Artificial Intelligence 1
Winter Semester 2023/24

– Lecture Notes –

Prof. Dr. Michael Kohlhase

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

2024-02-08

Chapter 1
Preliminaries

1.1 Administrative Ground Rules

Prerequisites for AI-1

- **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)

Prerequisites for AI-1

- **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 computer science)
  - It allows us to be very precise about what we mean. (good for you)

Prerequisites for AI-1

- **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 computer science)
  - 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!

Prerequisites for AI-1

- **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 computer science)
  - 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-1 is non-trivial)
- You can do this course if you want! (and I hope you are successful)

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. ($\leq$ 50% $\rightsquigarrow$ no bonus)
  - Bonus points $\widehat{=}$ percentage sum of the best 10 tuesday quizzes divided by 100.

Assessment, Grades

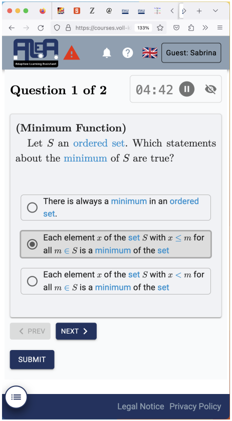
- **Overall (Module) Grade:**
  - Grade via the exam (Klausur) $\rightsquigarrow$ 100% of the grade.
  - Up to 10% bonus on-top for an exam with $\geq$ 50% points. ($\leq$ 50% $\rightsquigarrow$ no bonus)
  - Bonus points $\widehat{=}$ percentage sum of the best 10 tuesday quizzes divided by 100.
- **Exam:** 90 minutes exam conducted in presence on paper (~ April 1. 2024)
- **Retake Exam:** 90 min exam six months later (~ October 1. 2024)
- You have to register for exams in campo in the first month of classes.
- **Note:** You can de-register from an exam on campo up to three working days before.

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. ($\leq$ 50% $\rightsquigarrow$ no bonus)
  - Bonus points $\widehat{=}$ percentage sum of the best 10 tuesday quizzes divided by 100.
- **Exam:** 90 minutes exam conducted in presence on paper (~ April 1. 2024)
- **Retake Exam:** 90 min exam six months later (~ October 1. 2024)
- ⚠ You have to register for exams in campo in the first month of classes.
- **Note:** You can de-register from an exam on campo up to three working days before.
- **Tuesday Quizzes:** Every tuesday we start the lecture with a 10 min online quiz – the tuesday quiz – about the material from the previous week.(starts in week 2)

Tuesday Quizzes

- **Tuesday Quizzes:** Every tuesday we start the lecture with a 10 min online quiz – the tuesday quiz – about the material from the previous week.(starts in week 2)
- **Motivations:** We do this to
  - keep you prepared and working continuously. (primary)
  - update the ALeA learner model (fringe benefit)
- The tuesday quiz will be given in the ALeA system
  - `https://courses.voll-ki.fau.de/quiz-dash/ai-1`
  - You have to be logged into ALeA!
  - You can take the quiz on your laptop or phone, ...
  - ... in the lecture or at home ...
  - ... via WLAN or 4G Network. (do not overload)
  - Quizzes will only be available 16:15-16:25!



Tomorrow: Pretest

- ⚠ Tomorrow we will try out the tuesday quiz 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: $\geq$ March 2023)
  - Make sure that you are logged into ALeA (via FAU IDM; see below)
- **Definition 1.1.** 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 AI-1 and
  - tests the ALeA quiz infrastructure for the tuesday quizzes.
- Participation in this test is optional; it will not influence your grades in any way.
- The test covers the prerequisites of AI-1 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)

Special Admin Conditions

- Some degree programs do not "import" the course Artificial Intelligence, and thus you may not be able to register for the exam via `https://campus.fau.de`.
  - Just send me an e-mail and come to the exam, we will issue a "Schein".
  - 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 $\leftrightsquigarrow 7.5 + 7.5 = 15$.

1.2 Getting Most out of AI-1

AI-1 Homework Assignments

- **Homework Assignments:** Small individual problem/programming/proof task
  - but take time to solve (at least read them directly $\leadsto$ questions)
- ⚠ Homeworks give no bonus points, but without trying you are unlikely to pass the exam.

AI-1 Homework Assignments

- **Homework Assignments:** Small individual problem/programming/proof task
  - but take time to solve (at least read them directly $\leadsto$ questions)
- Homeworks give no bonus points, but without trying you are unlikely to pass the exam.
- **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 group 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!)

AI-1 Homework Assignments

- **Homework Assignments:** Small individual problem/programming/proof task
  - but take time to solve (at least read them directly ⇝ questions)
- ⚠ Homeworks give no bonus points, but without trying you are unlikely to pass the exam.
- **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 group 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!)
- ⚠ We will not be able to grade all homework assignments!
- **Graded Assignments:** To keep things running smoothly
  - Homeworks will be posted on StudOn.
  - Sign up for AI-1 under `https://www.studon.fau.de/crs4622069.html`.
  - Homeworks are handed in electronically there. (plain text, program files, PDF)
  - Do not sign up for the "AI-2 Übungen" on StudOn (we do not use them)
- **Ungraded Assignments:** Are peer-feedbacked in ALeA (see below)

Tutorials for Artificial Intelligence 1

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

Tutorials for Artificial Intelligence 1

- **Approach:** Weekly tutorials and homework assignments (first one in week two)
- **Goal 1:** Reinforce what was taught in class. (you need practice)
- **Goal 2:** Allow you to ask any question you have in a protected environment.
- **Instructor/Lead TA:** Florian Rabe (KWARC Postdoc)
  - Room: 11.137 @ Händler building, `florian.rabe@fau.de`
- **Tutorials:** One each taught by Florian Rabe (lead); Mahdi Mantash, Robert Kurin, Florian Guthmann.
- **Life-saving Advice:** Go to your tutorial, and prepare for it by having looked at the slides and the homework assignments!

Tutorials for Artificial Intelligence 1

- **Approach:** Weekly tutorials and homework assignments (first one in week two)
- **Goal 1:** Reinforce what was taught in class. (you need practice)
- **Goal 2:** Allow you to ask any question you have in a protected environment.
- **Instructor/Lead TA:** Florian Rabe (KWARC Postdoc)
  - Room: 11.137 @ Händler building, `florian.rabe@fau.de`
- **Tutorials:** One each taught by Florian Rabe (lead); Mahdi Mantash, Robert Kurin, Florian Guthmann.
- **Life-saving Advice:** Go to your tutorial, and prepare for it by having looked at the slides and the homework assignments!
- **Caveat:** We cannot grade all submissions with 5 TAs and ∼1000 students.
- **Also:** Group submission has not worked well in the past! (too many freeloaders)

Collaboration

- **Definition 2.1.** Collaboration (or cooperation) is the process of groups of agents working or acting together for common, mutual, or some underlying benefit, as opposed to working in competition for selfish benefit. In a collaboration, every agent contributes to the common goal.
- In learning situations, the benefit is "better learning outcomes".
- **Observation:** In collaborative learning, the overall result can be significantly better than in competitive learning.
- **Good Practice:** Form study groups. (long- or short-term)
  - ⚠ those learners who work most, learn most
  - ⚠ freeloaders – indivicuals who only watch – learn very little!
- It is OK to collaborate on homework assignments in AI-1! (no bonus points)
- Choose your study group well (We will (eventually) help via ALeA)

Do I need to attend the lectures

- Attendance is not mandatory for the AI-1 lecture

Do I need to attend the lectures

- Attendance is not mandatory for the AI-1 lecture
- There are two ways of learning AI-1: (both are OK, your mileage may vary)
  - Approach B: Read a Book
  - Approach I: come to the lectures, be involved, interrupt me whenever you have a question.

  The only advantage of I over B is that books do not answer questions (yet! ⇝ we are working on this in AI research)

Do I need to attend the lectures

- Attendance is not mandatory for the AI-1 lecture
- There are two ways of learning AI-1: (both are OK, your mileage may vary)
  - Approach B: Read a Book
  - Approach I: come to the lectures, be involved, interrupt me whenever you have a question.

  The only advantage of I over B is that books do not answer questions (yet! ⇝ we are working on this in AI research)
- Approach S: come to the lectures and sleep does not work!

Do I need to attend the lectures

- Attendance is not mandatory for the AI-1 lecture
- There are two ways of learning AI-1: (both are OK, your mileage may vary)
  - Approach B: Read a Book
  - Approach I: come to the lectures, be involved, interrupt me whenever you have a question.

  The only advantage of I over B is that books do not answer questions (yet! ⇝ we are working on this in AI research)
- Approach S: come to the lectures and sleep does not work!
- **I really mean it:** If you come to class, be involved, ask questions, challenge me with comments, tell me about errors, . . .

Do I need to attend the lectures

- Attendance is not mandatory for the AI-1 lecture
- There are two ways of learning AI-1: (both are OK, your mileage may vary)
  - Approach B: Read a Book
  - Approach I: come to the lectures, be involved, interrupt me whenever you have a question.

  The only advantage of I over B is that books do not answer questions (yet! ⇝ we are working on this in AI research)
- Approach S: come to the lectures and sleep does not work!
- **I really mean it:** If you come to class, be involved, ask questions, challenge me with comments, tell me about errors, ...
  - I would much rather have a lively discussion than get through all the slides

Do I need to attend the lectures

- Attendance is not mandatory for the AI-1 lecture
- There are two ways of learning AI-1: (both are OK, your mileage may vary)
  - Approach B: Read a Book
  - Approach I: come to the lectures, be involved, interrupt me whenever you have a question.

  The only advantage of I over B is that books do not answer questions (yet! ⇝ we are working on this in AI research)
- Approach S: come to the lectures and sleep does not work!
- **I really mean it:** If you come to class, be involved, ask questions, challenge me with comments, tell me about errors, ...
  - I would much rather have a lively discussion than get through all the slides
  - You learn more, I have more fun (Approach B serves as a backup)

Do I need to attend the lectures

- Attendance is not mandatory for the AI-1 lecture
- There are two ways of learning AI-1: (both are OK, your mileage may vary)
  - Approach B: Read a Book
  - Approach I: come to the lectures, be involved, interrupt me whenever you have a question.

  The only advantage of I over B is that books do not answer questions (yet! ⇝ we are working on this in AI research)
- Approach S: come to the lectures and sleep does not work!
- **I really mean it:** If you come to class, be involved, ask questions, challenge me with comments, tell me about errors, ...
  - I would much rather have a lively discussion than get through all the slides
  - You learn more, I have more fun (Approach B serves as a backup)
  - You may have to change your habits, overcome shyness, ... (please do!)

Do I need to attend the lectures

- Attendance is not mandatory for the AI-1 lecture
- There are two ways of learning AI-1: (both are OK, your mileage may vary)
  - Approach B: Read a Book
  - Approach I: come to the lectures, be involved, interrupt me whenever you have a question.

  The only advantage of I over B is that books do not answer questions (yet! ⇝ we are working on this in AI research)
- Approach S: come to the lectures and sleep does not work!
- **I really mean it:** If you come to class, be involved, ask questions, challenge me with comments, tell me about errors, ...
  - I would much rather have a lively discussion than get through all the slides
  - You learn more, I have more fun (Approach B serves as a backup)
  - You may have to change your habits, overcome shyness, ... (please do!)
- This is what I get paid for, and I am more expensive than most books (get your money's worth)

1.3 Learning Resources for AI-1

Textbook, Handouts and Information, Forums, Videos

- **Textbook:** *Russel/Norvig: Artificial Intelligence, A modern Approach* [RN09].
  - basically "broad but somewhat shallow"
  - great to get intuitions on the basics of AI

  Make sure that you read the edition ≥ 3 $\leftsquigarrow$ vastly improved over ≤ 2.

Textbook, Handouts and Information, Forums, Videos

- **Textbook:** *Russel/Norvig: Artificial Intelligence, A modern Approach* [RN09].
  - basically "broad but somewhat shallow"
  - great to get intuitions on the basics of AI

  Make sure that you read the edition ≥ 3 $\leftsquigarrow$ vastly improved over ≤ 2.
- **Course notes:** will be posted at `http://kwarc.info/teaching/AI/notes.pdf`
  - more detailed than [RN09] in some areas
  - I mostly prepare them as we go along (semantically preloaded $\leadsto$ research resource)
  - please e-mail me any errors/shortcomings you notice. (improve for the group)

Textbook, Handouts and Information, Forums, Videos

- **Textbook:** *Russel/Norvig: Artificial Intelligence, A modern Approach* [RN09].
  - basically "broad but somewhat shallow"
  - great to get intuitions on the basics of AI

  Make sure that you read the edition ≥ 3 $\leftsquigarrow$ vastly improved over ≤ 2.
- **Course notes:** will be posted at `http://kwarc.info/teaching/AI/notes.pdf`
  - more detailed than [RN09] in some areas
  - I mostly prepare them as we go along (semantically preloaded $\rightsquigarrow$ research resource)
  - please e-mail me any errors/shortcomings you notice. (improve for the group)
- **StudOn Forum:** `https://www.studon.fau.de/crs4622069.html` for
  - announcements, homeworks (my view on the forum)
  - questions, discussion among your fellow students (your forum too, use it!)

Textbook, Handouts and Information, Forums, Videos

- **Textbook:** *Russel/Norvig: Artificial Intelligence, A modern Approach* [RN09].
  - basically "broad but somewhat shallow"
  - great to get intuitions on the basics of AI

  Make sure that you read the edition ≥ 3 $\leftsquigarrow$ vastly improved over ≤ 2.
- **Course notes:** will be posted at `http://kwarc.info/teaching/AI/notes.pdf`
  - more detailed than [RN09] in some areas
  - I mostly prepare them as we go along (semantically preloaded $\rightsquigarrow$ research resource)
  - please e-mail me any errors/shortcomings you notice. (improve for the group)
- **StudOn Forum:** `https://www.studon.fau.de/crs4622069.html` for
  - announcements, homeworks (my view on the forum)
  - questions, discussion among your fellow students (your forum too, use it!)
- **Course Videos:** AI-1 will be streamed/recorded at `https://fau.tv/course/id/3595`
  - **Organized**: Video course nuggets are available at `https://fau.tv/course/id/1690` (short; organized by topic)
  - **Backup**: The lectures from WS 2016/17 to SS 2018 have been recorded (in English and German), see `https://www.fau.tv/search/term.html?q=Kohlhase`

Textbook, Handouts and Information, Forums, Videos

- **Textbook:** *Russel/Norvig: Artificial Intelligence, A modern Approach* [RN09].
  - basically "broad but somewhat shallow"
  - great to get intuitions on the basics of AI

  Make sure that you read the edition ≥ 3 $\leftsquigarrow$ vastly improved over ≤ 2.
- **Course notes:** will be posted at `http://kwarc.info/teaching/AI/notes.pdf`
  - more detailed than [RN09] in some areas
  - I mostly prepare them as we go along (semantically preloaded $\rightsquigarrow$ research resource)
  - please e-mail me any errors/shortcomings you notice. (improve for the group)
- **StudOn Forum:** `https://www.studon.fau.de/crs4622069.html` for
  - announcements, homeworks (my view on the forum)
  - questions, discussion among your fellow students (your forum too, use it!)
- **Course Videos:** AI-1 will be streamed/recorded at `https://fau.tv/course/id/3595`
  - **Organized**: Video course nuggets are available at `https://fau.tv/course/id/1690` (short; organized by topic)
  - **Backup**: The lectures from WS 2016/17 to SS 2018 have been recorded (in English and German), see `https://www.fau.tv/search/term.html?q=Kohlhase`
- **Do not let the videos mislead you:** Coming to class is highly correlated with passing the course!

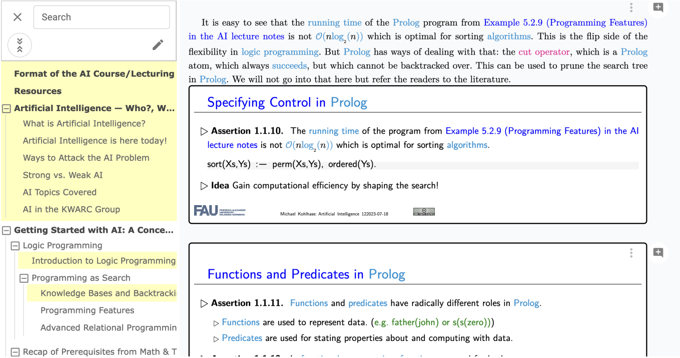
Practical recommendations on Lecture Resources

- Excellent Guide: [Nor+18a] (german Version at [Nor+18b])

1.4 AI-Supported Learning

ALeA: Adaptive Learning Assistant

- **Idea:** Use AI methods to help teach/learn AI (AI4AI)
- **Concretely:** Provide HTML versions of the AI-1 slides/notes and embed learning support services into them. (for pre/postparation of lectures)
- **Definition 4.1.** Call a document active, iff it is interactive and adapts to specific information needs of the readers. (course notes on steroids)
- **Intuition:** ALeA serves active course materials. (PDF mostly inactive)
- **Goal:** Make ALeA more like a teacher + study group than like a book
- **Example 4.2 (Course Notes).** $\widehat{=}$ Slides + Comments



yellow parts in table of contents (left) already covered in lecture

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

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

- **AI-1 in ALeA:** https://courses.voll-ki.fau.de/course-home/ai-1
  - All details for the course.
  - recorded syllabus (keep track of material covered in course)
  - syllabus of the last semester (for over/preview)
- **ALeA Status:** The ALeA system is deployed at FAU for over 1000 students taking six 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.

Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 4.6 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of the definition. (even works recursively)

A Conce...

rch

at

Strategies

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.

te.

unction

a goal

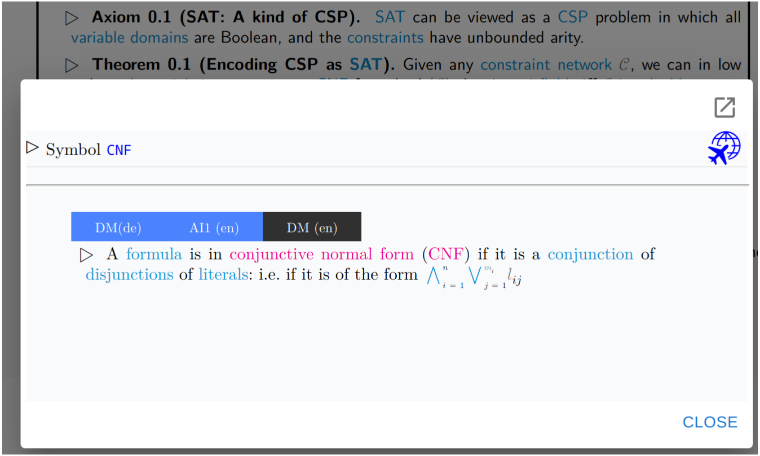
state, or ∞ if no such path exists, is called the goal distance function for Π.

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 $\mathcal{I} \subseteq \mathcal{S}$.

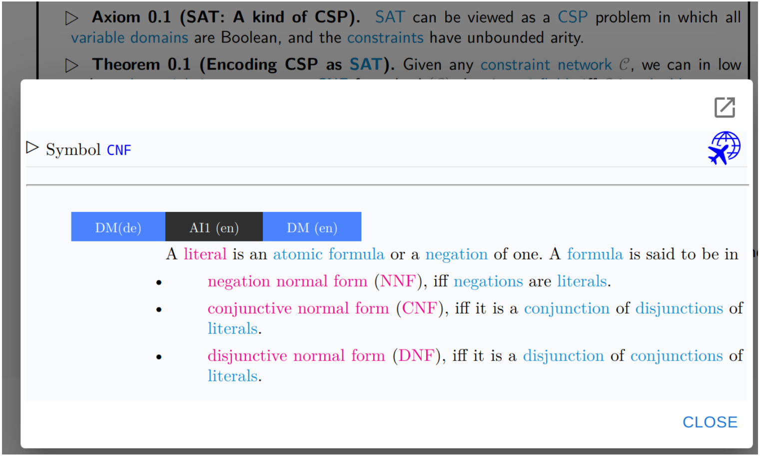
Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 4.9 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of the definition. (even works recursively)
- **Example 4.10 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.



Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 4.12 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of the definition. (even works recursively)
- **Example 4.13 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.



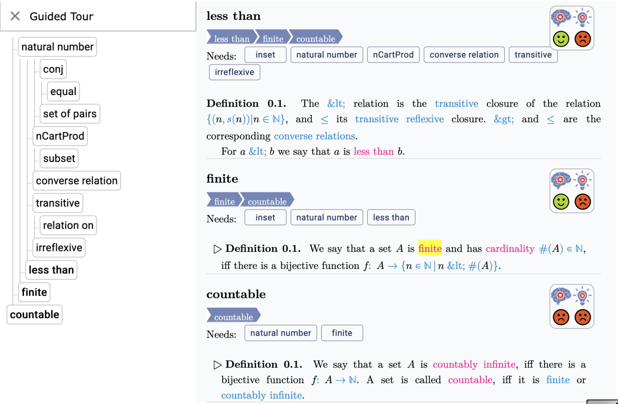
Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 4.15 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of the definition. (even works recursively)
- **Example 4.16 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.

Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 4.18 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of the definition. (even works recursively)
- **Example 4.19 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.
- **Example 4.20 (Guided Tour).** A guided tour for a concept c assembles definitions/etc. into a self-contained mini-course culminating at c.

$c =$ countable $\leadsto$



Learning Support Services in ALeA

- **Idea:** Embed learning support services into active course materials.
- **Example 4.21 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of the definition. (even works recursively)
- **Example 4.22 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.
- **Example 4.23 (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)

(Practice) Problems Everywhere

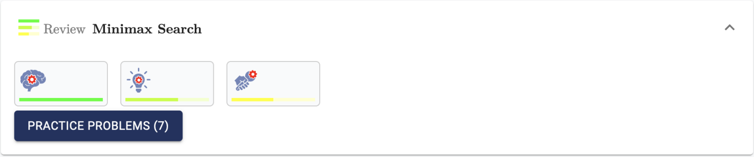
- **Problem:** Learning requires a mix of understanding and test-driven practice.
- **Idea:** ALeA supplies targeted practice problems everywhere.
- **Concretely:** Revision markers at the end of sections.

(Practice) 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

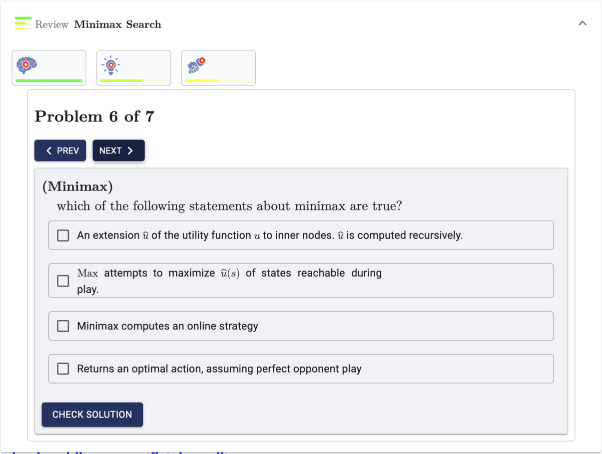
(Practice) Problems Everywhere

- **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.



