, one for the source, and one for the target. The input for the target is the output of the last hidden layer of the source RNN.
- **Definition 3.1.** A sequence-to-sequence (seq2seq) model is a neural model for translating an input sequence x into an output sequence y by an encoder followed by a decoder generates y.



- **Example 3.2.** A simple seq2seq model (without embedding and output layers)



Each block represents one LSTM time step; inputs are fed successively followed by the token `<start>` to start the decoder.

SOME RIGHTS RESERVED

Seq2Seq Evaluation

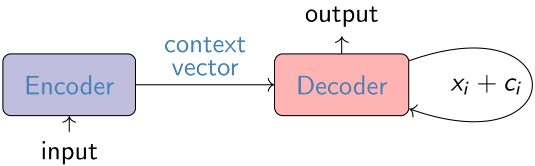
- **Remark:** Seq2seq models were a major breakthrough in NLP and MT. But they have three major shortcomings:
  - **nearby context bias**: RNNs remember with their hidden state, which has more information about a word in – say – step 56 than in step 5. BUT long-distance context can also be important.
  - **fixed context size**: the entire information about the source sentence must be compressed into the fixed-dimensional – typically 1024 – vector. Larger vectors ⇝ slow training and overfitting.
- **Idea:** Concatenate all source RNN hidden vectors to use all of them to mitigate the nearby context bias.
- **Problem:** Huge increase of weights ⇝ slow training and overfitting.

SOME RIGHTS RESERVED

Attention

- **Bad Idea:** Concatenate all source RNN hidden vectors to use all of them to mitigate the nearby context bias.
- **Better Idea:** The decoder generates the target sequence one word at a time.
  ⇝ Only a small part of the source is actually relevant.
  the decoder must focus on different parts of the source for every word.
- **Idea:** We need a neural component that does context-free summarization.
- **Definition 3.3.** An attentional seq2seq model is a seq2seq that passes along a context vector c_i in the decoder. If $h_i = RNN(h_{i-1}, x_i)$ is the standard decoder, then the decoder with attention is given by $h_i = RNN(h_{i-1}, x_i + c_i)$, where $x_i + c_i$ is the concatenation of the input x_i and context vectors c_i with

$$\begin{array}{lcll} r_{ij} & = & h_{i-1} \cdot s_j & \text{raw attention score} \\ a_{ij} & = & e^{r_{ij}} / (\sum_k e^{r_{ij}}) & \text{attention probability matrix} \\ c_i & = & \sum_j a_{ij} \cdot s_j & \text{context vector} \end{array}$$



output

context vector

Encoder

Decoder

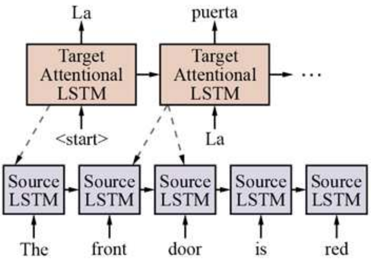
$x_i + c_i$

input

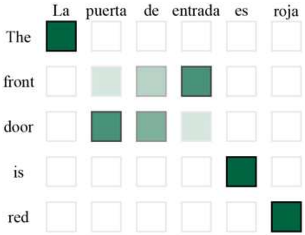
SOME RIGHTS RESERVED

Attention: English to Spanish Translation

- **Example 3.4.** An attentional seq2seq model for English-to-Spanish translation



dashed lines represent attention



attention probablity matrix
darker colors ⇝ higher probabilities

- **Remarks:** The attention
  - component learns no weights and supports variable-length sequences.
  - is entirely latent – the developer does not influence it.