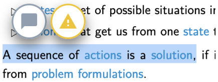
(Practice) Problems Everywhere

- **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 competencies)



Localized Interactions with the Community

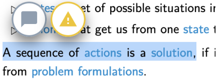
- Selecting text brings up localized – i.e. anchored on the selection – interactions:



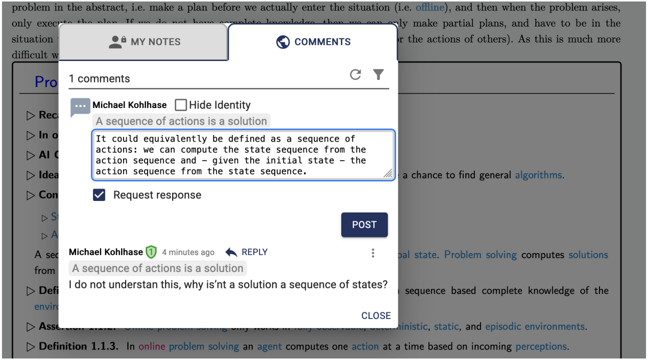
  - post a (public) comment or take (private) note
  - report an error to the course authors/instructors

Localized Interactions with the Community

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



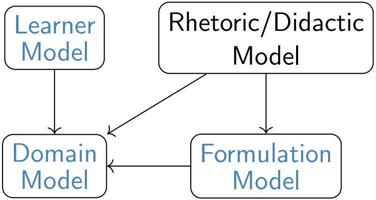
  - post a (public) comment or take (private) note
  - report an error to the course authors/instructors
- Localized comments induce a thread in the ALeA forum (like the StudOn Forum, but targeted towards specific learning objects)



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

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

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

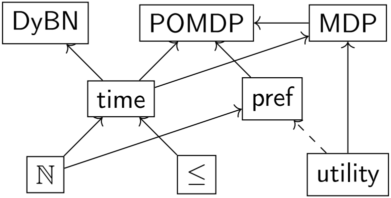


(Good) teachers

- 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

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

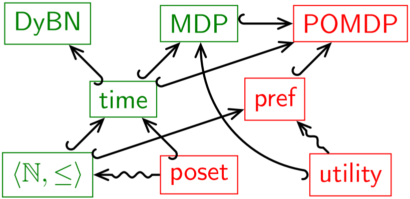


A theory graph provides (modular representation of the domain)

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

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

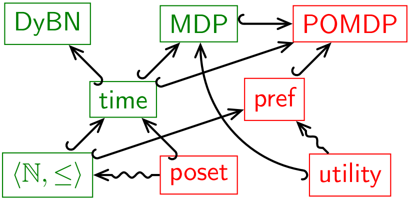


The learner model is a function from learner IDs $\times$ symbol URIs to competency values

- 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

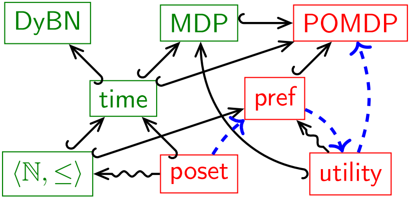


Learning objects are the text fragments learners see and interact with; they are structured by

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

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
- **Ingredient 2:** Learner model $\hat{=}$ adding competency estimations
- **Ingredient 3:** A collection of ready-formulated learning objects
- **Ingredient 4:** Educational dialogue planner $\leadsto$ guided tours

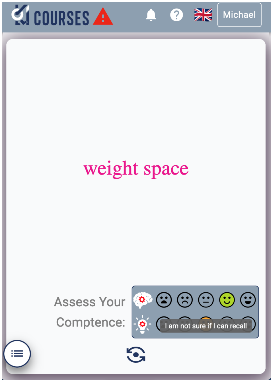


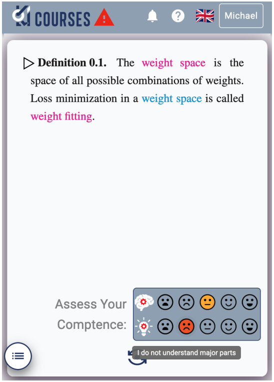
The dialogue planner assembles learning objects into active course materials using

- 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

New Feature: Drilling with Flashcards

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





...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)

Learner Data and Privacy in ALeA

- **Observation:** Most 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!

Learner Data and Privacy in ALeA

- **Observation:** Most 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)

Learner Data and Privacy in ALeA

- **Observation:** Most 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.

Concrete Todos for ALeA

- **Recall:** You will use ALeA for the tuesday quizzes (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-1`
  - 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)

Concrete Todos for ALeA

- **Recall:** You will use ALeA for the tuesday quizzes (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-1`
  - 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)

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

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?

2.1 What is Artificial Intelligence?

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 computer science 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.

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

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

What is Artificial Intelligence? Components

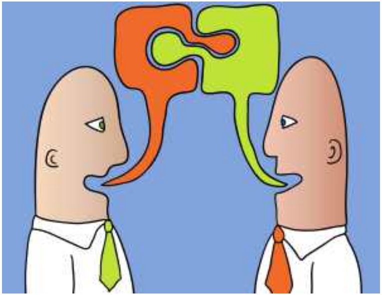
- **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.
- This needs a combination of

Perception

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

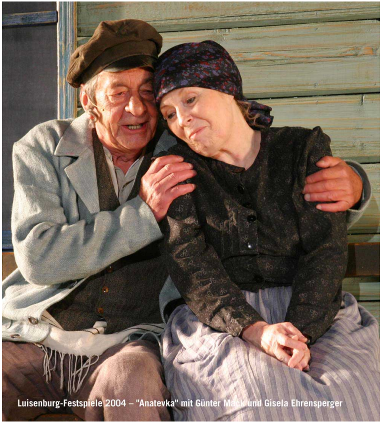
Language understanding



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

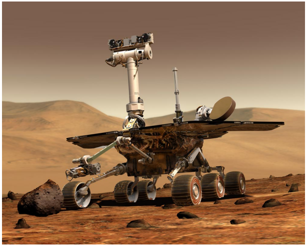
Emotion

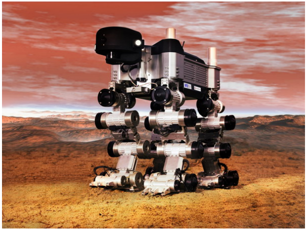


2.2 Artificial Intelligence is here today!

Artificial Intelligence is here today!

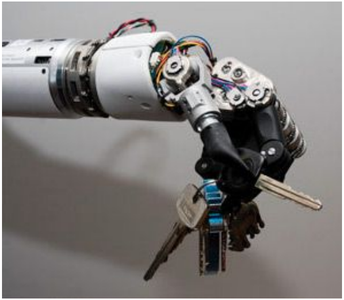
- in outer space
  - in outer space systems need autonomous control:
  - remote control impossible due to time lag
- in artificial limbs
- in household appliances
- in hospitals
- for safety/security





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

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

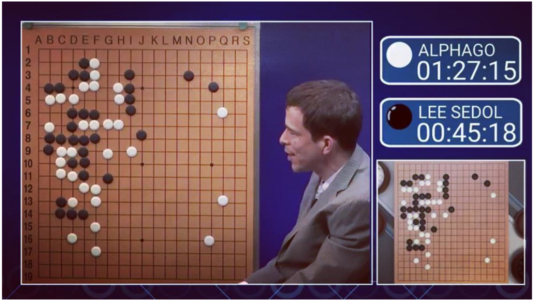
Artificial Intelligence is here today!

- 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

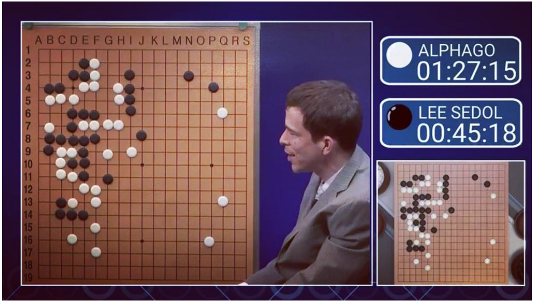
"It's the latest innovation in office safety. When your computer crashes, an air bag is activated so you won't bang your head in frustration."

And here's what you all have been waiting for . . .



- AlphaGo is a program by Google DeepMind to play the board game go.
- In March 2016, it beat Lee Sedol in a five-game match, the first time a go program has beaten a 9 dan professional without handicaps.

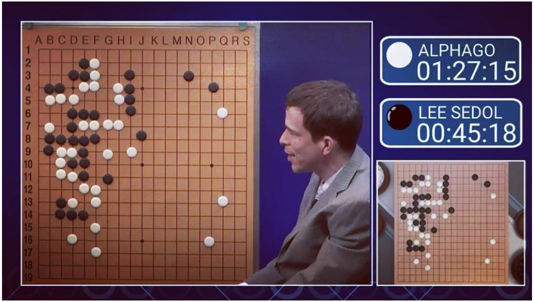
And here's what you all have been waiting for . . .



- AlphaGo is a program by Google DeepMind to play the board game go.

  In December 2017 AlphaZero, a successor of AlphaGo "learned" the games go, chess, and shogi in 24 hours, achieving a superhuman level of play in these three games by defeating world-champion programs.

And here's what you all have been waiting for . . .



- AlphaGo is a program by Google DeepMind to play the board game go.

  By September 2019, AlphaStar, a variant of AlphaGo, attained "grandmaster level" in Starcraft II, a real time strategy game with partially observable state. AlphaStar now among the top 0.2% of human players.

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.
- **Another Consequence:** AI as a field is an incubator for many innovative technologies.
- **AI Conundrum:** Once AI solves a subfield it is called "computer science". (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.

2.3 Ways to Attack the AI Problem

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.

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.

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.

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.

Two ways of reaching Artificial Intelligence?

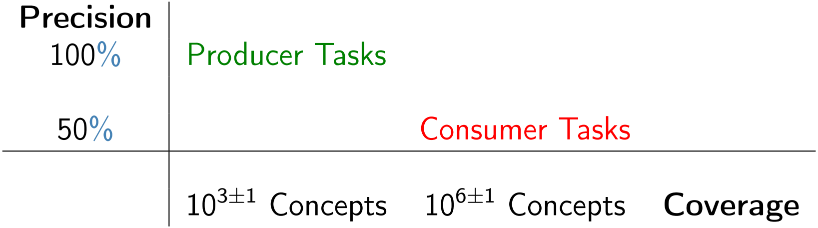
- We can classify the AI approaches by their coverage and the analysis depth (they are complementary)

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)

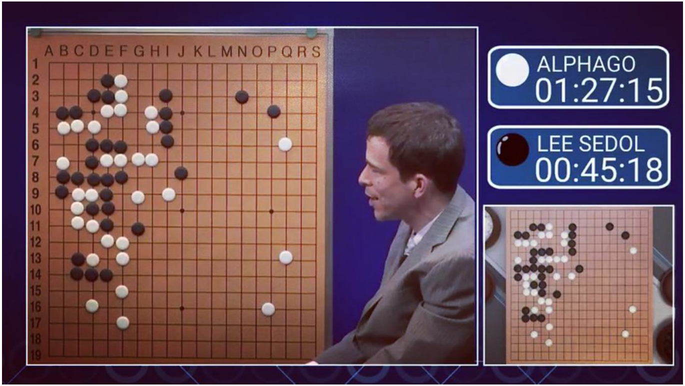
Environmental Niches for both Approaches to AI

- **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)



- **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.

To get this out of the way . . .



- AlphaGo = search + neural networks (symbolic + subsymbolic AI)
  - we do search this semester and cover neural networks in AI-2.
  - I will explain AlphaGo a bit in .

2.4 Strong vs. Weak AI

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.
- **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!

A few words on AGI...

- The conceptual and mathematical framework (agents, environments etc.) 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)

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)

2.5 AI Topics Covered

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 $\leadsto$ Programming)
  - Description logics and semantic web
- Planning
  - Planning Frameworks
  - Planning Algorithms
  - Planning and Acting in the real world

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

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

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)

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)

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)

2.6 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 $\widehat{=}$ 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

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 $\rightarrow$ 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		

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, ...
- We always try to find a topic at the intersection of your and our interests.
- We also often have positions!. (HiWi, Ph.D.: $\frac{1}{2}$, PostDoc: full)

Part 1
Getting Started with AI: A Conceptual Framework

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!

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-1
  2. prepare a conceptual framework in which we can think about "intelligence" (natural and artificial), and
  3. recap some methods and results from theoretical computer science.

Chapter 3
Logic Programming

3.1 Introduction to Logic Programming and ProLog

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; [Kow97])
- We will use the programming language Prolog as an example.

Prolog Terms and Literals

- **Definition 1.2.** Prologs expresses knowledge about the world via
  - constants denoted by lower case strings,
  - variables denoted by upper-case strings or starting with _, and
  - functions and predicates (lower-case 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)),...

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 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 upper-case) 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.

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 terms called goal literal (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
```

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
```

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
```

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:** $[\mathrm{sokrates}/X]$

3.2 Programming as Search

3.2.1 Knowledge Bases and Backtracking

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 employes 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 ;.

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
```

3.2.2 Programming Features

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 n^{th} 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 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.
```

Adding Lists to Prolog

- Lists are represented by terms of the form [a,b,c,...]
- First/rest representation [F|R], where R is a rest list.
- 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).
```

Relational Programming Techniques

- **Example 2.7.** Parameters have no unique direction "in" or "out"

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

- **Example 2.8.** Symbolic programming by structural induction

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

- **Example 2.9.** Generate and test:

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

3.2.3 Advanced Relational Programming

Specifying Control in Prolog

- *Remark 2.10.* The running time of the program from 2.9 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!

Functions and Predicates in Prolog

- *Remark 2.11.* 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.
- *Remark 2.12.* In functional programming, functions are used for both. (even more confusing than in Prolog if you think about it)

Functions and Predicates in Prolog

- *Remark 2.15.* 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 2.16.* In functional programming, functions are used for both. (even more confusing than in Prolog if you think about it)
- **Example 2.17.** Consider again the reverse program 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.

Functions and Predicates in Prolog

- *Remark 2.19.* 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 2.20.* In functional programming, functions are used for both.
  (even more confusing than in Prolog if you think about it)
- **Example 2.21.** Consider again the reverse program 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 2.22 (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([X|R]),Y) :− leafcount(X,Z), leafcount(t(R,W)), Y is Z + W.
```

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 [Fis; LPN],
  - [Fla94] has a very thorough logic-based introduction,
  - consult also the SWI Prolog Manual [SWI],

Chapter 4
Recap of Prerequisites from Math & Theoretical Computer Science

4.1 Recap: Complexity Analysis in AI?

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 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	$.5ms$	$175\mu s$	$32\mu s$
10	$1ms$	$.7ms$	$1ms$
45	$4.5ms$	$14ms$	$1.1Y$
100	...	...	...
1 000	...	...	...
10 000	...	...	...
1 000 000	...	...	...

What?! One year?

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

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

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-1.

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-1.
- **Definition:** Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that analgorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha){:=}t$.
  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 $s{\in}S$.

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-1.
- **Definition:** Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that analgorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha){:=}t$.
  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)

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-1.
- **Definition:** Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that analgorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha){:=}t$.
  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:** 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
$\mathcal{O}(\ln(n))$	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 $n{>}n_0$.

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-1.
- **Definition:** Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that analgorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha){:=}t$.
  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:** 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
$\mathcal{O}(\ln(n))$	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 $n{>}n_0$.
- For $k'{>}2$ and $k{>}1$ we have $\mathcal{O}(1){\subset}\mathcal{O}(\log n){\subset}\mathcal{O}(n){\subset}\mathcal{O}(n^2){\subset}\mathcal{O}(n^{k'}){\subset}\mathcal{O}(k^n)$

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-1.
- **Definition:** Let $S \subseteq \mathbb{N} \to \mathbb{N}$ be a set of natural number functions, then we say that analgorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha):=t$.
  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:** 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
$\mathcal{O}(\ln(n))$	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 $n>n_0$.
- For $k'>2$ and $k>1$ we have $\mathcal{O}(1)\subset\mathcal{O}(\log n)\subset\mathcal{O}(n)\subset\mathcal{O}(n^2)\subset\mathcal{O}(n^{k'})\subset\mathcal{O}(k^n)$
- **For AI-1:** I expect that given an algorithm, you can determine its complexity class. (next)

Determining the Time/Space Complexity of Algorithms

- **Definition 1.3.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by induction on the structure of α:
  - constant: can be accessed in constant time

Determining the Time/Space Complexity of Algorithms

- **Definition 1.4.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by induction on the structure of α:
  - constant: If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.

Determining the Time/Space Complexity of Algorithms

- **Definition 1.5.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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

Determining the Time/Space Complexity of Algorithms

- **Definition 1.6.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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 $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.

Determining the Time/Space Complexity of Algorithms

- **Definition 1.7.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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

Determining the Time/Space Complexity of Algorithms

- **Definition 1.8.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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 $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.

Determining the Time/Space Complexity of Algorithms

- **Definition 1.9.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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 compupte the value $\leadsto$ has its complexity

Determining the Time/Space Complexity of Algorithms

- **Definition 1.10.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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 $C_\Gamma(\alpha) = \Gamma \cup (v, S)$.

Determining the Time/Space Complexity of Algorithms

- **Definition 1.11.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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

Determining the Time/Space Complexity of Algorithms

- **Definition 1.12.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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 $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\psi)}(\psi)$.

Determining the Time/Space Complexity of Algorithms

- **Definition 1.13.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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

Determining the Time/Space Complexity of Algorithms

- **Definition 1.14.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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 $C_\Gamma(\alpha) = \Gamma \cup C_\Gamma(\gamma) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \cup C_{\Gamma \cup C_\Gamma(\gamma)}(\psi)$.

Determining the Time/Space Complexity of Algorithms

- **Definition 1.15.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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: multiplies complexities

Determining the Time/Space Complexity of Algorithms

- **Definition 1.16.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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 $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi)$.

Determining the Time/Space Complexity of Algorithms

- **Definition 1.17.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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.

Determining the Time/Space Complexity of Algorithms

- **Definition 1.18.** Given a function Γ that maps variables v to sets $\Gamma(v)$, we compute $T_\Gamma(\alpha)$ and $C_\Gamma(\alpha)$ of an imperative algorithm α by 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: If α is $\mathbf{while}\gamma\mathbf{do}\varphi\mathbf{end}$, 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 $\leadsto$ recurrence relations and Master's theorem.

Why Complexity Analysis? (General)

- **Example 1.19.** 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.

Why Complexity Analysis? (General)

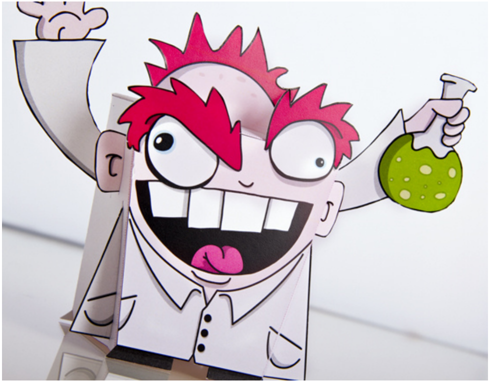
- **Example 1.20.** Once upon a time I was trying to invent an efficient algorithm.
  - My first algorithm attempt didn't work, so I had to try harder.
  - But my 2nd attempt didn't work either, which got me a bit agitated.

Why Complexity Analysis? (General)

- **Example 1.21.** 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...

Why Complexity Analysis? (General)

- **Example 1.22.** 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:



Why Complexity Analysis? (General)

- **Example 1.23.** 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 hardness was there to rescue me.

Why Complexity Analysis? (General)

- **Example 1.24.** 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.

Reminder (?): NP and PSPACE (details $\rightsquigarrow$ e.g. [GJ79])

- **Turing Machine:** Works on a tape consisting of cells, across which its Read/Write head moves. The machine has internal states. There is a transition function 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)

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 hardness help?

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 hardness 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*?

4.2 Recap: Formal Languages and Grammars

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 $\langle \mathtt{a}, \mathtt{b}, \mathtt{c}\rangle$
- **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 $s = \langle s_1, \ldots, s_n\rangle \in A^n$ and $t = \langle t_1, \ldots, t_m\rangle \in A^m$ is defined as $\langle s_1, \ldots, s_n, t_1, \ldots, t_m\rangle \in A^{n+m}$.
  We will often write $\mathrm{conc}(s,t)$ as $s + t$ or simply st
- **Example 2.6.** $\mathrm{conc}("\mathtt{text}", "\mathtt{book}") = "\mathtt{text}" + "\mathtt{book}" = "\mathtt{textbook}"$

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 $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, then $A^* = \{\epsilon, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{aa}, \mathtt{ab}, \mathtt{ac}, \mathtt{ba}, \ldots, \mathtt{aaa}, \ldots\}$.
- **Definition 2.9.** A set $L \subseteq A^*$ is called a formal language over A.
- **Definition 2.10.** We use $\mathtt{c}^{[n]}$ for the string that consists of the character $\mathtt{c}$ repeated n times.
- **Example 2.11.** $\#^{[5]} = \langle \#, \#, \#, \#, \# \rangle$
- **Example 2.12.** The set $M := \{\mathtt{b}\mathtt{a}^{[n]} \mid n \in \mathbb{N}\}$ of strings that start with character $\mathtt{b}$ followed by an arbitrary numbers of $\mathtt{a}$'s is a formal language over $A = \{\mathtt{a}, \mathtt{b}\}$.
- **Definition 2.13 (Operations on Languages).** Let L, L_1, and L_2 be formal languages over the same alphabet, then we define language level operations: The concatenation of L_1 and L_2; $L_1L_2 := \{s_1s_2 \mid s_1 \in L_1 \wedge s_2 \in L_2\}$, $L^+ := \{s^+ \mid s \in L\}$, and $L^* := \{s^* \mid s \in L\}$.

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 (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).
- **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.

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, VP, $Article$, N, and Vi are nonterminals.

- **Definition 2.16.** The subset of lexical rules, i.e. those whose body consists of a single terminal is called its lexicon and the set of body symbols the vocabulary (or alphabet). The nonterminals in their heads are called lexical categories.
- **Definition 2.17.** The non-lexicon production rules are called structural, and the nonterminals in the heads are called phrasal categories.

Phrase Structure Grammars (Theory)

- **Idea:** Each symbol sequence in a formal language can be analyzed/generated by the grammar.
- **Definition 2.18.** 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 2.19.** 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.
- **Definition 2.20.** The language $\mathrm{L}(G)$ of G is the set of its sentences. **Definition 2.21.** We call two grammars equivalent, iff they have the same languages.
- **Definition 2.22.** 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.

Phrase Structure Grammars (Example)

- **Example 2.23.** 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.

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

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

Grammar Types (Chomsky Hierarchy [Cho65])

- **Observation:** The shape of the grammar determines the "size" of its language.
- **Definition 2.24.** We call a grammar and the formal language it accepts:
  1. context-sensitive, if the bodies of production rules have no less symbols than the heads,
  2. context-free, if the heads have exactly one symbol,
  3. regular, if additionally, bodies is empty or consists of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular, iff it is the language of a respective grammar. Context-free grammars are sometimes CFLs and context-free languages CFGs.

Grammar Types (Chomsky Hierarchy [Cho65])

- **Observation:** The shape of the grammar determines the "size" of its language.
- **Definition 2.26.** We call a grammar and the formal language it accepts:
  1. context-sensitive, if the bodies of production rules have no less symbols than the heads,
  2. context-free, if the heads have exactly one symbol,
  3. regular, if additionally, bodies is empty or consists of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular, iff it is the language of a respective grammar. Context-free grammars are sometimes CFLs and context-free languages CFGs.
- **Example 2.27 (Languages and their Grammars).**
  - Context-sensitive: The language $\{a^{[n]}b^{[n]}c^{[n]}\}$ is accepted by

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

Grammar Types (Chomsky Hierarchy [Cho65])

- **Observation:** The shape of the grammar determines the "size" of its language.
- **Definition 2.28.** We call a grammar and the formal language it accepts:
  1. context-sensitive, if the bodies of production rules have no less symbols than the heads,
  2. context-free, if the heads have exactly one symbol,
  3. regular, if additionally, bodies is empty or consists of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular, iff it is the language of a respective grammar. Context-free grammars are sometimes CFLs and context-free languages CFGs.
- **Example 2.29 (Languages and their Grammars).**
  - Context-sensitive: The language $\{a^{[n]}b^{[n]}c^{[n]}\}$
  - Context-free: The language $\{a^{[n]}b^{[n]}\}$ is accepted by $S \rightarrow \mathrm{a}\ S\ \mathrm{b} \mid \epsilon$.

Grammar Types (Chomsky Hierarchy [Cho65])

- **Observation:** The shape of the grammar determines the "size" of its language.
- **Definition 2.30.** We call a grammar and the formal language it accepts:
  1. context-sensitive, if the bodies of production rules have no less symbols than the heads,
  2. context-free, if the heads have exactly one symbol,
  3. regular, if additionally, bodies is empty or consists of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular, iff it is the language of a respective grammar. Context-free grammars are sometimes CFLs and context-free languages CFGs.
- **Example 2.31 (Languages and their Grammars).**
  - Context-sensitive: The language $\{a^{[n]}b^{[n]}c^{[n]}\}$
  - Context-free: The language $\{a^{[n]}b^{[n]}\}$
  - Regular: The language $\{a^{[n]}\}$ is accepted by $S \rightarrow S$ a

Grammar Types (Chomsky Hierarchy [Cho65])

- **Observation:** The shape of the grammar determines the "size" of its language.
- **Definition 2.32.** We call a grammar and the formal language it accepts:
  1. context-sensitive, if the bodies of production rules have no less symbols than the heads,
  2. context-free, if the heads have exactly one symbol,
  3. regular, if additionally, bodies is empty or consists of a nonterminal, optionally followed by a terminal symbol.

  By extension, a formal language L is called context-sensitive/context-free/regular, iff it is the language of a respective grammar. Context-free grammars are sometimes CFLs and context-free languages CFGs.
- **Example 2.33 (Languages and their Grammars).**
  - Context-sensitive: The language $\{a^{[n]}b^{[n]}c^{[n]}\}$
  - Context-free: The language $\{a^{[n]}b^{[n]}\}$
  - Regular: The language $\{a^{[n]}\}$
- **Observation:** Natural languages are probably context-sensitive but parsable in real time! (like languages low in the hierarchy)

Useful Extensions of Phrase Structure Grammars

- **Definition 2.34.** 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), and
  - grouping: $(s_1 ; \ldots ; s_n)$, useful e.g. for repetition.
- **Observation:** All of these can be eliminated, .e.g ($\leadsto$ 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 $Y \rightarrow Y\ s$.

An Grammar Notation for AI-1

- **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-1 we will only use context-free grammars (simpler, but problem still applies)
- **in AI-1:** 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	$\in$	Σ_k^p	
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

4.3 Mathematical Language Recap

Mathematical Structures

- **Observation:** Mathematicians often cast object classes as mathematical structures.
- We have just seen this: repeated here for convenience.
- **Definition 3.1.** A phrase structure 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).
- **Observation:** Even though we call production rules "pairs" above, they are also mathematical structures $\langle h, b\rangle$ with a funny notation $h{\rightarrow}b$.

Mathematical Structures in Programming

- Most programming languages have some way of creating "named structures". Referencing components is usually done via "dot notation"
- **Example 3.2 (Structs in C).**

```
// Create strutures grule grammar
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";
}
```

In AI-1 we use a mixture between Math and Programming Styles

- In AI-1 we use mathematical notation, ...
- **Definition 3.3.** A structure signature combines the components, their "types", and accessor names of a mathematical structure in a tabular overview.
- **Example 3.4.**

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

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

  Read N Set nonterminal symbols as "N is in set and is a nonterminal symbol". Here – and in the future – we will use Set for the class of sets $\rightsquigarrow$ "N is a set".
- I will try to give structure signatures where necessary.

Chapter 5
Rational Agents: a Unifying Framework for Artificial Intelligence

5.1 Introduction: Rationality in Artificial Intelligence

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*" [Hau85]	"*The formalization of mental faculties in terms of computational models*" [CM85]
Acting	"*The art of creating machines that perform actions requiring intelligence when performed by people*" [Kur90]	"*The branch of CS concerned with the automation of appropriate behavior in complex situations*" [LS93]

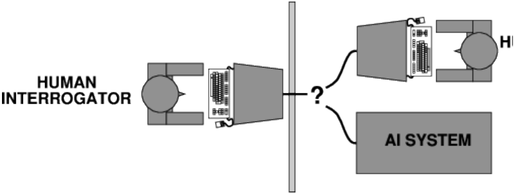
- **Idea:** Rationality is performance-oriented rather than based on imitation.

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:** $\leadsto$ 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)

Acting humanly: The Turing test

- Introduced by Alan Turing (1950) "Computing machinery and intelligence" [Tur50]:
- "Can machines think?" $\longrightarrow$ "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 [Tur50], 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!

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!

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?

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)

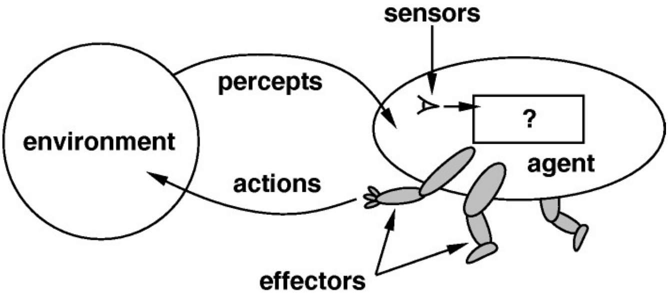
The Rational Agents

- **Definition 1.5.** An agent is an entity that perceives and acts.
- **Central Idea:** This course is about designing 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* $\leadsto$ design best program for given machine resources.

5.2 Agents and Environments as a Framework for AI

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).



- **Example 2.2.** Agents include humans, robots, softbots, thermostats, etc.

Modeling Agents Mathematically and Computationally

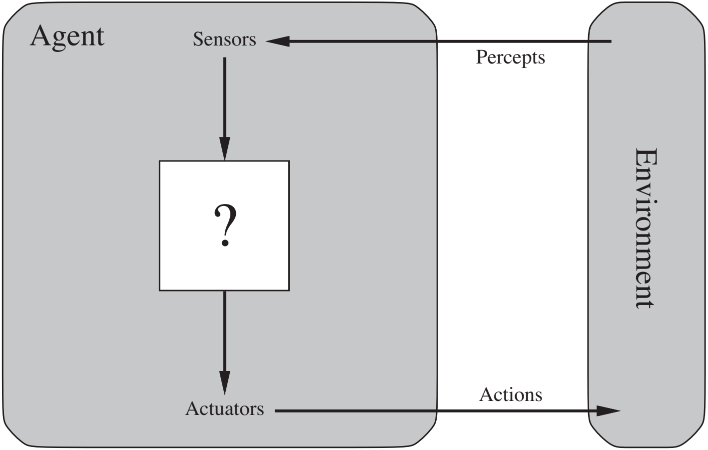
- **Definition 2.3.** A percept is the perceptual input of an agent at a specific time instant.
- **Definition 2.4.** Any recognizable, coherent employment of the actuators of an agent is called an action.
- **Definition 2.5.** The agent function f_a of an agent a maps from percept histories to actions:

$$f_a \colon \mathcal{P}^* \to \mathcal{A}$$

- We assume that agents can always perceive their own actions. (but not necessarily their consequences)
- **Problem:** agent functions can become very big (theoretical tool only)
- **Definition 2.6.** An agent function can be implemented by an agent program that runs on a physical agent architecture.

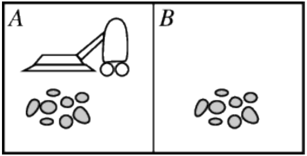
Agent Schema: Visualizing the Internal Agent Structure

- **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.

Example: Vacuum-Cleaner World and Agent



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*
$\vdots$	$\vdots$

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

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

Example: Vacuum-Cleaner World and Agent

- **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

Table-Driven Agents

- **Idea:** We can just implement the agent function as a table and look up 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?

5.3 Good Behavior $\rightsquigarrow$ Rationality

Rationality

- **Idea:** Try to design agents that are successful! (aka. “do the right thing”)
- **Definition 3.1.** A performance measure is a function that evaluates a sequence of environments.
- **Example 3.2.** A performance measure for the vacuum cleaner world 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.
- **Question:** Why is rationality a good quality to aim for?

Consequences of Rationality: Exploration, Learning, Autonomy

- **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)
- **Note:** rational $\rightsquigarrow$ exploration, learning, autonomy
- **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 has to learn all relevant traits, invariants, properties of the environment and actions.

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)

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.

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)

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?)

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

- **Observation 4.4.** *The real world is (of course) a partially observable, stochastic, sequential, dynamic, continuous, and multi agent environment.(worst case for AI)*

5.5 Types of Agents

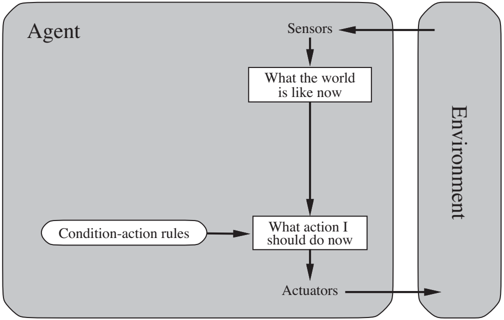
Agent types

- **Observation:** So far we have described (and analyzed) agents only by their behavior (cf. agent function $f : \mathcal{P}^* \to \mathcal{A}$).
- **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. simple reflex agents
  2. model-based agents
  3. goal-based agents
  4. utility-based agents

  All these can be turned into learning agents.

Simple reflex agents

- **Definition 5.1.** A simple reflex agent is an agent a that only bases its actions on the last percept: so the agent function simplifies to $f_a\colon \mathcal{P}\to\mathcal{A}$.
- **Agent Schema:**



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

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

Simple reflex agents (continued)

- **General Agent Program:**

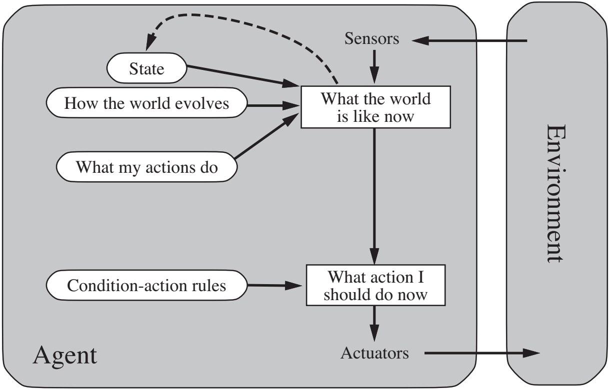
```
function Simple−Reflex−Agent (percept) returns an action
  persistent: rules /∗ a set of condition−action rules∗/

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

- **Problem:** Simple 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 simple reflex agent would have to compare subsequent percepts to realize.
- **Problem:** Partially observable environments get simple reflex agents into trouble.
- **Example 5.4.** Vacuum cleaner robot with defective location sensor $\leadsto$ infinite loops.

Model-based Reflex Agents: Idea

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



Model-based Reflex Agents: Definition

- **Definition 5.5.** A model-based agent is an agent whose actions depend on
  - a world model: a set $\mathcal{S}$ of possible states.
  - a sensor model S that given a state s and a percepts p determines a new state $S(s,p)$.
  - a transition model T, that predicts a new state $T(s,a)$ from a state s and an action a.
  - An action function f that maps (new) states to an actions.

  If the world model of a model-based agent A is in state s and A has taken action a, A will transition to state $s' = T(S(p,s),a)$ and take action $a' = f(s')$.
- **Note:** As different percept sequences lead to different states, so the agent function $f_a \colon \mathcal{P}^* \rightarrow \mathcal{A}$ no longer depends only on the last percept.
- **Example 5.6 (Tail Lights Again).** Model-based agents can do the 101 if the states include a concept of tail light brightness.

Model-Based Agents (continued)

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

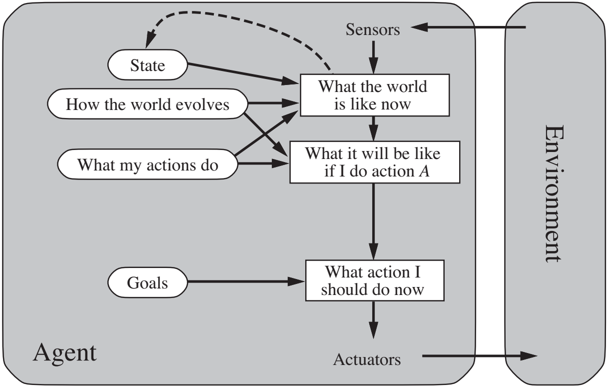
```
function Model−Based−Agent (percept) returns an action
  var state /* a description of the current state of the world */
  persistent rules /* a set of condition−action rules */
  var action /* the most recent action, initially none */

  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.

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:**



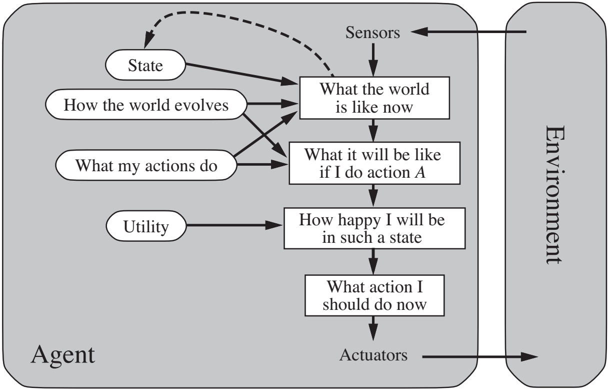
Goal-based agents (continued)

- **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 goal function f that given a (new) state s' selects an action a to best reach $\mathcal{G}$.

  The action 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.

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:**



Utility-based vs. Goal-based Agents

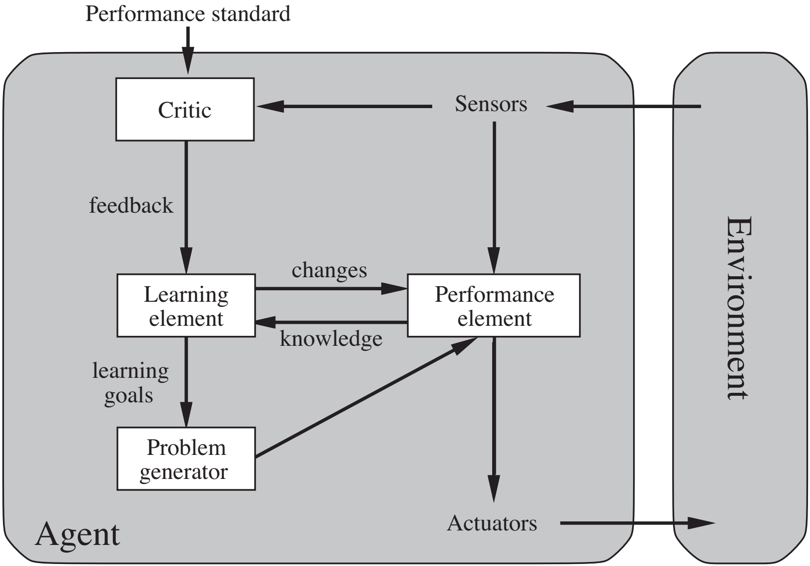
- **Question:** What is the difference between goal-based and utility-based agents?
- **Utility-based Agents are a Generalization:** We can always force goal-directedness by a utility function that only rewards goal states.
- **Goal-based Agents can do less:** A utility function allows rational decisions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

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.

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.

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?

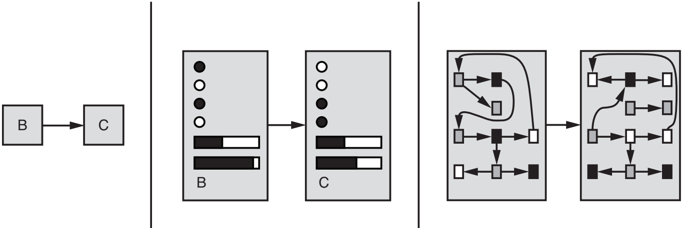
5.6 Representing the Environment in Agents

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 wold model.
- **Definition 6.1.** 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 and their relationships.

Atomic/Factored/Structured State Representations

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

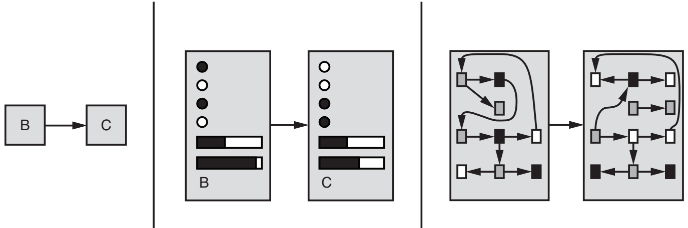


(a) Atomic | (b) Factored | (b) Structured

- **Example 6.2.** 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.

Atomic/Factored/Structured State Representations

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

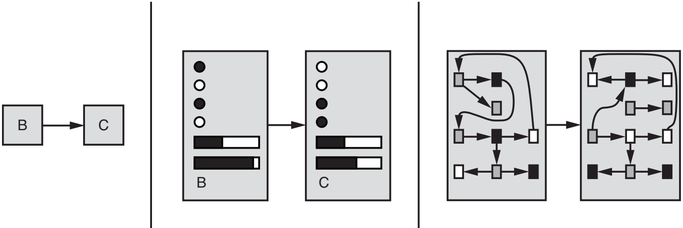


(a) Atomic (b) Factored (b) Structured

- **Example 6.3.** 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)

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.
  - 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.

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

Part 2
General Problem Solving

Chapter 6
Problem Solving and Search

6.1 Problem Solving

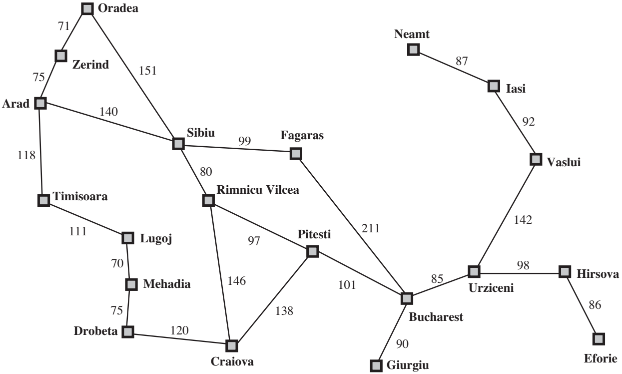
Problem Solving: Introduction

- **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)

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

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.

The Math of Problem Formulation: Search Problems

- **Definition 1.5.** 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 (also called terminal state) ($\mathcal{G}\subseteq\mathcal{S}$) 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 $\mathcal{T}(a,s)\neq\emptyset$. 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 a search problem $\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$ consists of a sequence $a_1,\ldots,a_n$ of actions such that for all $1{\leq}i{<}n$
  - a_i is applicable to state $s_{(i-1)}$, where $s_0\in\mathcal{I}$,
  - $s_i\in\mathcal{T}_{a_i}(s_{(i-1)})$, and $s_n\in\mathcal{G}$.
- **Idea:** A solution bring us from $\mathcal{I}$ to a goal state.
- **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.

Structure Overview: Search Problem

- The structure overview for search problems:

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

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 with transition model $\mathcal{T}$ deterministic, iff $|\mathcal{T}(a,s)| \leq 1$.
-
- **Definition 1.12.** In a deterministic search problem, $\mathcal{T}_a$ induces partial function $S_a \colon \mathcal{S} \rightharpoonup \mathcal{S}$ whose natural domain is the set of states where a is applicable: $S_a(s) := s'$ if $\mathcal{T}_a = \{s'\}$ and undefined at s otherwise. We call S_a the successor function for a and $S_a(s)$ the successor state of s.
- **Definition 1.13.** The predicate that tests for goal states is called a goal test.

Blackbox/Declarative Problem Descriptions

- **Observation:** $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ from 1.5 is essentially a blackbox description; it (think programming API)
  - provides the functionality needed to construct a state space, but
  - gives the algorithm no information about the problem.
- **Definition 1.14.** A declarative description (also called whitebox description) describes the problem itself $\leadsto$ problem description language
- **Example 1.15 (Planning Problems as Declarative Descriptions).** The STRIPS language describes planning problems in terms of
  - a set P of propositional variables (propositions)
  - a set $I \subseteq P$ of propositions true in the initial state.
  - a set $G \subseteq P$, where state $s \subseteq P$ is a goal state if $G \subseteq s$
  - a set A of actions, each $a \in A$ with preconditions pre_a, add list add_a, and delete list del_a: a is applicable, if $\mathrm{pre}_a \subseteq s$, the result state is then $(s \cup \mathrm{add}_a) \backslash \mathrm{del}_a$,
  - a function c that maps all actions a to their cost $c(a)$.
- **Observation 1.16.** *Declarative descriptions are strictly more powerful than blackbox descriptions: they induce blackbox descriptions, but also allow to analyze/simplify the problem.*
- We will come back to this later $\leadsto$ planning.

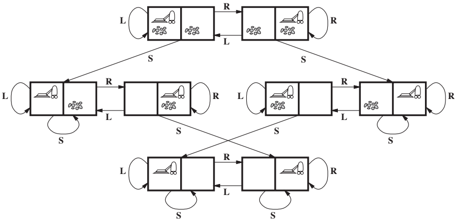
6.2 Problem Types

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)

Example: vacuum-cleaner world

- **Single-state Problem:**



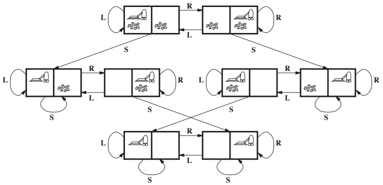
  - Start in 5
  - **Solution**: [*right*, *suck*]

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

right	$\rightarrow$	$\{2, 4, 6, 8\}$
suck	$\rightarrow$	$\{4, 8\}$
left	$\rightarrow$	$\{3, 7\}$
suck	$\rightarrow$	$\{7\}$

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**: $[\mathit{suck},\mathit{right},\mathit{suck}]$
    suck $\rightarrow \{5,7\}$
    right $\rightarrow \{6,8\}$
    suck $\rightarrow \{6,8\}$



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

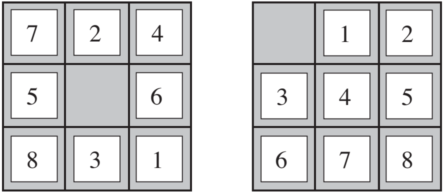
Single-state problem formulation

- 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.

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:** $Arad \rightarrow Zerind$ represents complex set of possible routes.
- **(Abstract) solution:** Set of real paths that are solutions in the real world.

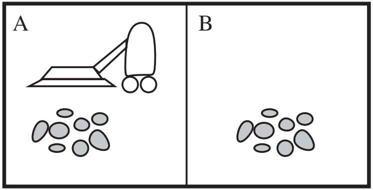
Example: The 8-puzzle



Start State Goal State

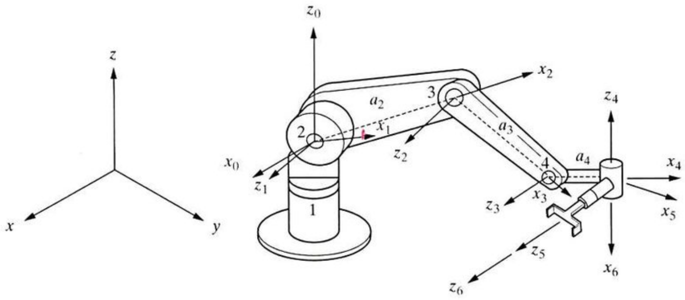
States	integer locations of tiles
Actions	*left*, *right*, *up*, *down*
Goal test	= goal state?
Path cost	1 per move

Example: Vacuum-cleaner



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

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

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.

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.

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).

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

6.3 Search

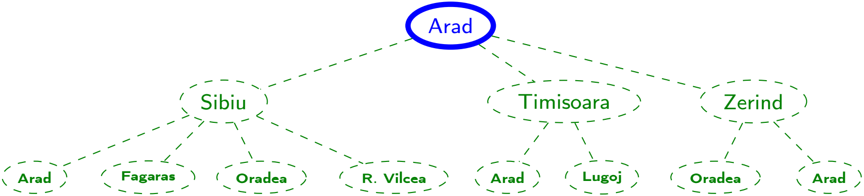
Tree Search Algorithms

- **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)

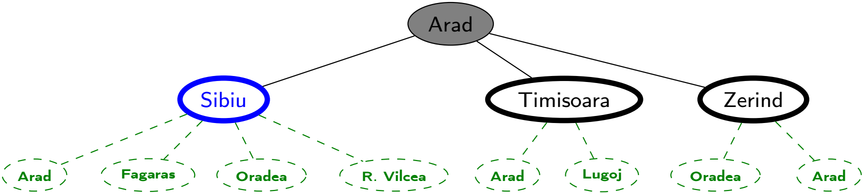
```
procedure Tree−Search (problem, strategy) : <a solution or failure>
  <initialize the search tree using the initial state of problem>
  loop
     if <there are no candidates for expansion> <return failure> end if
    <choose a leaf node for expansion according to strategy>
     if <the node contains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