Attention: Greedy Decoding

- During training, a seq2seq model tries to maximize the probability of each word in the training sequence, conditioned on the source and the previous target words.
- **Definition 3.5.** The procedure that generates the target one word at a time and feeds it back at the next time step is called decoding.
- **Definition 3.6.** Always selecting the highest probability word is called greedy decoding.
- **Problem:** This may not always maximize the probability of the whole sequence
- **Example 3.7.** Let's use a greedy decoder on "*The front door is red*".
  - The correct translation is "*La puerta de entrada es roja*".
  - Suppose we have generated the first word "*La*" for "*The*".
  - A greedy decoder might propose "*entrada*" for "*front*".
- Greedy decoding is fast, but has no mechanism for correcting mistakes.
- **Solution:** Use an optimizing search algorithm (e.g. local beam search)

SOME RIGHTS RESERVED

Decoding with Beam Search

- **Recall:** Greedy decoding is not optimal!
- **Idea:** Search for an optimal decoding (or at least a good one) using one of the search algorithms from .
- Local beam search is a common choice in machine translation. Concretely:
  - keep the top k hypotheses at each stage,
  - extending each by one word using the top k choices of words,
  - then chooses the best k of the resulting k^2 new hypotheses.

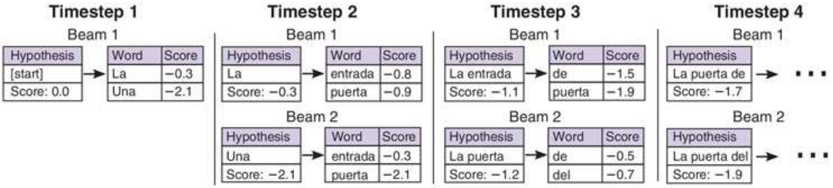
  When all hypotheses in the beam generate the special <end> token, the algorithm outputs the highest scoring hypothesis.
- **Observation:** The better the seq2seq models get, the smaller we can keep beam size

  Today beams of $b = 4$ are sufficient after $b = 100$ a decade ago.

SOME RIGHTS RESERVED

Decoding with Beam Search

- **Example 3.8.** A local beam search with beam size $b = 2$



Timestep 1

Beam 1

Hypothesis
[start]
Score: 0.0

Word	Score
La	−0.3
Una	−2.1

Timestep 2

Beam 1

Hypothesis
La
Score: −0.3

Word	Score
entrada	−0.8
puerta	−0.9

Beam 2

Hypothesis
Una
Score: −2.1

Word	Score
entrada	−0.3
puerta	−2.1

Timestep 3

Beam 1

Hypothesis
La entrada
Score: −1.1

Word	Score
de	−1.5
puerta	−1.9

Beam 2

Hypothesis
La puerta
Score: −1.2

Word	Score
de	−0.5
del	−0.7

Timestep 4

Beam 1

Hypothesis
La puerta de
Score: −1.7

Beam 2

Hypothesis
La puerta del
Score: −1.9

  - Word scores are log-probabilities generated by the decoder softmax
  - hypothesis score is the sum of the word scores.

  At time step 3, the highest scoring hypothesis "*La entrada*" can only generate low-probability continuations, so it "falls off the beam". (as intended)

SOME RIGHTS RESERVED

31.4 The Transformer Architecture

SOME RIGHTS RESERVED

Self-Attention

- **Idea:** "Attention is all you need!" (see [**VasShaPar:aiayn17**])
- So far, attention was used from the encoder to the decoder.
- Self-attention extends this so that each hidden states sequence also attends to itself. (*coder to *coder)
- **Idea:** Just use the dot product of the input vectors
- **Problem:** Always high, so each hidden state will be biased towards attending to itself.
- Self-attention solves this by first projecting the input into three different representations using three different weight matrices:
  - the query vector $\mathrm{q}_i = \mathrm{W}_q \mathrm{x}_i \mathrel{\widehat{=}}$ standard attention
  - key vector $\mathrm{k}_i = \mathrm{W}_k \mathrm{x}_i \mathrel{\widehat{=}}$ the source in seq2seq
  - value vector $\mathrm{v}_i = \mathrm{W}_v \mathrm{x}_i$ is the context being generated