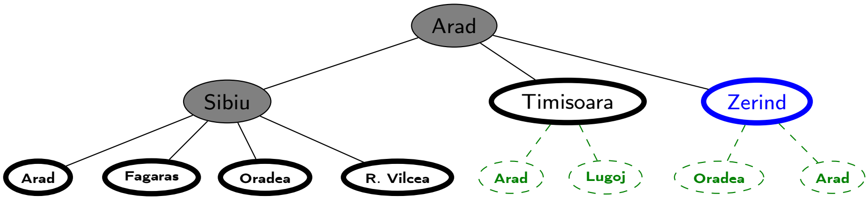
Tree Search: Example



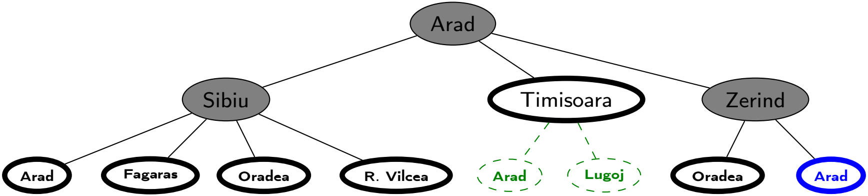
Tree Search: Example



Tree Search: Example

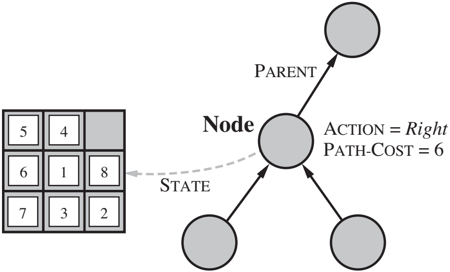


Tree Search: Example



Implementation: States vs. nodes

- **Recap:** A state is a (representation of) a physical configuration.
- **Remark:** The nodes of a search tree are implemented as a data structure that includes accessors for parent, children, depth, path cost, etc.



- **Observation:** Paths in the search tree correspond to paths in the state space.
- **Observation:** As a search tree node has access to parents, we can read off the solution from a goal node.
- **Definition 3.2.** A goal node is a node labeled with a goal state
- **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.

Implementation of Search Algorithms

```
procedure Tree_Search (problem,strategy)
  fringe := insert(make_node(initial_state(problem)))
    loop
     if fringe <is empty> fail end if
      node := first(fringe,strategy)
      if NodeTest(State(node)) return State(node)
      else fringe := insert_all(expand(node,problem),strategy)
      end if
   end loop
end procedure
```

- **Definition 3.4.** The fringe is a list nodes not yet expanded in tree search.
- It is ordered by the strategy. (see below)

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 means here always *worst-case* complexity!

6.4 Uninformed Search Strategies

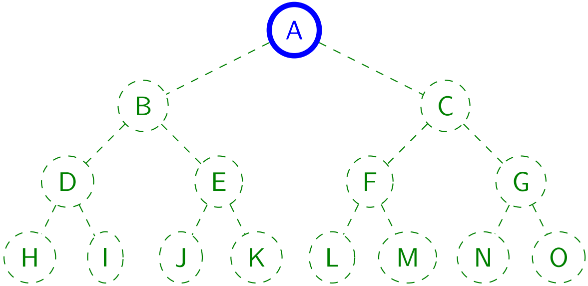
Uninformed search strategies

- **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

6.4.1 Breadth-First Search Strategies

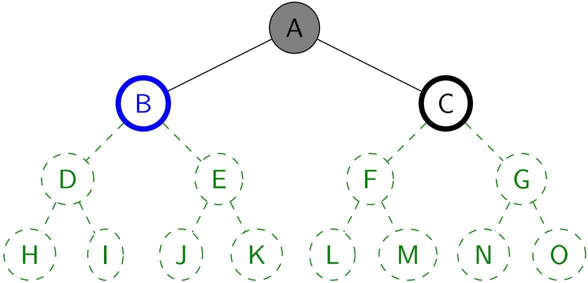
Breadth-First Search

- **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).**



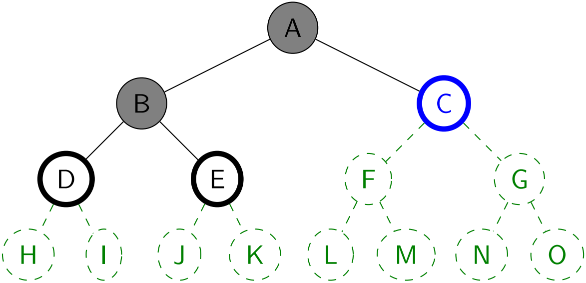
Breadth-First Search

- **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).**



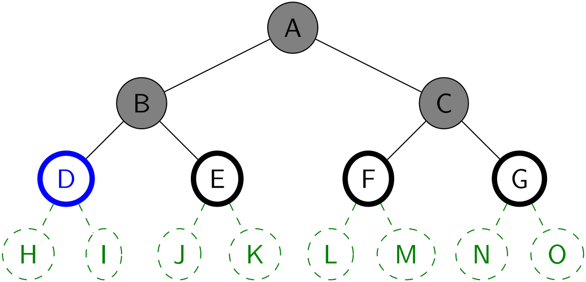
Breadth-First Search

- **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).**



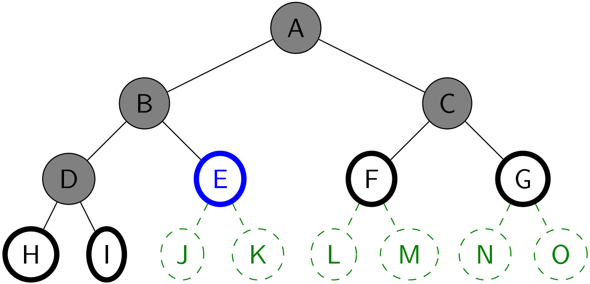
Breadth-First Search

- **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).**



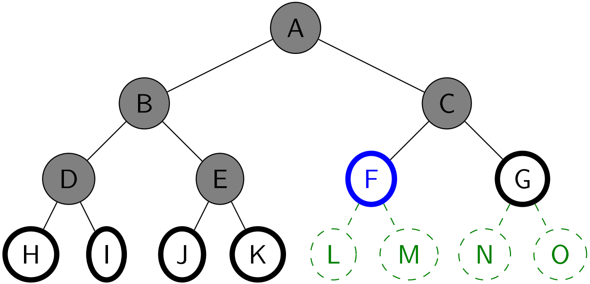
Breadth-First Search

- **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).**



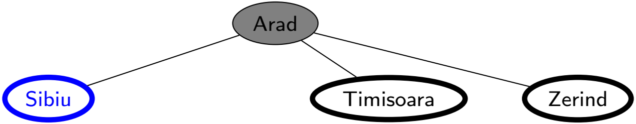
Breadth-First Search

- **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).**

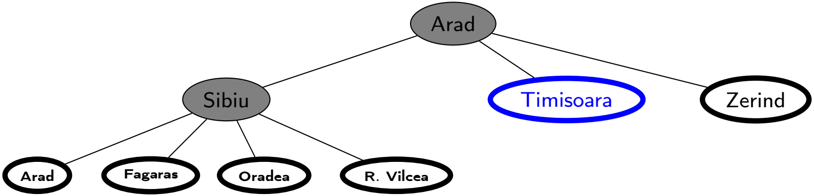


Breadth-First Search: Romania

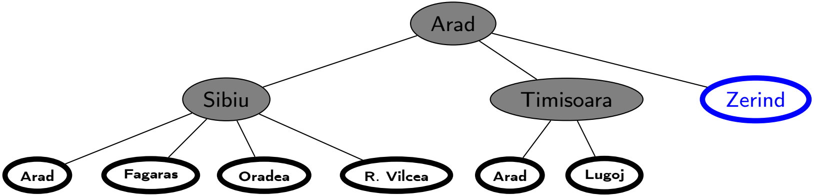
Breadth-First Search: Romania



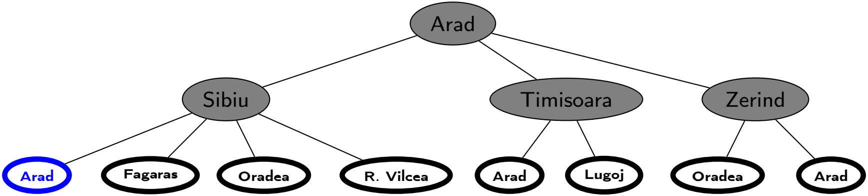
Breadth-First Search: Romania



Breadth-First Search: Romania



Breadth-First Search: Romania

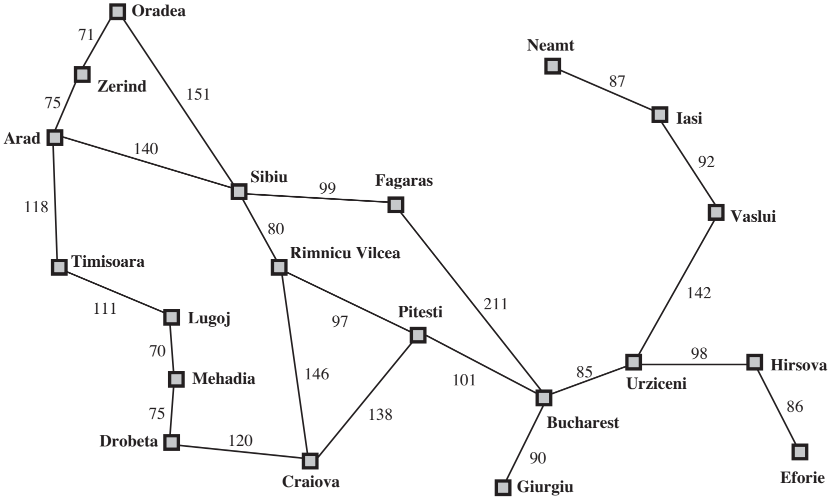


Breadth-first search: Properties

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

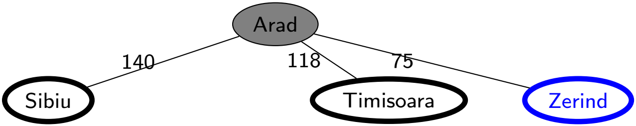


Uniform-cost search

- **Idea:** Expand least cost unexpanded node.
- **Definition 4.14.** 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:**

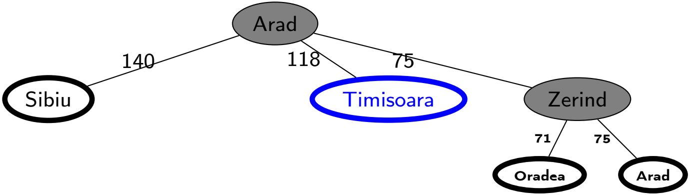
Uniform-cost search

- **Idea:** Expand least cost unexpanded node.
- **Definition 4.15.** 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:**



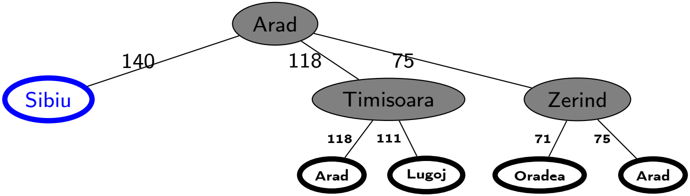
Uniform-cost search

- **Idea:** Expand least cost unexpanded node.
- **Definition 4.16.** 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:**



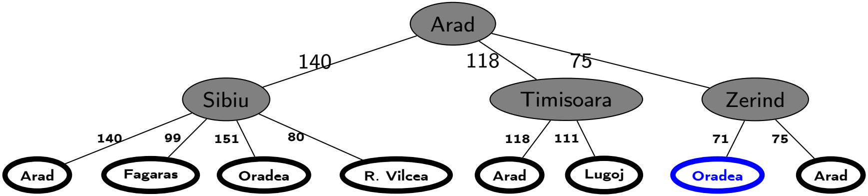
Uniform-cost search

- **Idea:** Expand least cost unexpanded node.
- **Definition 4.17.** 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 4.18.** 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: 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	dito
Optimality	Yes

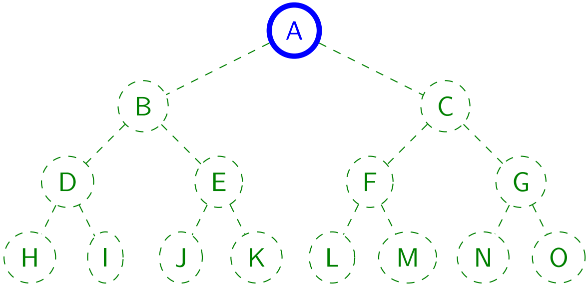
6.4.2 Depth-First Search Strategies

Depth-first Search

- **Idea:** Expand deepest unexpanded node.
- **Definition 4.19.** 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 4.20.** 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)

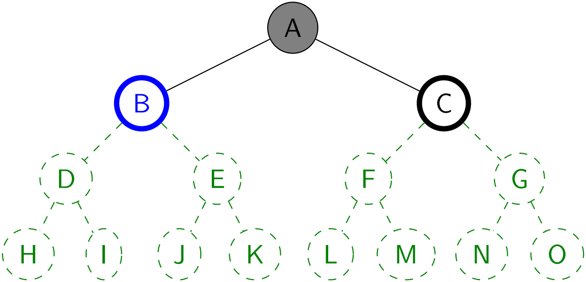
Depth-First Search

- **Example 4.21 (Synthetic).**



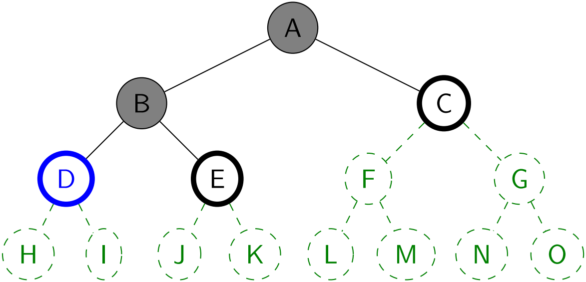
Depth-First Search

- **Example 4.22 (Synthetic).**



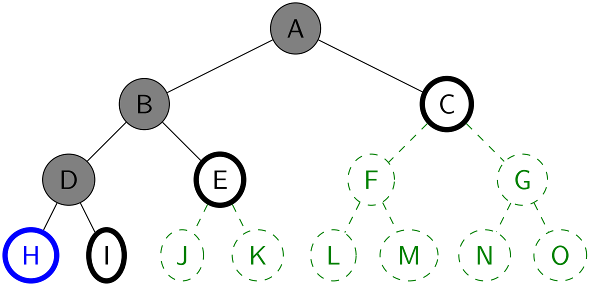
Depth-First Search

- **Example 4.23 (Synthetic).**



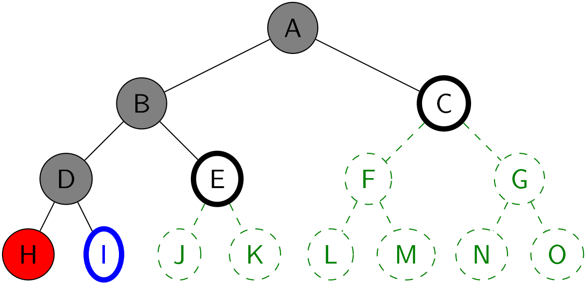
Depth-First Search

▶ **Example 4.24 (Synthetic).**



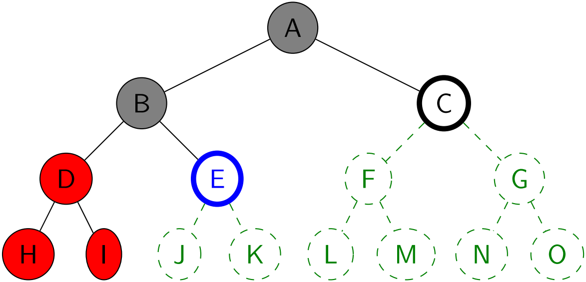
Depth-First Search

- **Example 4.25 (Synthetic).**



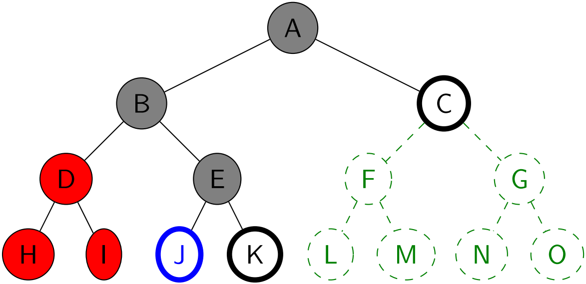
Depth-First Search

▶ **Example 4.26 (Synthetic).**



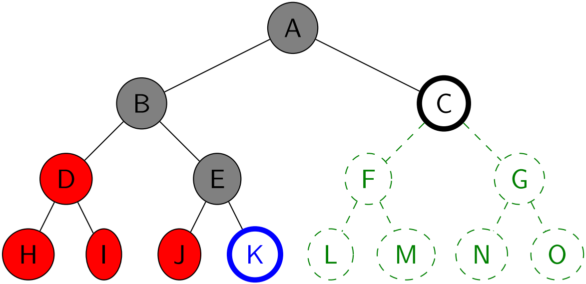
Depth-First Search

- **Example 4.27 (Synthetic).**



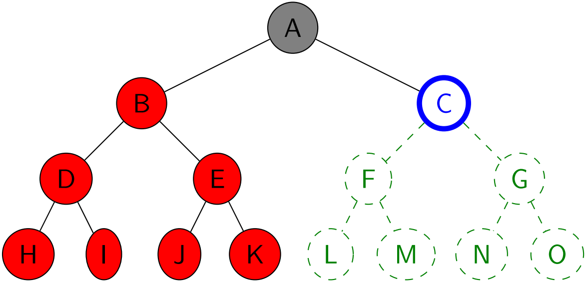
Depth-First Search

- **Example 4.28 (Synthetic).**



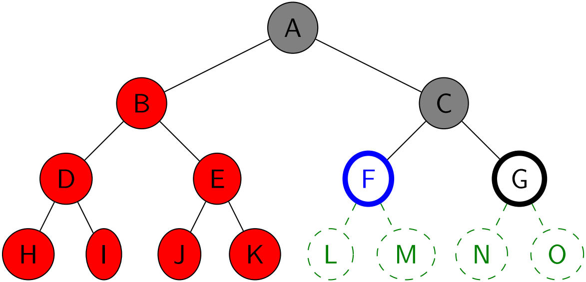
Depth-First Search

- **Example 4.29 (Synthetic).**



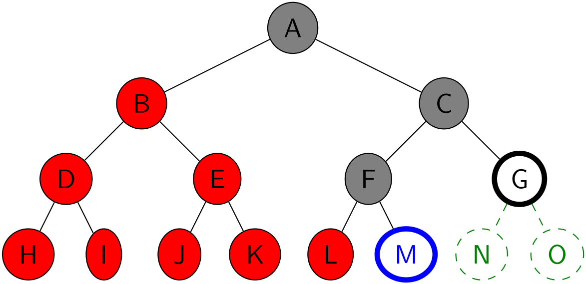
Depth-First Search

- **Example 4.30 (Synthetic).**



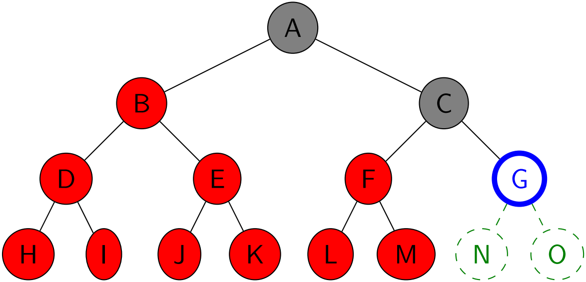
Depth-First Search

- **Example 4.31 (Synthetic).**



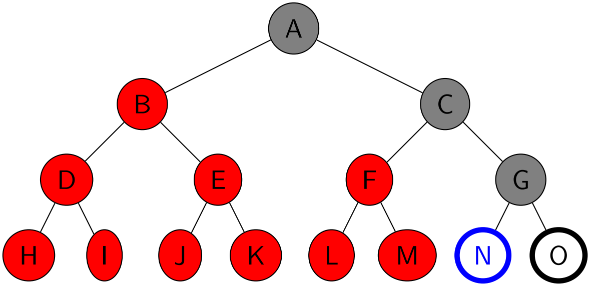
Depth-First Search

▶ **Example 4.32 (Synthetic).**



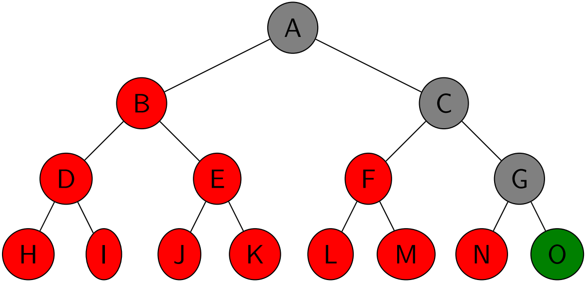
Depth-First Search

- **Example 4.33 (Synthetic).**



Depth-First Search

- **Example 4.34 (Synthetic).**



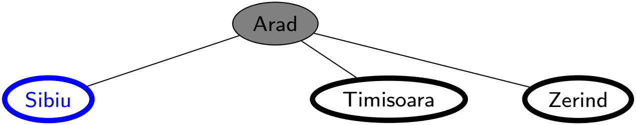
Depth-First Search: Romania

- **Example 4.35 (Romania).**

Arad

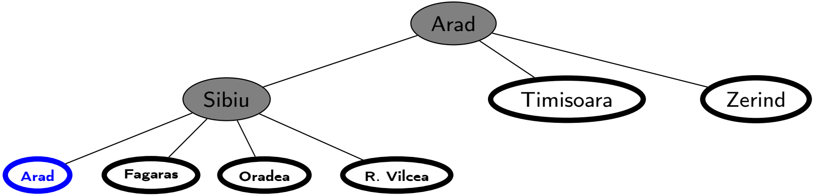
Depth-First Search: Romania

- **Example 4.36 (Romania).**



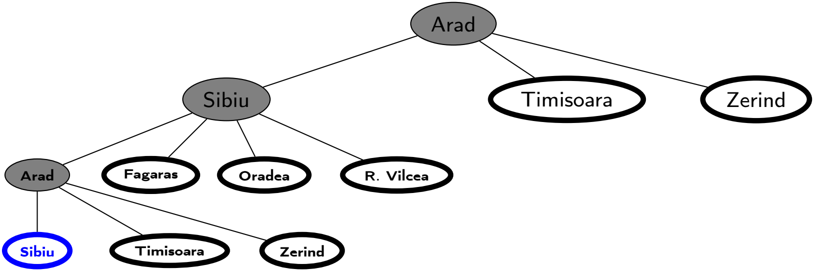
Depth-First Search: Romania

▶ **Example 4.37 (Romania).**



Depth-First Search: Romania

- **Example 4.38 (Romania).**



Depth-first search: Properties

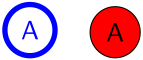
Completeness	Yes: if state space finite No: if search tree contains infinite paths or loops
Time complexity	$\mathcal{O}(b^m)$ (we need to explore until max depth m in any case!)
Space complexity	$\mathcal{O}(bm)$ (i.e. linear space) (need at most store m levels and at each level at most b nodes)
Optimality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

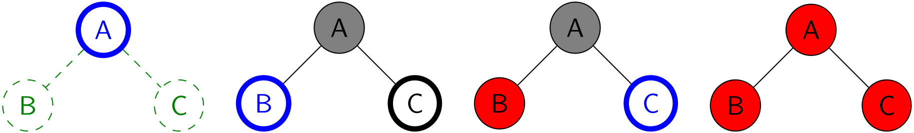
Iterative deepening search

- **Definition 4.39.** Depth limited search is depth first search with a depth limit.
- **Definition 4.40.** Iterative deepening search (IDS) is depth limited search with ever increasing depth limits.
- **procedure** Tree_Search (problem)
  <initialize the search tree using the initial state of problem>
  for depth = 0 **to** ∞
  result := Depth_Limited_search(problem,depth)
  if depth $\neq$ cutoff **return** result **end if**
  end for
  end procedure

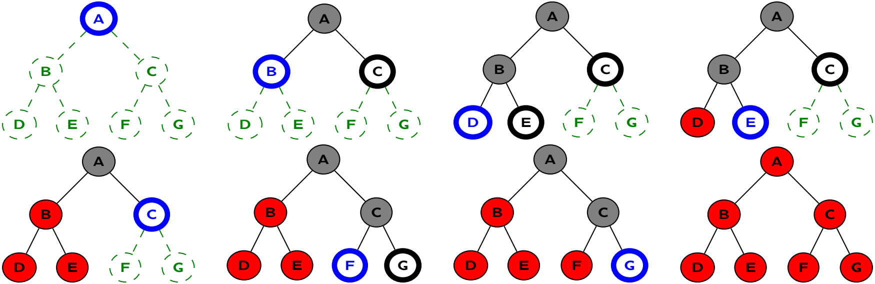
Ilustration: Iterative Deepening Search at various Limit Depths



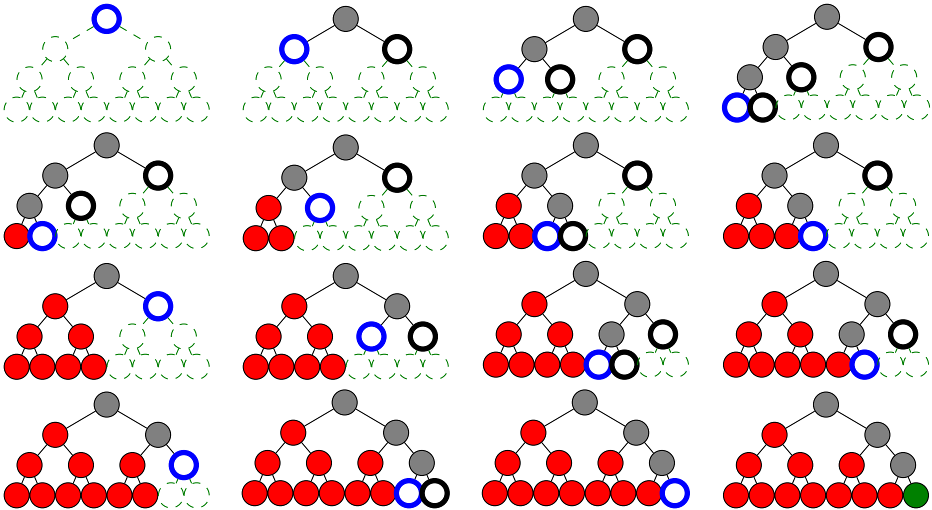
Ilustration: Iterative Deepening Search at various Limit Depths



Ilustration: Iterative Deepening Search at various Limit Depths



Ilustration: Iterative Deepening Search at various Limit Depths



Iterative deepening search: Properties

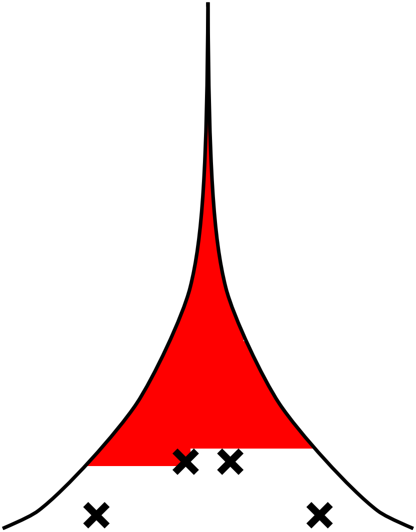
Completeness	Yes
Time complexity	$(d+1)\cdot b^0+d\cdot b^1+(d-1)\cdot b^2+\ldots+b^d\in\mathcal{O}(b^{d+1})$
Space complexity	$\mathcal{O}(b\cdot d)$
Optimality	Yes (if step cost $=1$)

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

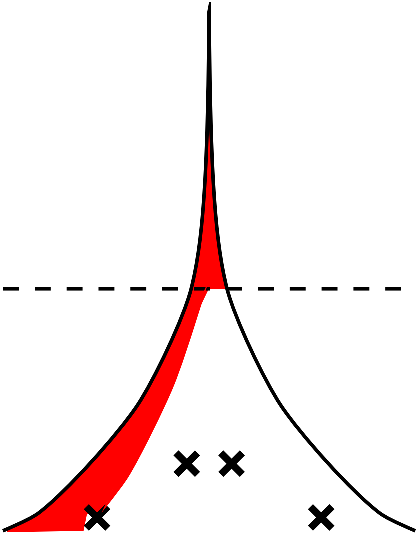
Comparison BFS (optimal) and IDS (not)

- **Example 4.41.** IDS may fail to be be optimal at step sizes > 1.

Breadth first search



Iterative deepening search



6.4.3 Further Topics

Tree Search vs. Graph Search

- We have only covered tree search algorithms.
- States duplicated in nodes are a huge problem for efficiency.
- **Definition 4.42.** 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 4.43.** *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 4.44.** 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)

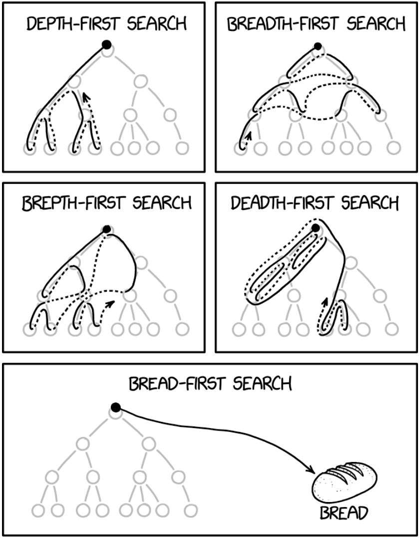
Uninformed Search Summary

- **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] $0<\epsilon\leq\mathrm{cost}$		

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.

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

Greedy search

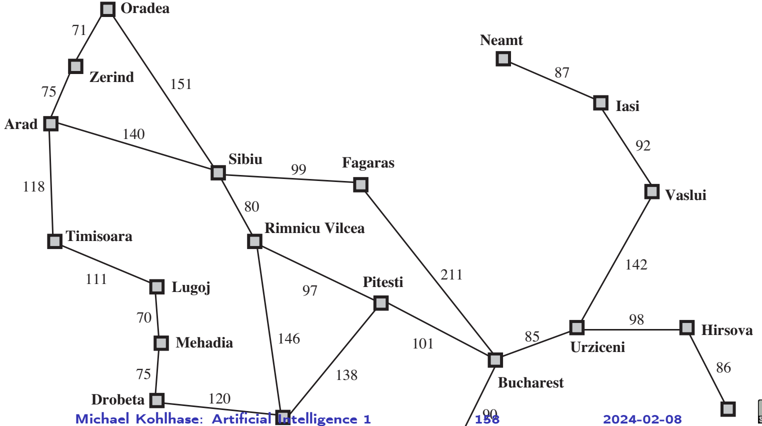
- **Idea:** Expand the node that *appears* to be closest to the goal.
- **Definition 5.4.** A heuristic is an evaluation function h on states that estimates the cost from n to the nearest goal state. We speak of heuristic search if the search algorithm uses a heuristic in some way.
- **Note:** All nodes 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

$$\text{internal search control} \rightsquigarrow \text{external search control}$$
$$\text{partial solution cost} \rightsquigarrow \text{goal cost estimation}$$

Romania with Straight-Line Distances

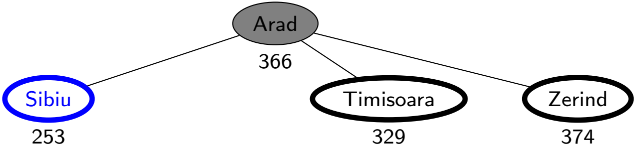
- **Example 5.7 (Informed Travel).**
  $h_{\mathrm{SLD}}(n) = \text{straight} - \text{line distance to Bucharest}$

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

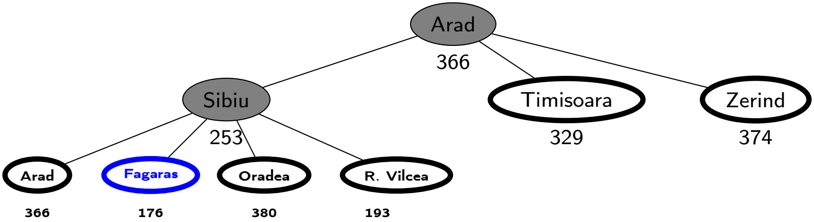


Greedy Search: Romania

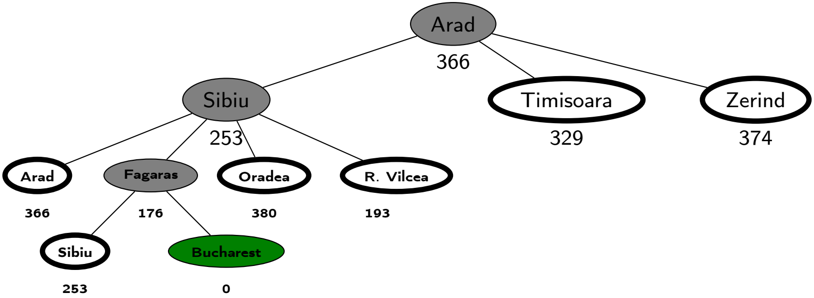
Greedy Search: Romania



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

I

	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

G

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

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

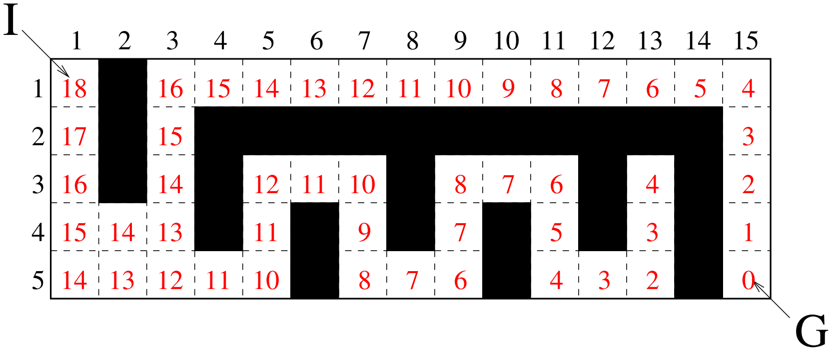
I

	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

G

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



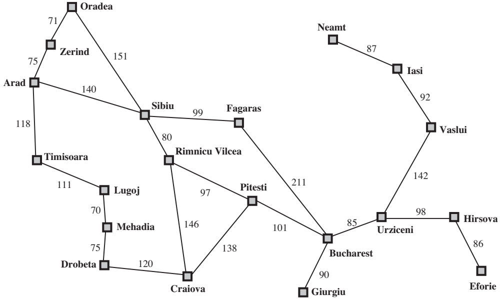
Greedy search: Properties

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

Greedy search: Properties

Completeness	No: Can get stuck in loops Complete in finite space 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 $\to$ Neamt $\to$ Iasi $\to$ Neamt $\to \cdots$



Greedy search: Properties

Completeness	No: Can get stuck in loops Complete in finite space 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 $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow \cdots$
- **Worst-case Time:** Same as depth first search.
- **Worst-case Space:** Same as breadth first search.
- **But:** A good heuristic can give dramatic improvements.

6.5.2 Heuristics and their Properties

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).

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" [Pól73]
- same word often used for "rule of thumb" or "imprecise solution method".

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.

Properties of Heuristic Functions, ctd.

- Let Π be a 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.
  2. step case
     2.1. We assume that $h(s') \leq h^*(s)$ for all states s' with a cheapest goal node path of length n.
     2.2. Let s be a state whose cheapest goal path has length $n+1$ and the first transition is $o = (s,s')$.
     2.3. By consistency, we have $h(s) - h(s') \leq c(o)$ and thus $h(s) \leq h(s') + c(o)$.
     2.4. By construction, $h^*(s)$ has a cheapest goal path of length n and thus, by induction hypothesis $h(s') \leq h^*(s')$.
     2.5. By construction, $h^*(s) = h^*(s') + c(o)$.
     2.6. Together this gives us $h(s) \leq h^*(s)$ as desired.
- Consistency is a sufficient condition for admissibility (easier to check)

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)

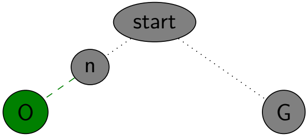
6.5.3 A-Star Search

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.

A^* Search: Optimality

- **Theorem 5.29.** *A^* search with admissible heuristic is optimal.*
- *Proof:* We show that sub-optimal 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

$f(G) = g(G)$	since $h(G) = 0$
$g(G) > g(O)$	since G suboptimal
$g(O) = g(n) + h^*(n)$	n on optimal path
$g(n) + h^*(n) \geq g(n) + h(n)$	since h is admissible
$g(n) + h(n) = f(n)$	

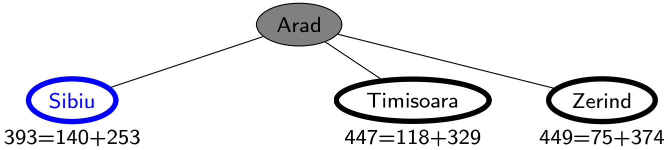
  3. Thus, $f(G) > f(n)$ and A^* never expands G.

A^* Search Example

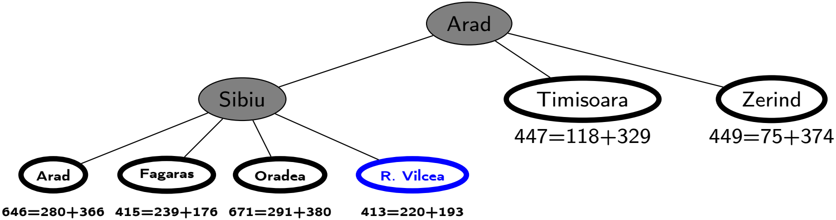
Arad

366=0+366

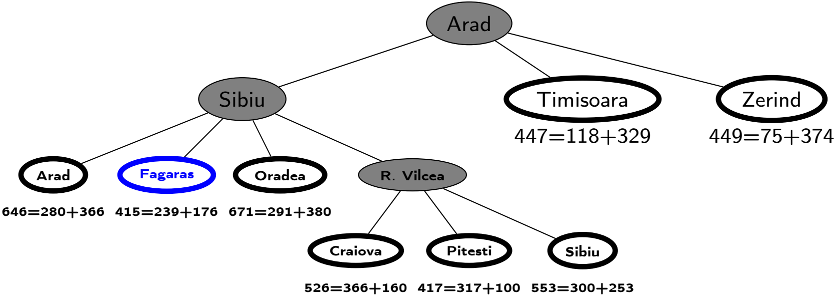
A^* Search Example



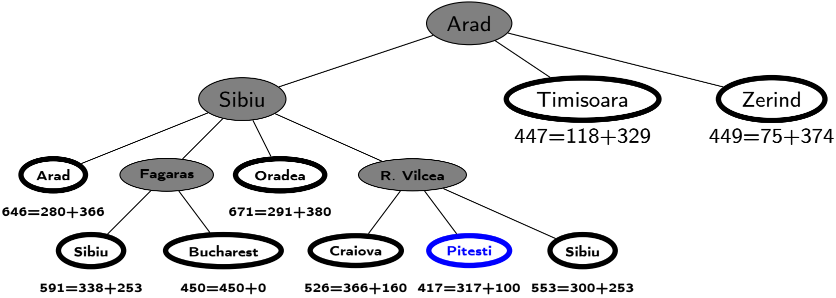
A^* Search Example



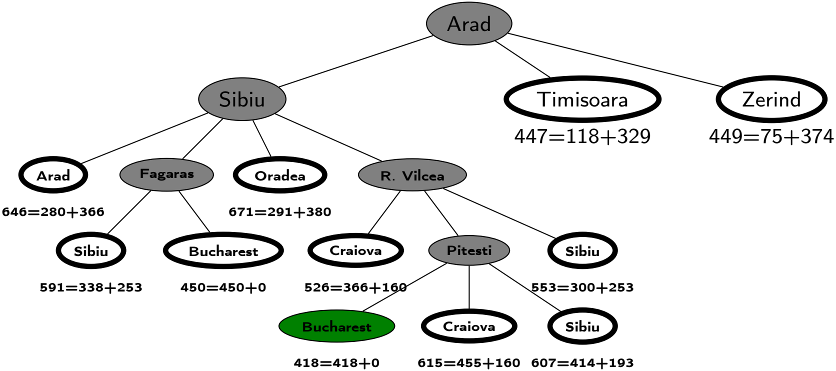
A^* Search Example



A^* Search Example

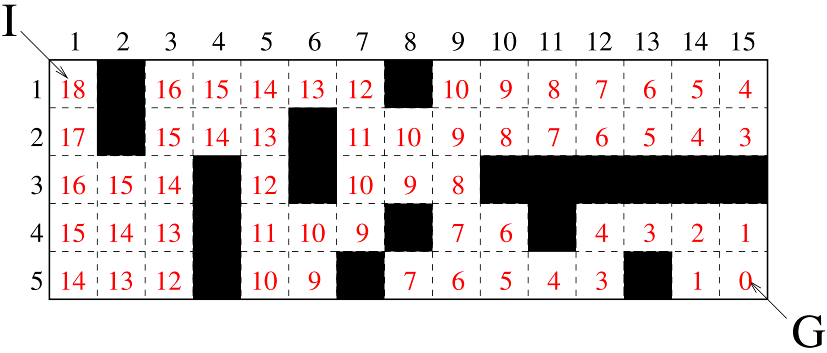


A^* Search Example



Additional Observations (Not Limited to Path Planning)

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



We will find a solution with little search.