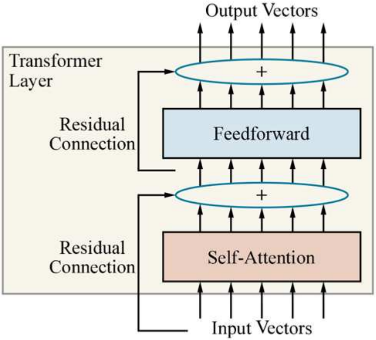
$$\begin{aligned} r_{ij} &= (\mathrm{q}_i \cdot \mathrm{k}_i)/\sqrt{d} \\ a_{ij} &= e^{r_{ij}} / (\textstyle\sum_k e^{r_{ij}}) \\ c_i &= \textstyle\sum_j a_{ij} \cdot \mathrm{v}_j \end{aligned}$$

where d is the dimension of k and q.

The Transformer Architecture

- **Definition 4.1.** The transformer architecture uses neural blocks called transformers, which are built up from multiple transformer layers.
- **Remark:** The context modeled in self-attention is agnostic to word order ⇝ transformers use positional embeddings to cope with that.
- **Example 4.2.**

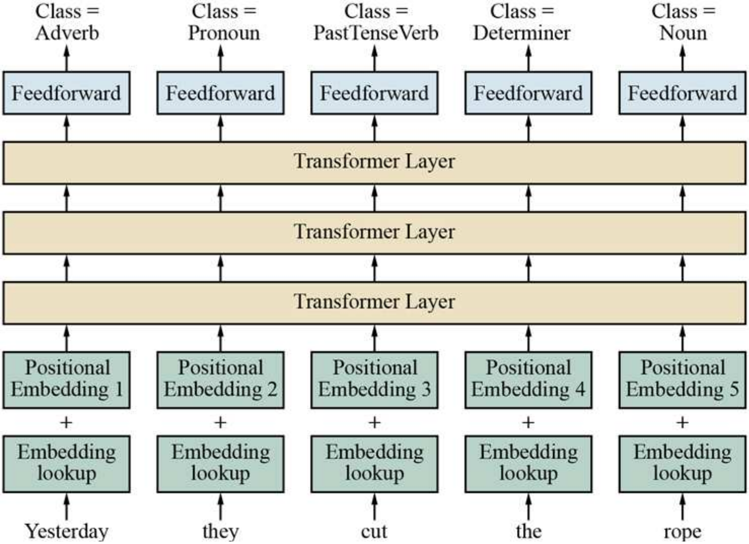
A single-layer transformer consists of self-attention, a feed-forward network, and residual connections to cope with the vanishing gradient problem.



- In practice transformers consist of 6-7 transformer layers.

A Transformer for POS tagging

- **Example 4.3.** A transformers for POS tagging:



SOME RIGHTS RESERVED

31.5 Large Language Models

SOME RIGHTS RESERVED

Pretraining and Transfer Learning

- Getting enough data to build a robust model can be a challenge.
- In NLP we often work with unlabeled data
  - syntactic/semantic labeling is much more difficult ⇝ costly than image labeling.
  - the Internet has lots of texts (adds $\sim 10^{11}$ words/day)
- **Idea:** Why not let other's do this work and re-use their training efforts.
- **Definition 5.1.** In pretraining we use
  - a large amount of shared general-domain language data to train an initial version of an NLP model.
  - a smaller amount of domain-specific data (perhaps labeled) to finetune it to the vocabulary, idioms, syntactic structures, and other linguistic phenomena that are specific to the new domain.
- Pretraining is a form of transfer learning:
- **Definition 5.2.** In Transfer learning (TL), knowledge learned from a task is re-used in order to boost performance on a related task.
- **Idea:** Take a pretrained neural network, replace the last layer(s), and then train those on your own corpus.
- **Observation:** Simple but surprisingly efficient!