Additional Observations (Not Limited to Path Planning)

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

I

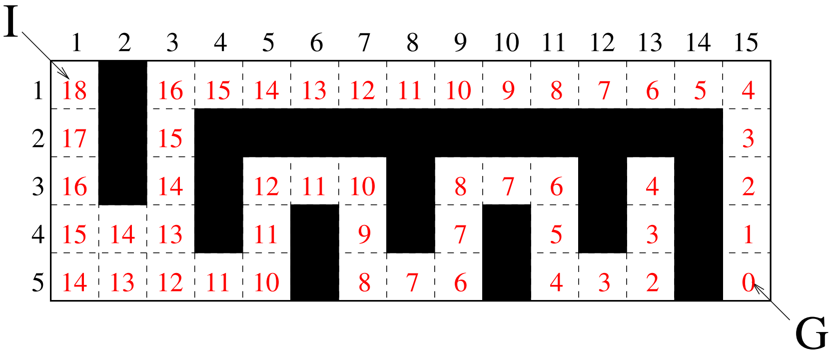
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	■	22	22	22	22	22	■	24	24	24	24	24	24	24
2	18	■	20	20	20	■	22	22	22	22	22	22	22	22	22
3	18	18	18	■	20	■	22	22	22	■	■	■	■	■	■
4	18	18	18	■	20	20	20	■	22	22	■	24	24	24	24
5	18	18	18	■	20	20	■	24	22	22	22	22	■	24	24

G

  - 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^*.

Additional Observations (Not Limited to Path Planning)

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



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

Additional Observations (Not Limited to Path Planning)

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

I

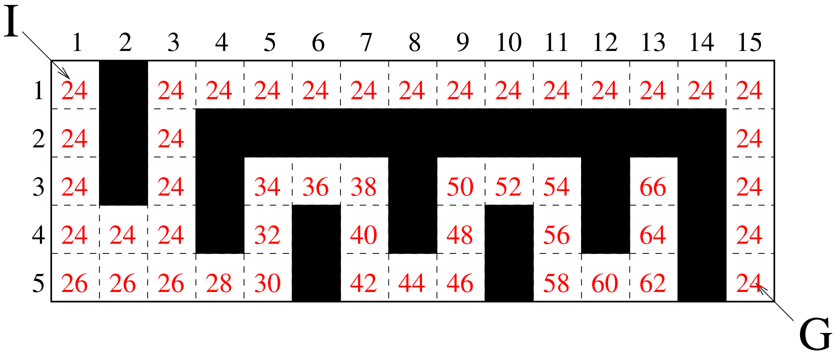
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18		24	24	24	24	24	24	24	24	24	24	24	24	24
2	18		22												24
3	18		20		22	22	22		26	26	26		30		24
4	18	18	18		20		22		24		26		28		24
5	18	18	18	18	18		22	22	22		26	26	26		24

G

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)

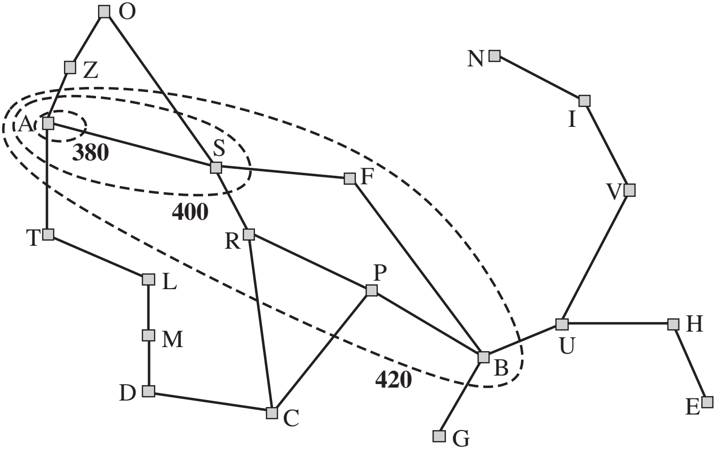
- **Example 5.34 (A^* ($g+h$) using h^*).**



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

- A^* gradually adds "f-contours" of nodes



A^* search: Properties

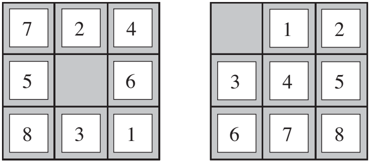
- Properties or A^*

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

- A^* 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.

6.5.4 Finding Good Heuristics

Admissible heuristics: Example 8-puzzle



Start State Goal State

- **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*

Dominance

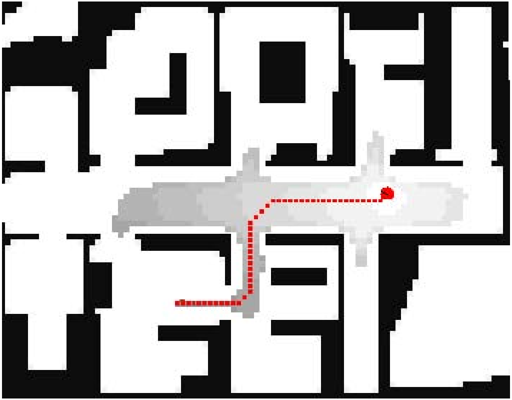
- **Definition 5.38.** Let h_1 and h_2 be two admissible heuristics we say that h_2 dominates h_1 if $h_2(n) \geq h_1(n)$ for all n.
- **Theorem 5.39.** *If h_2 dominates h_1, then h_2 is better for search than h_1.*

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.

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!

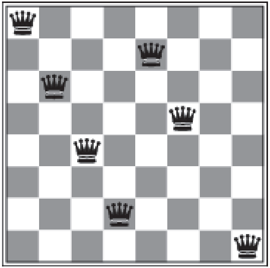
6.6 Local Search

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.

Local Search Problems

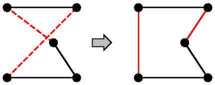
- **Idea:** Sometimes the path to the solution is irrelevant.
- **Example 6.5 (8 Queens Problem).** Place 8 queens on a chess board, so that no two queens threaten each other.



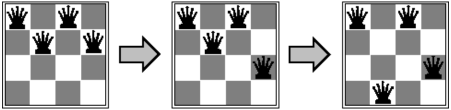
- This problem has various solutions (the one of the right isn't one of them)
- **Definition 6.6.** A local search algorithm is a search algorithm that operates on a single state, the current state (rather than multiple paths). (advantage: constant space)
- Typically local search algorithms only move 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,. . .

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



Hill-climbing (gradient ascent/descent)

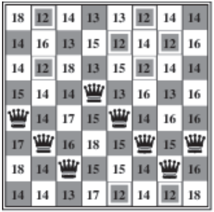
- **Idea:** Start anywhere and go in the direction of the steepest ascent.
- **Definition 6.9.** Hill climbing (also gradient ascent) is a local search algorithm that iteratively selects the best successor:

```
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
```

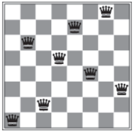
- **Intuition:** Like best first search without memory.
- Works, if solutions are dense and local maxima can be escaped.

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:

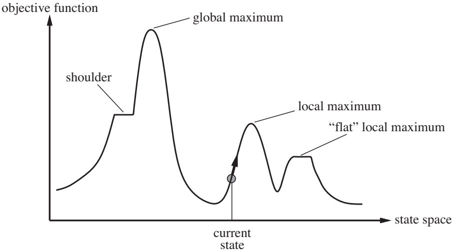


- **Problem:** The state space has local minima. e.g. the board on the right has $h = 1$ but every successor has $h > 1$.



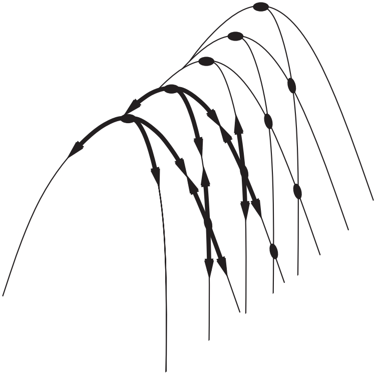
Hill-climbing

- **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)

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 cristal 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 [Met+53]
- Widely used in VLSI layout, airline scheduling, etc.

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^(Δ(E)/T)
     end if
  end for
end procedure
```

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

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 $\rightsquigarrow$ 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?

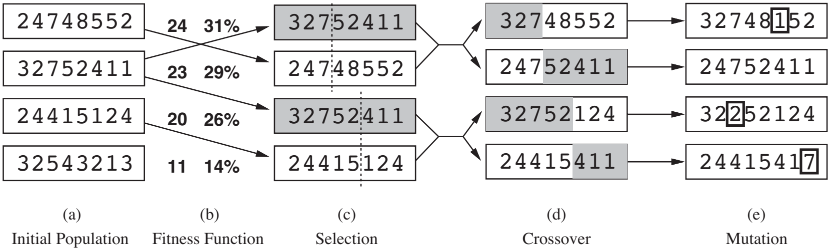
Local beam search

- **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!)

Genetic algorithms (very briefly)

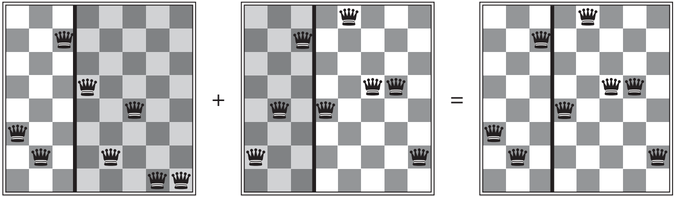
- **Definition 6.15.** A genetic algorithm is a variant of local beam search that generates successors by
  - randomly modifying states (mutation)
  - mixing pairs of states (sexual reproduction or crossover)

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



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

7.1 Introduction

The Problem

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

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

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)

"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).

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.

"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.

(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 $u \colon \mathcal{G} \to \mathbb{R}$. (the "score" of the game)
- **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)

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 a strategy is infeasible.
- **Solution:** Compute the next move "on demand", given the current state instead.

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: $200^{300} \simeq 10^{690}$.

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.

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.

Our Agenda for This Chapter

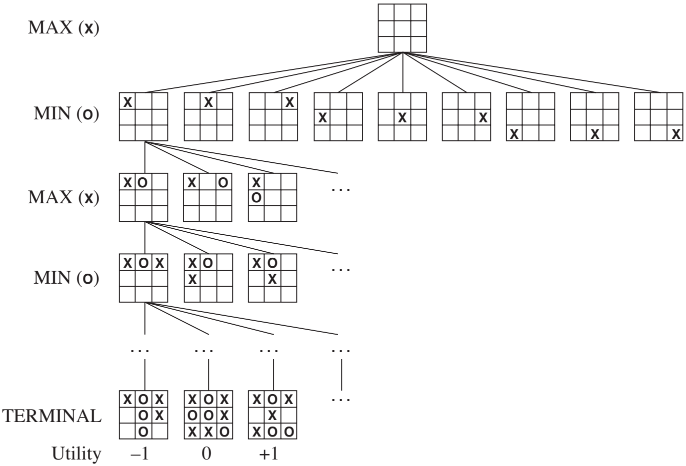
- **Minimax Search:** How to compute an optimal strategy?
  - Minimax is the canonical (and easiest to understand) algorithm for *solving* games, i.e., computing an optimal strategy.
- **Evaluation functions:** But what if we don't have the time/memory to solve the entire game?
  - Given limited time, 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).

7.2 Minimax Search

"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 $\leadsto$ hence "minimax".

Example Tic-Tac-Toe

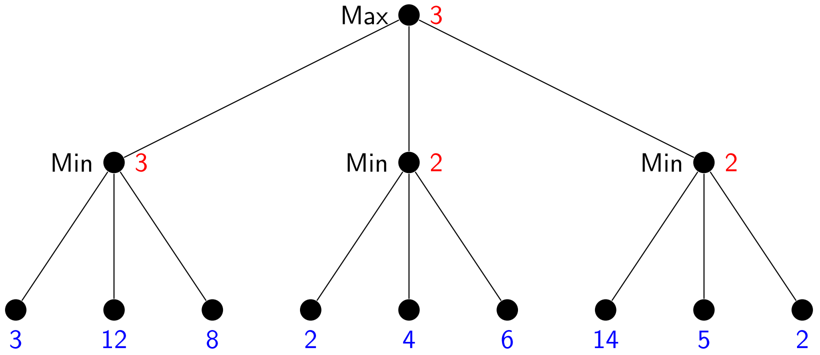


- Game tree, current player and action marked on the left.
- Last row: terminal positions with their utility.

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.

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 2.1.** The minimax algorithm (often just called minimax) is given by the following functions whose input 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 := −∞
  for each a ∈ 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 := +∞
  for each a ∈ Actions(s) do
    v := min(v,Max−Value(ChildState(s,a)))
  return v
```

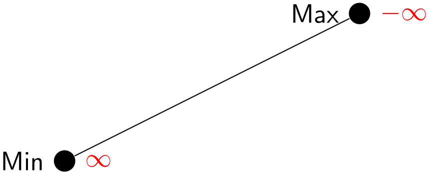
We call nodes, where Max/Min acts Max-nodes/Min-nodes.

Minimax: Example, Now in Detail

Max ● $-\infty$

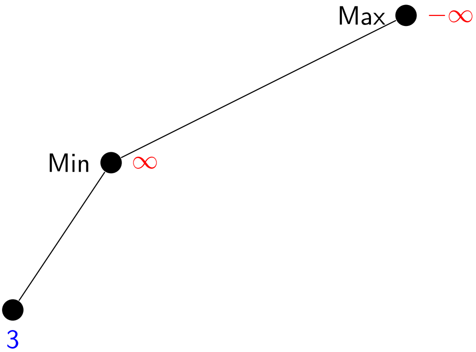
- So which action for Max is returned?

Minimax: Example, Now in Detail



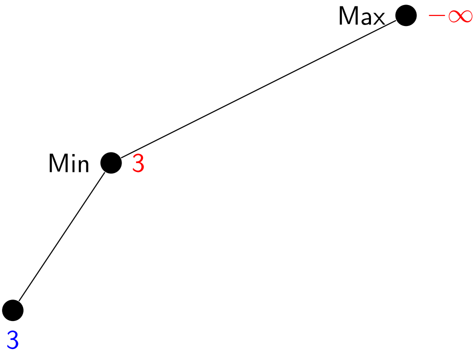
- So which action for Max is returned?

Minimax: Example, Now in Detail



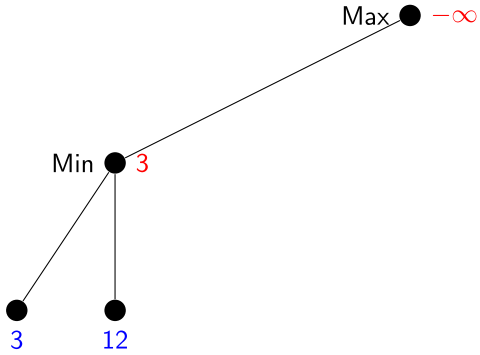
- So which action for Max is returned?

Minimax: Example, Now in Detail



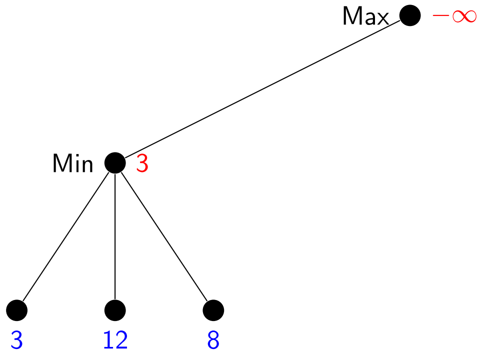
- So which action for Max is returned?

Minimax: Example, Now in Detail



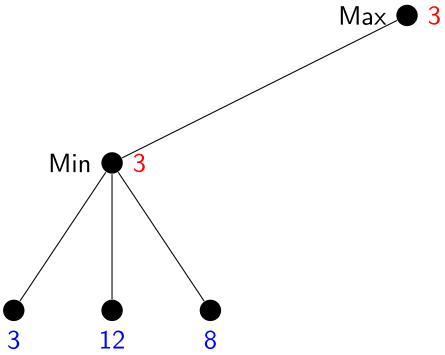
- So which action for Max is returned?

Minimax: Example, Now in Detail



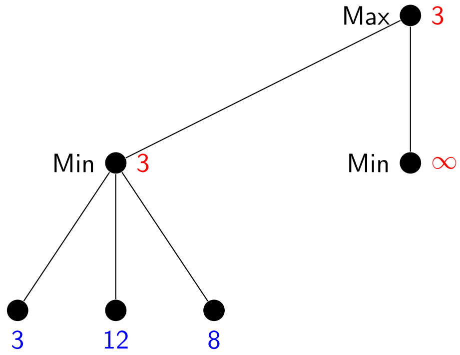
- So which action for Max is returned?

Minimax: Example, Now in Detail



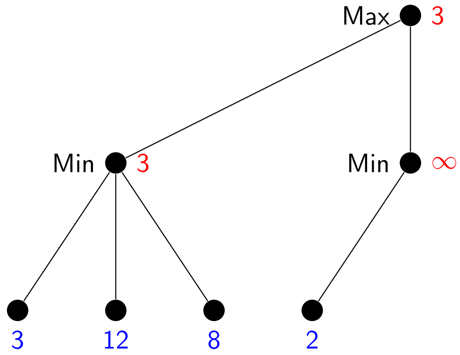
- So which action for Max is returned?

Minimax: Example, Now in Detail



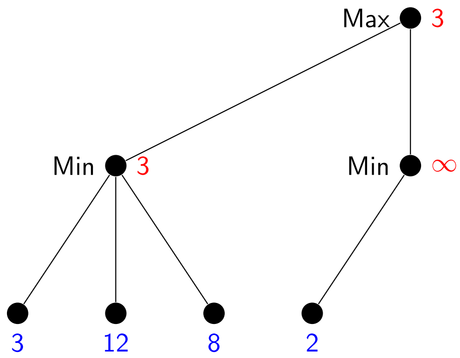
- So which action for Max is returned?

Minimax: Example, Now in Detail



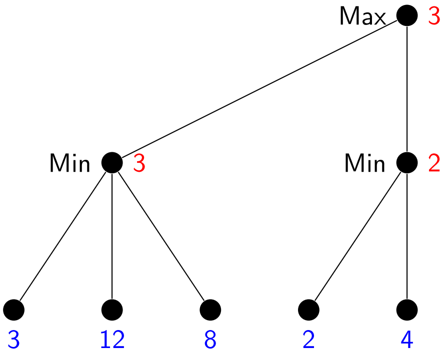
- So which action for Max is returned?

Minimax: Example, Now in Detail



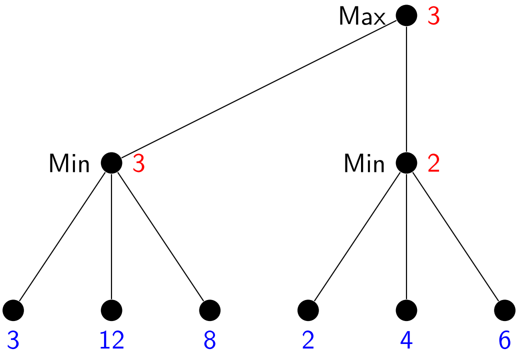
- So which action for Max is returned?

Minimax: Example, Now in Detail



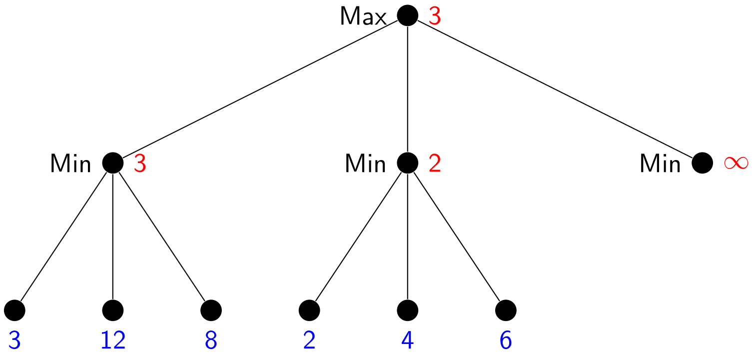
- So which action for Max is returned?

Minimax: Example, Now in Detail



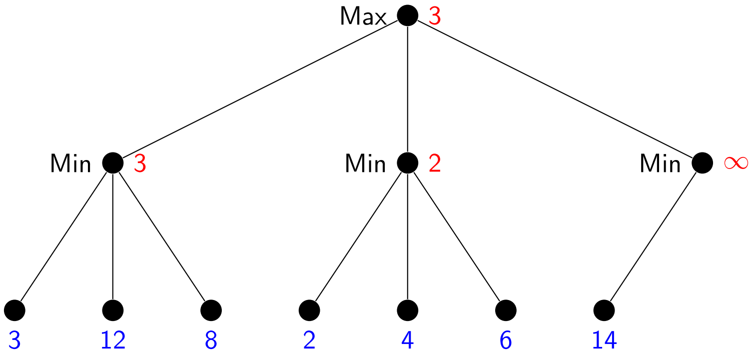
- So which action for Max is returned?

Minimax: Example, Now in Detail



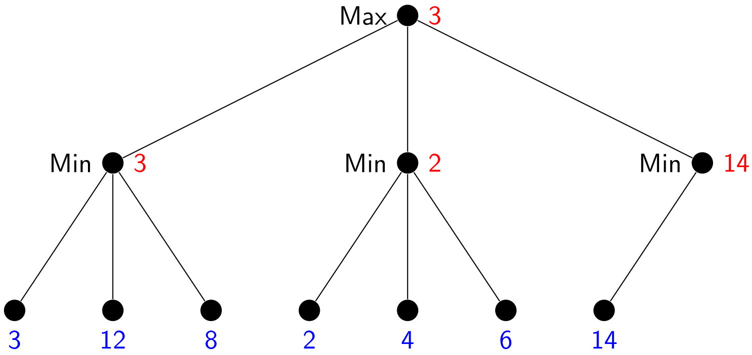
- So which action for Max is returned?

Minimax: Example, Now in Detail



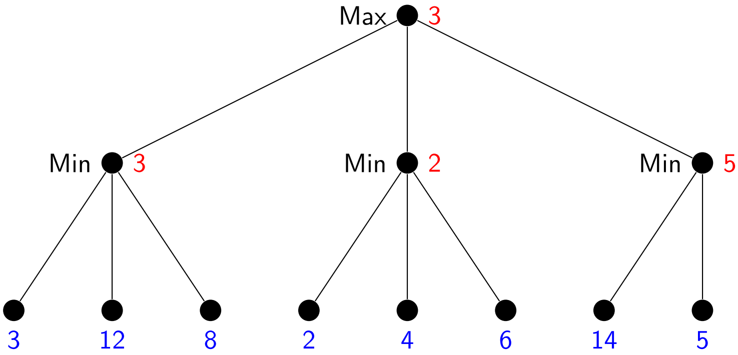
- So which action for Max is returned?

Minimax: Example, Now in Detail



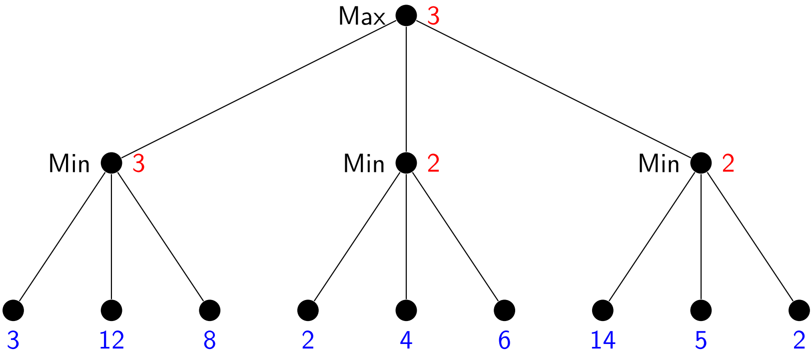
- So which action for Max is returned?

Minimax: Example, Now in Detail



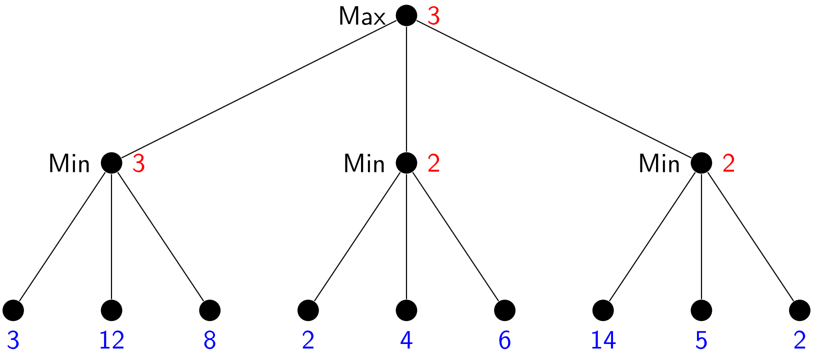
- So which action for Max is returned?

Minimax: Example, Now in Detail



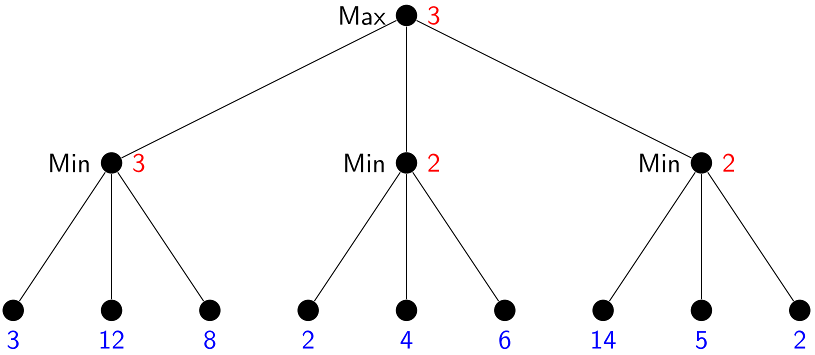
- So which action for Max is returned?

Minimax: Example, Now in Detail



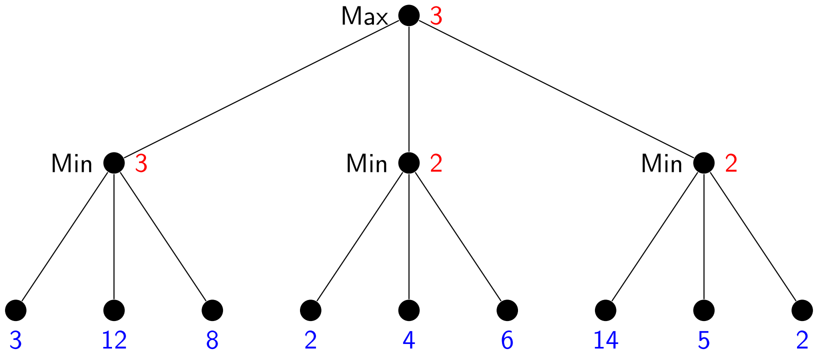
- So which action for Max is returned?

Minimax: Example, Now in Detail



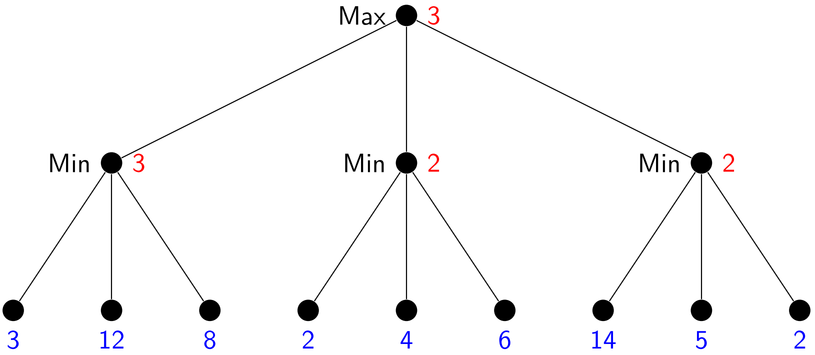
- So which action for Max is returned?

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".)

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, Pro and Contra

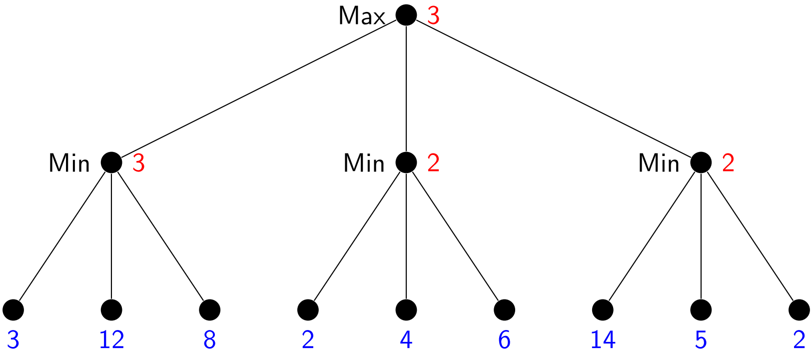
- **Minimax advantages:**
  - Minimax is the simplest possible (reasonable) search algorithm for games.
    (If any of you sat down, prior to this lecture, to implement a Tic-Tac-Toe player, chances are you either looked this up on Wikipedia, or invented it in the process.)
  - Returns an optimal action, 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.

7.3 Evaluation Functions

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 $\rightsquigarrow$ 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)

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

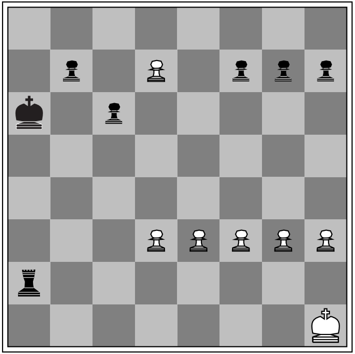
- Evaluation function in chess:
  - **Material**: Pawn 1, Knight 3, Bishop 3, Rook 5, Queen 9.
  - 3 points advantage $\rightsquigarrow$ safe win.
  - **Mobility**: How many fields do you control?
  - King safety, Pawn structure, . . .
- Note how simple this is! (probably is not how Kasparov evaluates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

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 $f_i\colon S \rightarrow \mathbb{R}$ 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 $s \xrightarrow{a} s'$, adapt $f(s)$ to $f(s')$ by considering only those features whose values have changed.

The Horizon Problem

- **Problem:** Critical aspects of the game can be cut off by the horizon.



Black to move

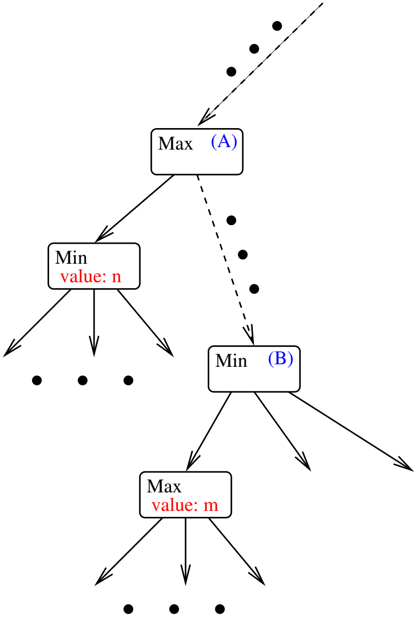
- Who's gonna win here?
  - White wins (pawn cannot be prevented from becoming a queen.)
  - Black has a $+4$ advantage in material, so if we cut-off here then our evaluation function will say "100, black wins".
  - The loss for black is "beyond our horizon" unless we search extremely deeply: black can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

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 3.3 (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 3.4.** 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.

7.4 Alpha-Beta Search

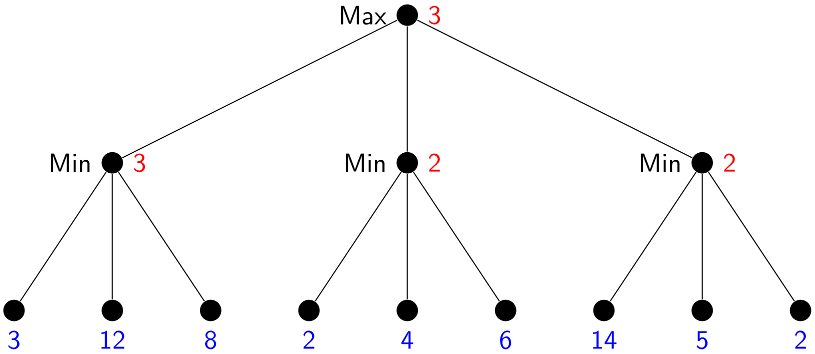
When We Already Know We Can Do Better Than This



- 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).

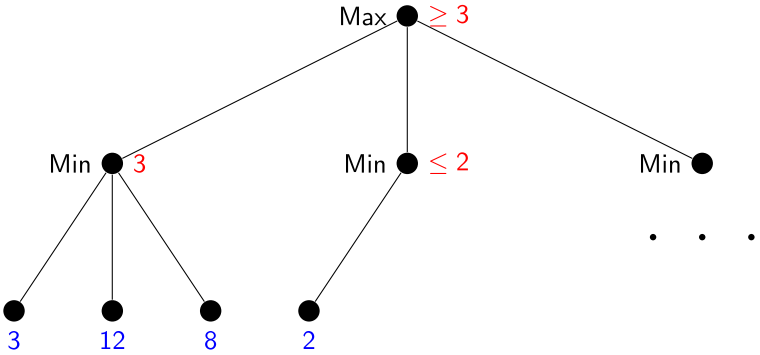
Alpha Pruning: Basic Idea

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



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 $\leadsto$ better algorithm

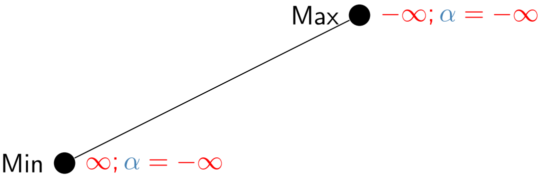
Alpha Pruning

- **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.
- **Example 4.2 (Computing alpha values).**

$$\text{Max} \bullet -\infty; \alpha = -\infty$$

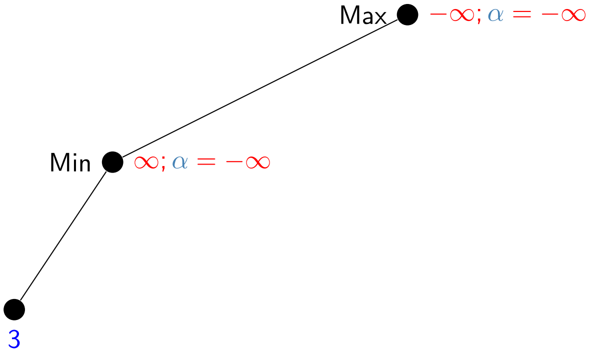
Alpha Pruning

- **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).**



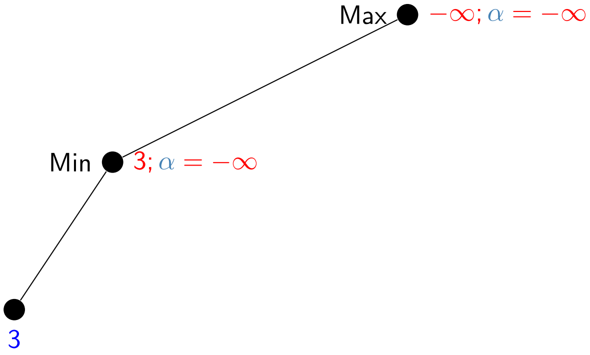
Alpha Pruning

- **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).**



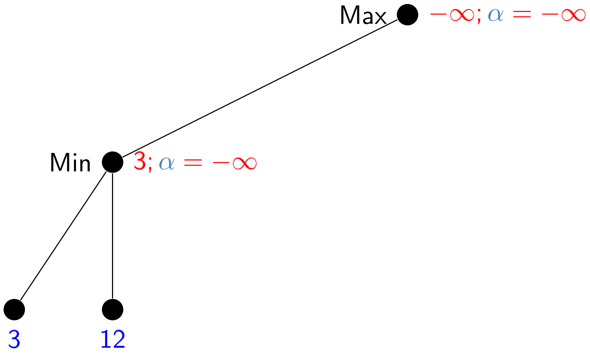
Alpha Pruning

- **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).**



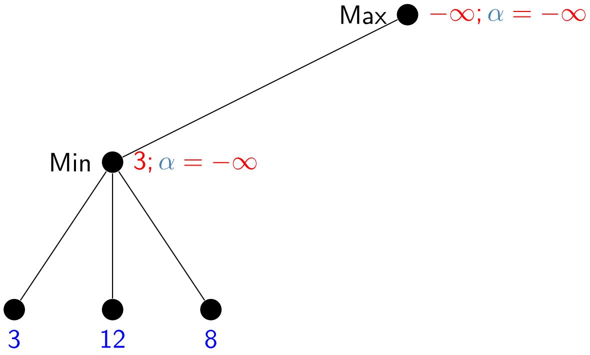
Alpha Pruning

- **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).**



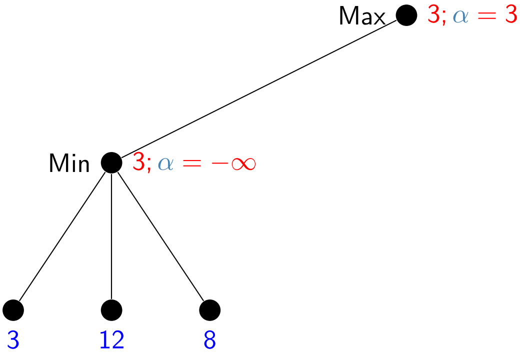
Alpha Pruning

- **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).**



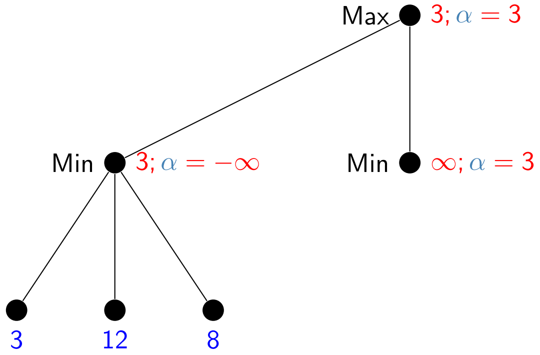
Alpha Pruning

- **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).**



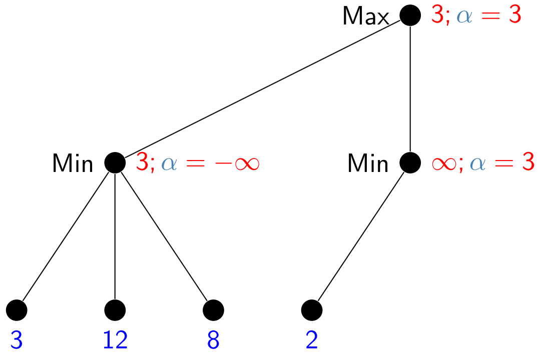
Alpha Pruning

- **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).**



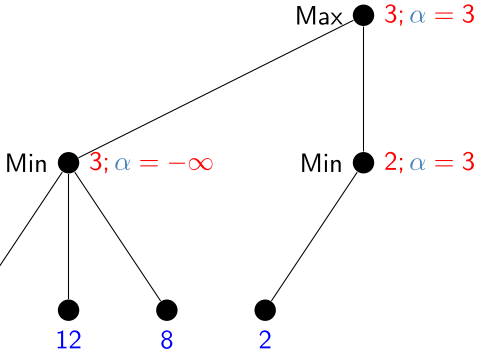
Alpha Pruning

- **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).**



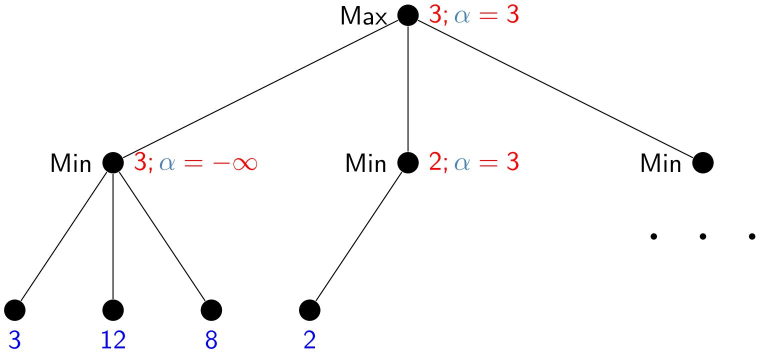
Alpha Pruning

- **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).**



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! $\leadsto$ alphabeta-pruning

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 209) + α/β book-keeping and pruning.