SOME RIGHTS RESERVED

Large Language Models

Definition 5.3. A Large Language Model (LLM) is a generic pretrained neural network, providing embeddings for sentences or entire documents for NLP tasks. In practice, they (usually) combine the following components:

- Tokenization: Splitting text into tokens (characters, words, punctuation,...)
- embeddings for these tokens, (e.g., Word2vec – or we let the transformer learn them)
- positional embeddings of tokens (encodes where in a sentence a token is)
- a transformer architecture, trained on
- a masked token prediction task.

LLMs can be used for a variety of tasks.

- *classification* (e.g., sentiment analysis, POS-tagging),
- *translation* (bwetween languages, styles, etc.),
- *generation* (e.g., text completion, summarization, chatbots),
- ...

SOME RIGHTS RESERVED

Tokenization - Byte Pair Encodings

So far: we have encoded text either as sequences of characters (non-semantic) or as sequences of words (semantic, but virtually unlimited vocabulary, OOV-problems).

Idea: Find a middle ground: Learn an optimal vocabulary of tokens from data and split text into a sequence of tokens.

Definition 5.4. The Byte Pair Encoding (BPE) algorithm learns a vocabulary of tokens of given size $N > 256$ from a corpus $\mathcal{C}$, by doing the following:

- Let $\ell = 256$ and set $\mathrm{BPE}(\langle b \rangle) = b$ for every byte $0 \leq b \leq 255$.
- While $\ell < N$, find the most common pair of tokens (a, b) and let $\mathrm{BPE}(\langle a, b \rangle) = \ell + 1$ (and increase ℓ by 1).
- Repeat until $\ell = N$.

$\leadsto$ we obtain a one-hot encoding of tokens of size N, where the most common sequences of bytes are represented by a single token. By retaining $\mathrm{BPE}(\langle b \rangle) = b$, we avoid OOV problems.

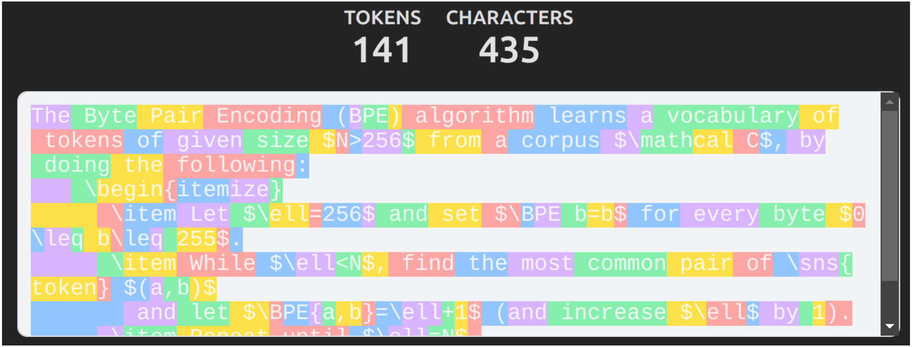
$\leadsto$ We can then train a word embedding on the resulting tokens

Alternative techniques include *WordPiece* and *SentencePiece*.

SOME RIGHTS RESERVED

Tokenization - Example

`https://huggingface.co/spaces/Xenova/the-tokenizer-playground`



```
The Byte Pair Encoding (BPE) algorithm learns a vocabulary of
 tokens of given size $N>256$ from a corpus $\mathcal C$, by
 doing the following:
     \begin{itemize}
         \item Let $\ell=256$ and set $\BPE b=b$ for every byte $0
\leq b\leq 255$.
     \item While $\ell<N$, find the most common pair of \sns{
token} $(a,b)$
             and let $\BPE{a,b}=\ell+1$ (and increase $\ell$ by 1).
     \item Repeat until $\ell=N$
```

Positional encodings

Definition 5.5. Let $\langle w_1, \ldots, w_n \rangle$ be a sequence of tokens. A positional encoding $\mathrm{PE}_i(w_i)$ is a vector that retains the position of w_i in the sequence *alongside* the word embedding of w_i.

We want positional encodings to satisfy the following properties:

1. $\mathrm{PE}_i(w) \neq \mathrm{PE}_j(w)$ for $i \neq j$,
2. PE should retain *distances*: if $i_1 - i_2 = j_1 - j_2$, then given the embeddings for w_1, w_2, we should be able to linearly transform $\langle \mathrm{PE}_{i_1}(w_1), \mathrm{PE}_{i_2}(w_2) \rangle$ into $\langle \mathrm{PE}_{j_1}(w_1), \mathrm{PE}_{j_2}(w_2) \rangle$.

$\leadsto$ no entirely separate embeddings for w_1, w_2 depending on positions

$\leadsto$ learning from short sentences generalizes (ideally) to longer ones

SOME RIGHTS RESERVED

Sinusoidal positional encoding

Idea: Let $\mathrm{PE}_t(w) = \mathrm{E}(w) + p_t$, for some suitable p_t (where $\mathrm{E}(w)$ is the word embedding for token w).
$\rightsquigarrow$ p_t has the same dimensionality as our embedding E.

Idea: Use a combination of sine and cosine functions with different frequencies for each dimension of the embedding.
Attention is all you need: For a vocabulary size d, we define

$$p_{t_i} := \begin{cases} \sin(\frac{t}{c^{2k/d}}) & \text{if } i = 2k \\ \cos(\frac{t}{c^{2k/d}}) & \text{if } i = 2k+1 \end{cases}$$

for some constant c. (10000 in the paper)
$\rightsquigarrow$ works for arbitrary sequence lengths and vocabulary sizes.

SOME RIGHTS RESERVED

Training Large Language Models

Three strategies for training LLMs:

- *Masked Token Prediction:* Given a sentence (e.g. "The river rose five feet"), randomly replace tokens by a special *mask* token (e.g. "The river `[MASK]` five feet"). The LLM should predict the masked tokens (e.g. "rose"). (BERT et al; well suited for *generic* tasks)
- *Discrimination:* Train a small masked token prediction model M. Given a masked sentence, let M generated possible completions. Train the actual model to distinguish between tokens generated by M and the *original* tokens. (Google Electra et al; well suited for *generic* tasks)
- *Next Token Prediction:* Given the (beginning of) a sentence, predict the next token in the sequence. (GPT et al; well suited for *generative* tasks)

⇝ All techniques turn an unlabelled corpus into a *supervised learning* task.

SOME RIGHTS RESERVED

Deep Learning for NLP: Evaluation

- Deep learning methods are currently dominant in NLP! (think ChatGPT)
  - Data-driven methods are easier to develop and maintain than symbolic ones
  - also perform better models crafted by humans (with reasonable effort)
- But problems remain;
  - DL methods work best on immense amounts of data. (small languages?)
  - LLM contain knowledge, but integration with symbolic methods elusive.

SOME RIGHTS RESERVED

Deep Learning for NLP: Evaluation

- Deep learning methods are currently dominant in NLP! (think ChatGPT)
  - Data-driven methods are easier to develop and maintain than symbolic ones
  - also perform better models crafted by humans (with reasonable effort)
- But problems remain;
  - DL methods work best on immense amounts of data. (small languages?)
  - LLM contain knowledge, but integration with symbolic methods elusive.
- DL4NLP methods do very well, but only after processing orders of magnitude more data than humans do for learning language.
- This suggests that there is of scope for new insigths from all areas.

SOME RIGHTS RESERVED

Chapter 32

What did we learn in AI 1/2?

SOME RIGHTS RESERVED

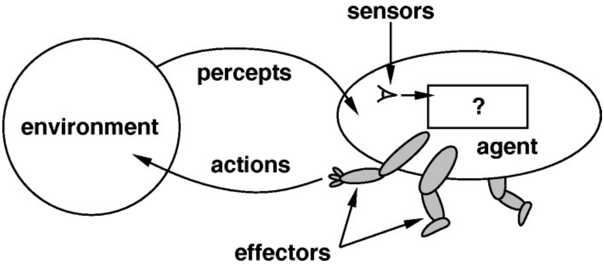
Topics of AI-1 (Winter Semester)

- Getting Started
  - What is artificial intelligence? (situating ourselves)
  - Logic programming in Prolog (An influential paradigm)
  - Intelligent Agents (a unifying framework)
- Problem Solving
  - Problem Solving and search (Black Box World States and Actions)
  - Adversarial search (Game playing) (A nice application of search)
  - constraint satisfaction problems (Factored World States)
- Knowledge and Reasoning
  - Formal Logic as the mathematics of Meaning
  - Propositional logic and satisfiability (Atomic Propositions)
  - First-order logic and theorem proving (Quantification)
  - Logic programming (Logic + Search$\rightsquigarrow$ Programming)
  - Description logics and semantic web
- Planning
  - Planning Frameworks
  - Planning Algorithms
  - Planning and Acting in the real world

SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

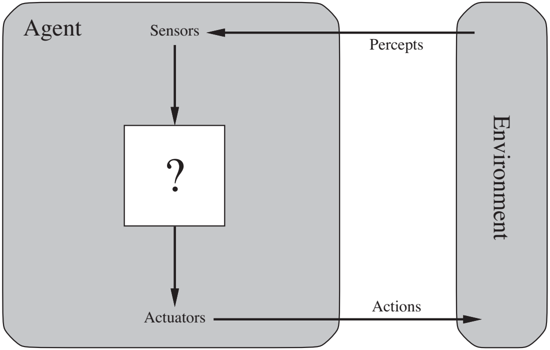
- Agents interact with the environment



SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

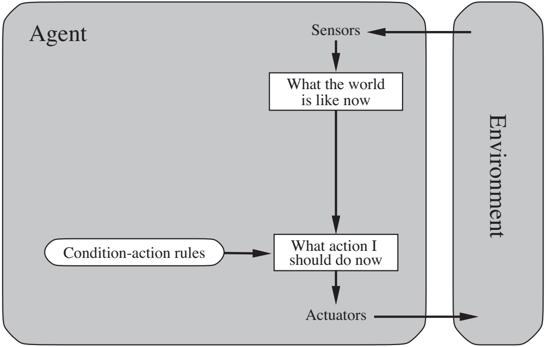
- General agent schema



SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

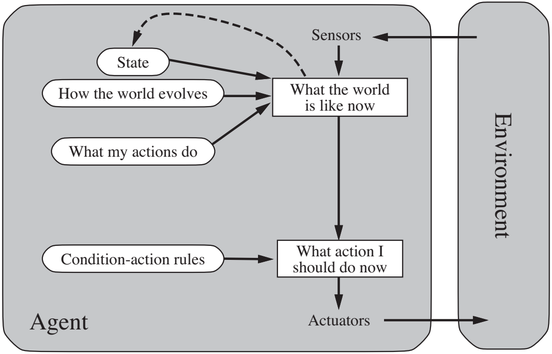
- Reflex Agents



SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

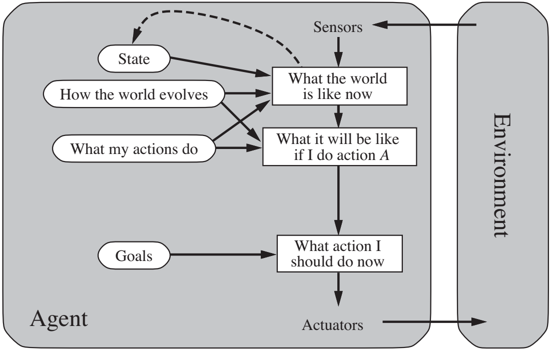
- Reflex Agents with State



SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

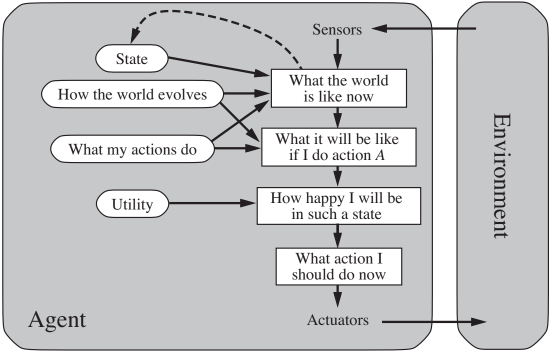
- Goal-Based Agents


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action *A*
Goals
What action I should do now
Agent
Actuators
Environment

SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

- Utility-Based Agent

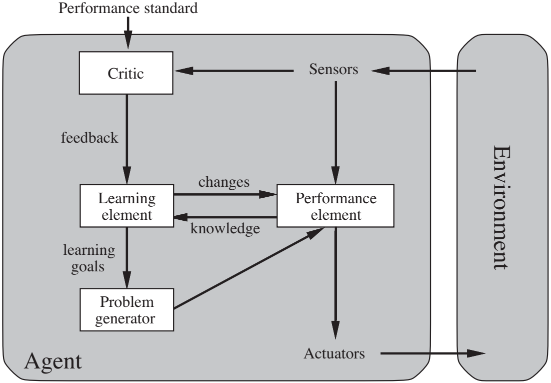


Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action *A*
How happy I will be in such a state
What action I should do now
Actuators
Agent
Environment

SOME RIGHTS RESERVED

Rational Agents as an Evaluation Framework for AI

- Learning Agents



Rational Agent

- **Idea:** Try to design agents that are successful (do the right thing)
- **Definition 0.1.** An agent is called rational, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date. This is called the MEU principle.
- **Note:** A rational agent need not be perfect
  - only needs to maximize expected value (rational $\neq$ omniscient)
    - need not predict e.g. very unlikely but catastrophic events in the future
  - percepts may not supply all relevant information (Rational $\neq$ clairvoyant)
    - if we cannot perceive things we do not need to react to them.
    - but we may need to try to find out about hidden dangers (exploration)
  - action outcomes may not be as expected (rational $\neq$ successful)
    - but we may need to take action to ensure that they do (more often) (learning)
- Rational $\rightsquigarrow$ exploration, learning, autonomy

SOME RIGHTS RESERVED

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)

SOME RIGHTS RESERVED

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)

SOME RIGHTS RESERVED

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)
- Describing world states by formal language (and drawing inferences)
  - Propositional logic and DPLL (deciding entailment efficiently)
  - First-order logic and ATP (reasoning about infinite domains)
  - **Digression**: Logic programming (logic + search)
  - Description logics as moderately expressive, but decidable logics

SOME RIGHTS RESERVED

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)
- Describing world states by formal language (and drawing inferences)
  - Propositional logic and DPLL (deciding entailment efficiently)
  - First-order logic and ATP (reasoning about infinite domains)
  - **Digression**: Logic programming (logic + search)
  - Description logics as moderately expressive, but decidable logics
- Planning: Problem Solving using white-box world/action descriptions
  - **Framework**: describing world states in logic as sets of propositions and actions by preconditions and add/delete lists
  - **Algorithms**: e.g heuristic search by problem relaxations

SOME RIGHTS RESERVED

Topics of AI-2 (Summer Semester)

- Uncertain Knowledge and Reasoning
  - Uncertainty
  - Probabilistic reasoning
  - Making Decisions in Episodic Environments
  - Problem Solving in Sequential Environments
- Foundations of machine learning
  - Learning from Observations
  - Knowledge in Learning
  - Statistical Learning Methods
- Communication (If there is time)
  - Natural Language Processing
  - Natural Language for Communication

SOME RIGHTS RESERVED

Statistical AI: Adding uncertainty and Learning

- Problem Solving under uncertainty (non-observable environment, stochastic states)
  - **Framework**: Probabilistic Inference: Conditional Probabilities/Independence
  - **Intuition**: Reasoning in Belief Space instead of State Space!
  - **Implementation**: Bayesian Networks (exploit conditional independence)
  - **Extension**: Utilities and Decision Theory (for static/episodic environments)

SOME RIGHTS RESERVED

Statistical AI: Adding uncertainty and Learning

- Problem Solving under uncertainty (non-observable environment, stochastic states)
  - **Framework**: Probabilistic Inference: Conditional Probabilities/Independence
  - **Intuition**: Reasoning in Belief Space instead of State Space!
  - **Implementation**: Bayesian Networks (exploit conditional independence)
  - **Extension**: Utilities and Decision Theory (for static/episodic environments)
- Problem Solving in Sequential Worlds:
  - **Framework**: Markov Processes, transition models
  - **Extension**: MDPs, POMDPs (+ utilities/decisions)
  - **Implementation**: Dynamic Bayesian Networks

SOME RIGHTS RESERVED

Statistical AI: Adding uncertainty and Learning

- Problem Solving under uncertainty (non-observable environment, stochastic states)
  - **Framework**: Probabilistic Inference: Conditional Probabilities/Independence
  - **Intuition**: Reasoning in Belief Space instead of State Space!
  - **Implementation**: Bayesian Networks (exploit conditional independence)
  - **Extension**: Utilities and Decision Theory (for static/episodic environments)
- Problem Solving in Sequential Worlds:
  - **Framework**: Markov Processes, transition models
  - **Extension**: MDPs, POMDPs (+ utilities/decisions)
  - **Implementation**: Dynamic Bayesian Networks
- Machine learning: adding optimization in changing environments (unsupervised)
  - **Framework**: Learning from Observations (positive/negative examples)
  - **Intuitions**: finding consistent/optimal hypotheses in a hypothesis space
  - **Problems**: consistency, expressivity, under/overfitting, computational/data resources.
  - Extensions
    - knowledge in learning (based on logical methods)
    - statistical learning (optimizing the probability distribution over hypspace, learning BNs)

SOME RIGHTS RESERVED

Statistical AI: Adding uncertainty and Learning

- Problem Solving under uncertainty (non-observable environment, stochastic states)
  - **Framework**: Probabilistic Inference: Conditional Probabilities/Independence
  - **Intuition**: Reasoning in Belief Space instead of State Space!
  - **Implementation**: Bayesian Networks (exploit conditional independence)
  - **Extension**: Utilities and Decision Theory (for static/episodic environments)
- Problem Solving in Sequential Worlds:
  - **Framework**: Markov Processes, transition models
  - **Extension**: MDPs, POMDPs (+ utilities/decisions)
  - **Implementation**: Dynamic Bayesian Networks
- Machine learning: adding optimization in changing environments (unsupervised)
  - **Framework**: Learning from Observations (positive/negative examples)
  - **Intuitions**: finding consistent/optimal hypotheses in a hypothesis space
  - **Problems**: consistency, expressivity, under/overfitting, computational/data resources.
  - Extensions
    - knowledge in learning (based on logical methods)
    - statistical learning (optimizing the probability distribution over hypspace, learning BNs)
  - Communication
    - Phenomena of natural language (NL is interesting/complex)
    - symbolic/statistical NLP (historic/as a backup)
    - Deep Learning for NLP (the current hype/sc

Topics of AI-3 – A Course not taught at FAU ☹

- Machine Learning
  - Theory and Practice of Deep Learning
  - More Reinforcement Learning
- Communicating, Perceiving, and Acting
  - More NLP, dialogue, speech acts, …
  - Natural Language Semantics/Pragmatics
  - Perception
  - Robotics
  - Emotions, Sentiment Analysis
- **The Good News:** All is not lost
  - There are tons of specialized courses at FAU (more as we speak)
  - Russell/Norvig's AIMA [**RusNor:AIMA09**] cover some of them as well!

SOME RIGHTS RESERVED

References I

SOME RIGHTS RESERVED