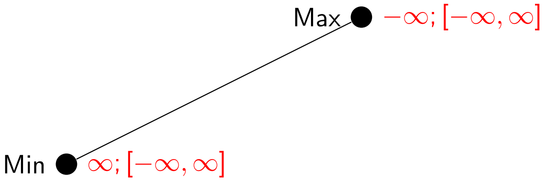
Alpha-Beta Search: Example

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

Max ● $-\infty; [-\infty, \infty]$

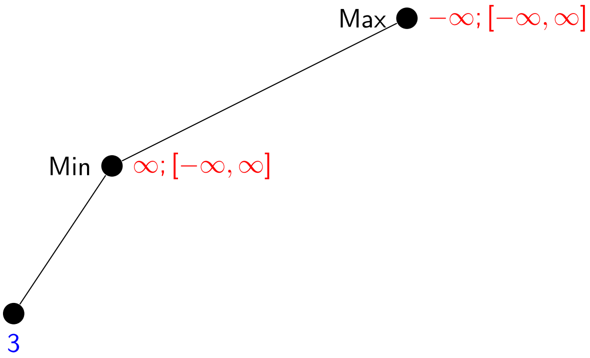
Alpha-Beta Search: Example

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



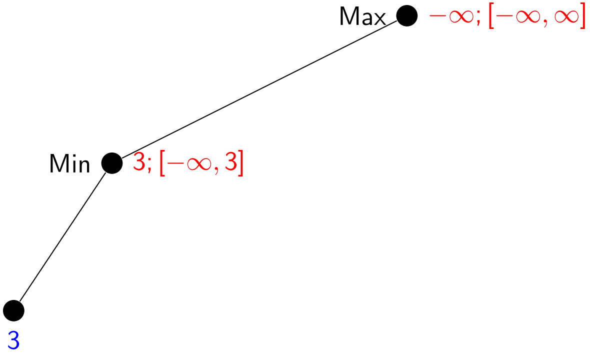
Alpha-Beta Search: Example

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



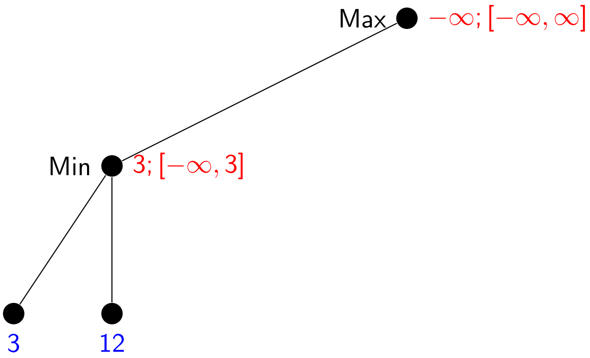
Alpha-Beta Search: Example

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



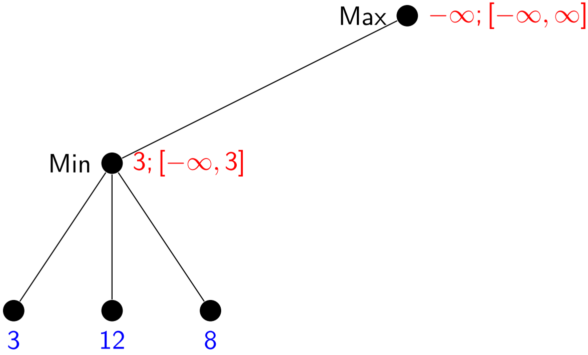
Alpha-Beta Search: Example

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



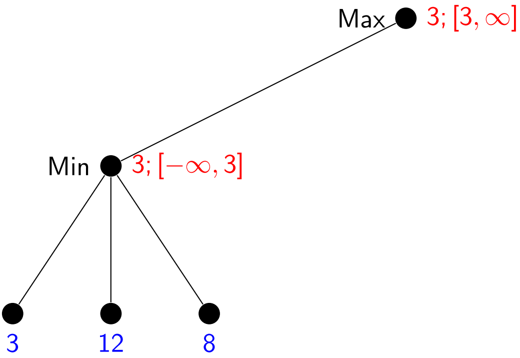
Alpha-Beta Search: Example

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



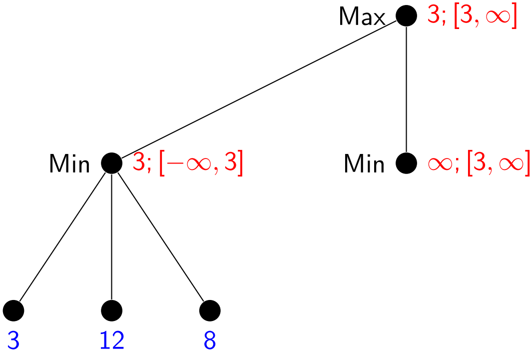
Alpha-Beta Search: Example

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



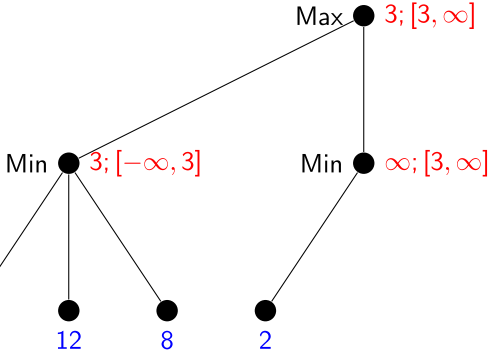
Alpha-Beta Search: Example

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



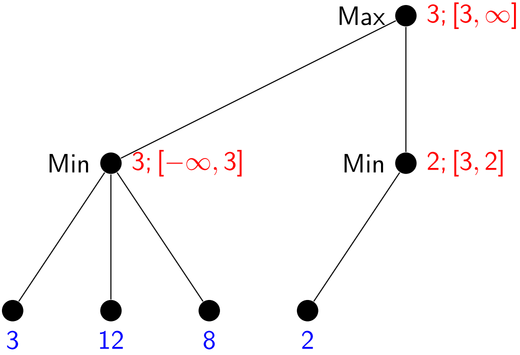
Alpha-Beta Search: Example

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



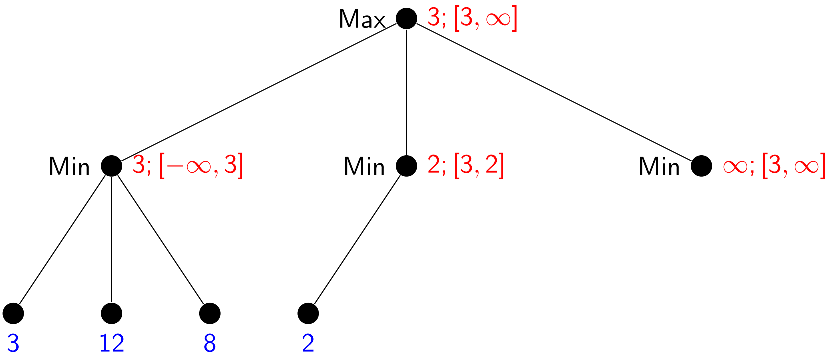
Alpha-Beta Search: Example

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



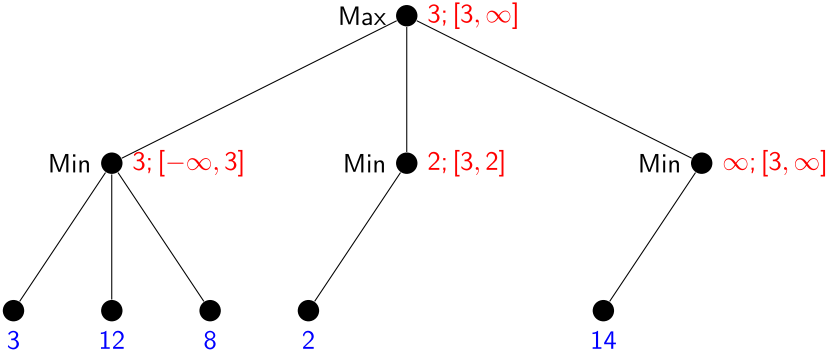
Alpha-Beta Search: Example

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



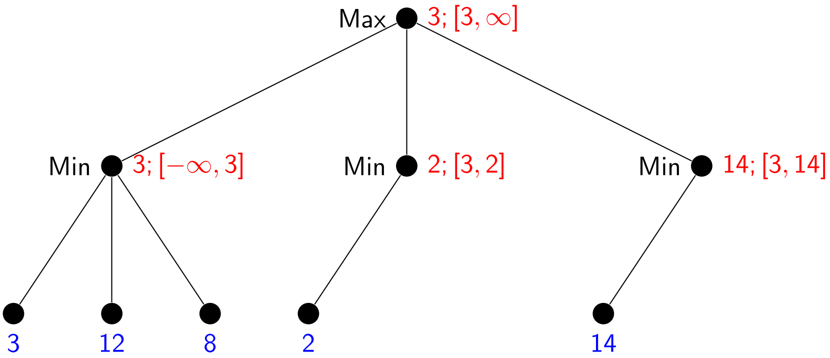
Alpha-Beta Search: Example

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



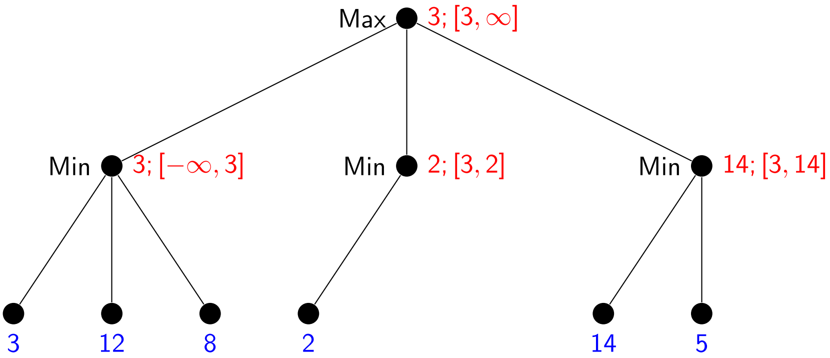
Alpha-Beta Search: Example

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



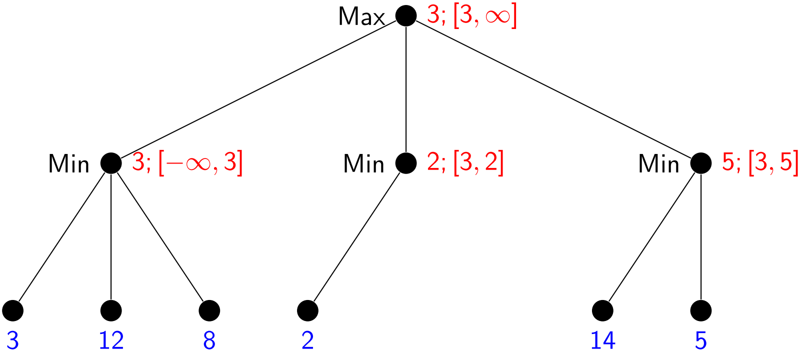
Alpha-Beta Search: Example

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



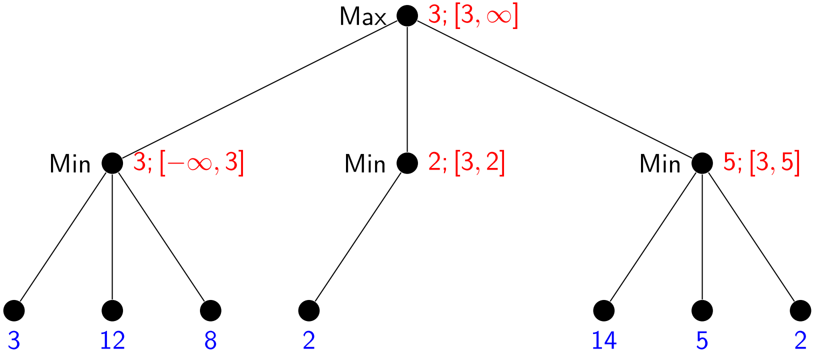
Alpha-Beta Search: Example

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



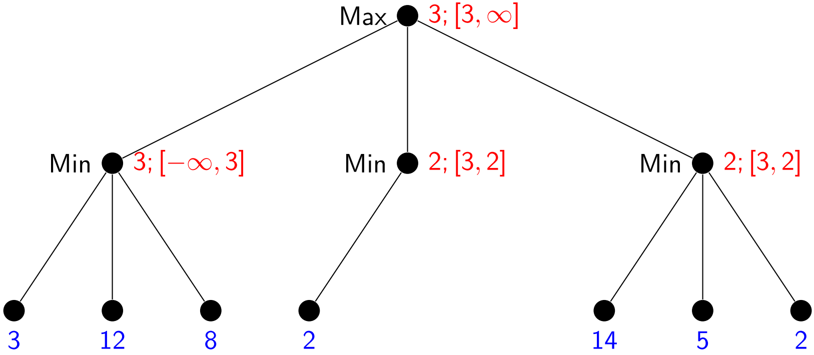
Alpha-Beta Search: Example

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



Alpha-Beta Search: Example

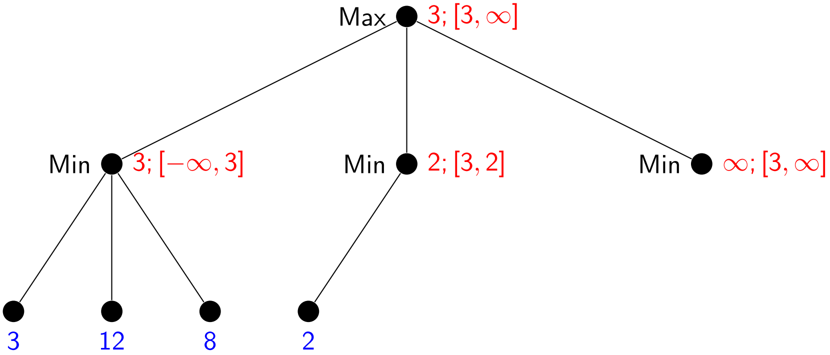
- **Notation:** $v; [\alpha, \beta]$



- **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!

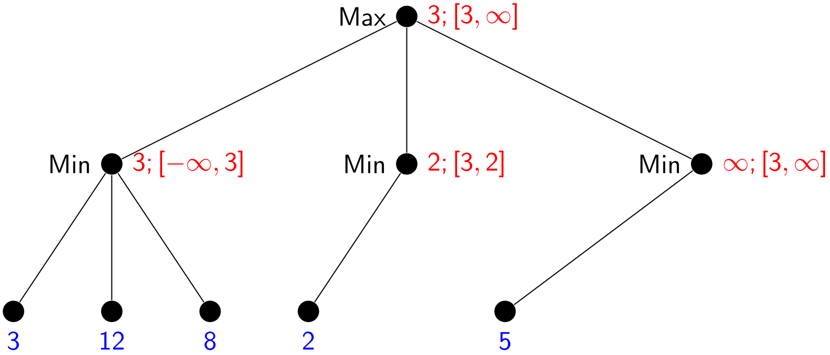
Alpha-Beta Search: Modified Example

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



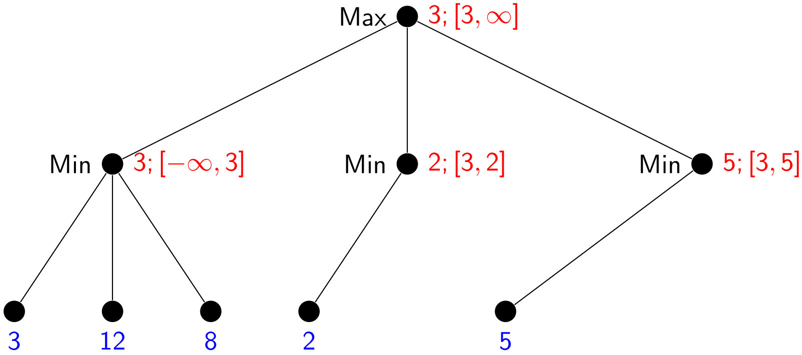
Alpha-Beta Search: Modified Example

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



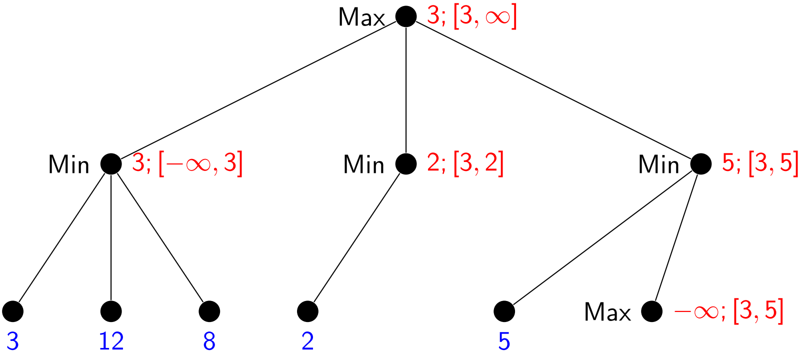
Alpha-Beta Search: Modified Example

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



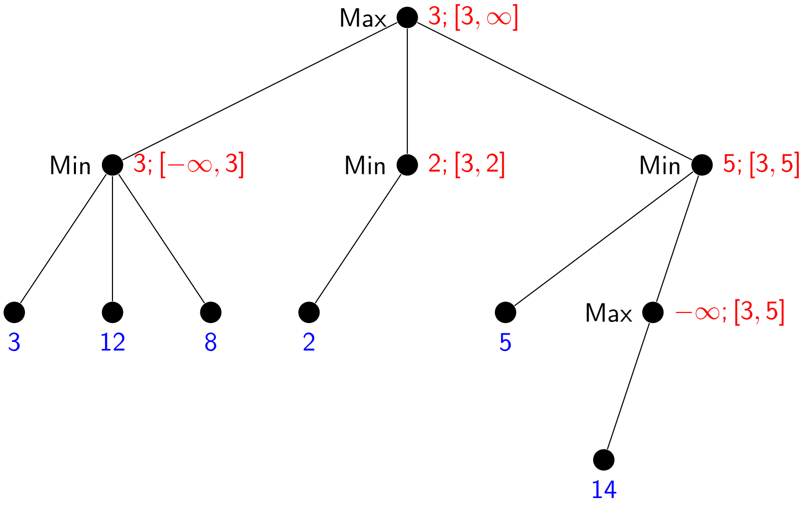
Alpha-Beta Search: Modified Example

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



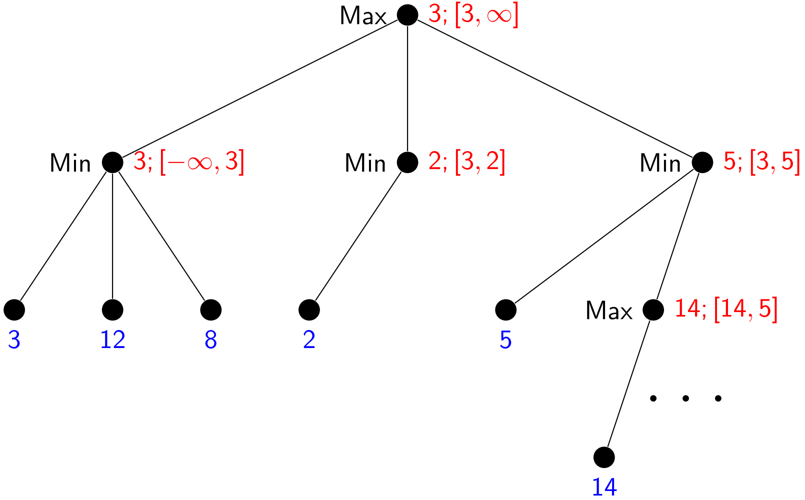
Alpha-Beta Search: Modified Example

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



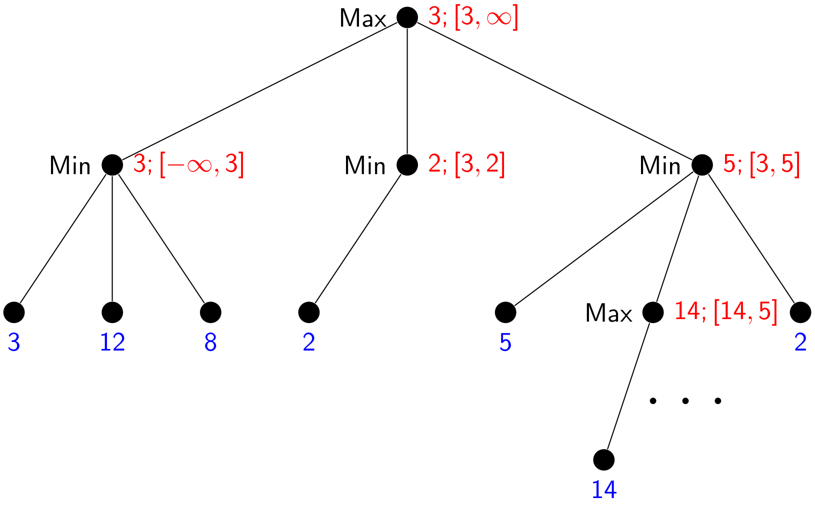
Alpha-Beta Search: Modified Example

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



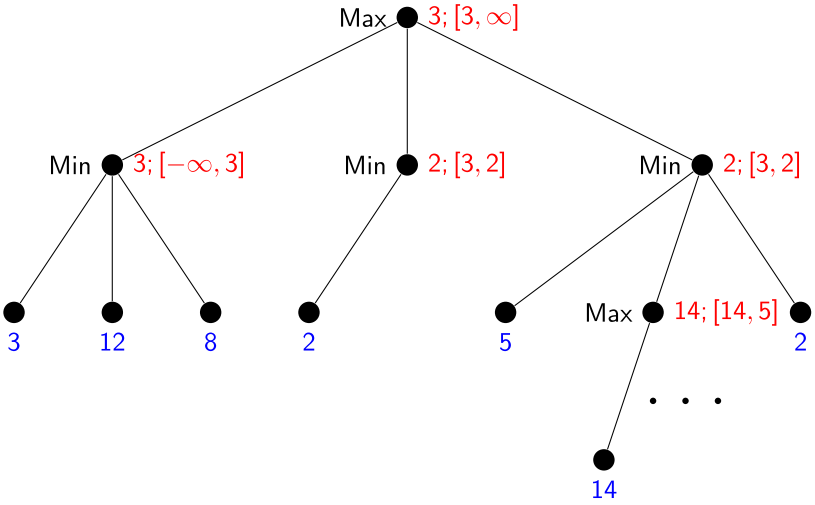
Alpha-Beta Search: Modified Example

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



Alpha-Beta Search: Modified Example

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



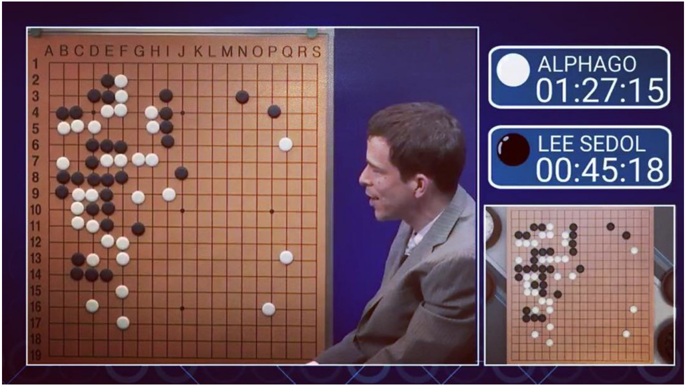
How Much Pruning Do We Get?

- 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.

7.5 Monte-Carlo Tree Search (MCTS)

And now . . .

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



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 $u(t)$

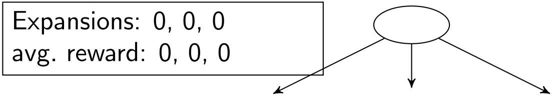
- **Definition 5.2.** For the Monte Carlo tree search algorithm (MCTS) we maintain a search tree T, the MCTS tree.

  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.

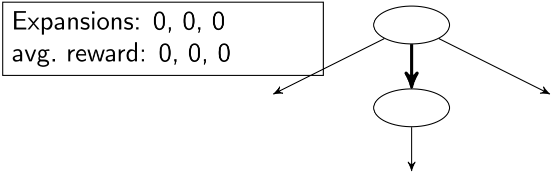
Monte-Carlo Sampling: Illustration of Sampling

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



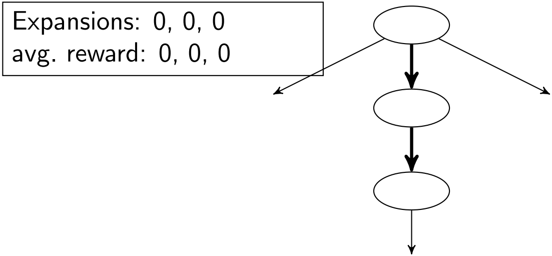
Monte-Carlo Sampling: Illustration of Sampling

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



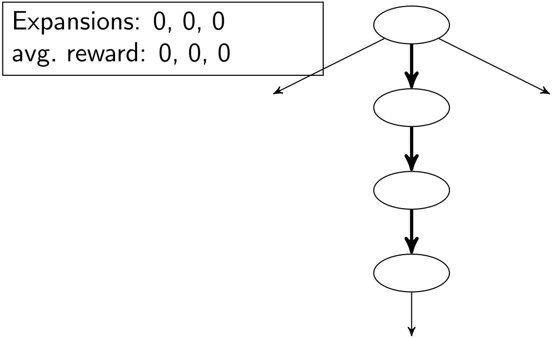
Monte-Carlo Sampling: Illustration of Sampling

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



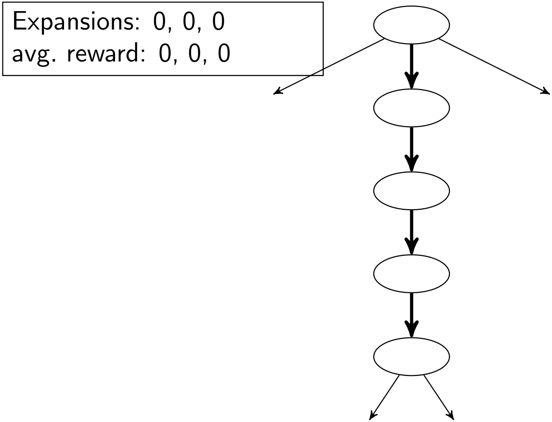
Monte-Carlo Sampling: Illustration of Sampling

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



Monte-Carlo Sampling: Illustration of Sampling

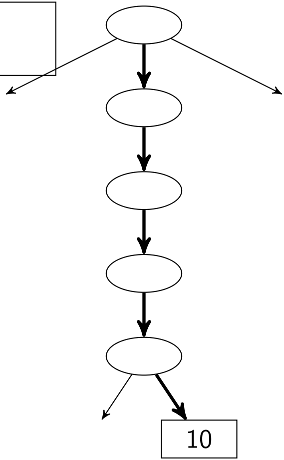
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.7 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Monte-Carlo Sampling: Illustration of Sampling

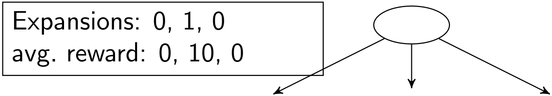
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.8 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

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



Monte-Carlo Sampling: Illustration of Sampling

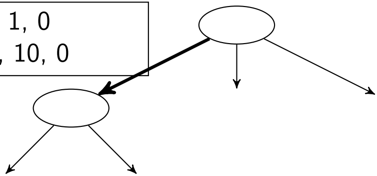
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.9 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Monte-Carlo Sampling: Illustration of Sampling

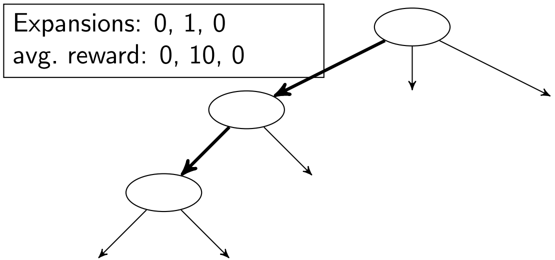
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.10 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

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



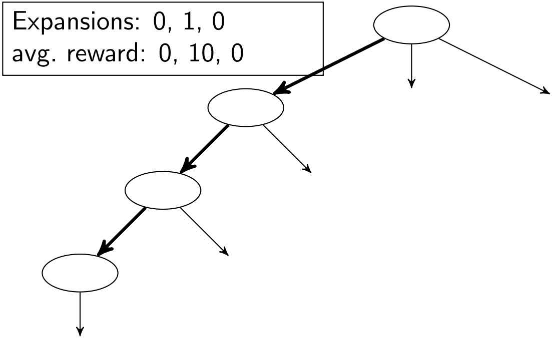
Monte-Carlo Sampling: Illustration of Sampling

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



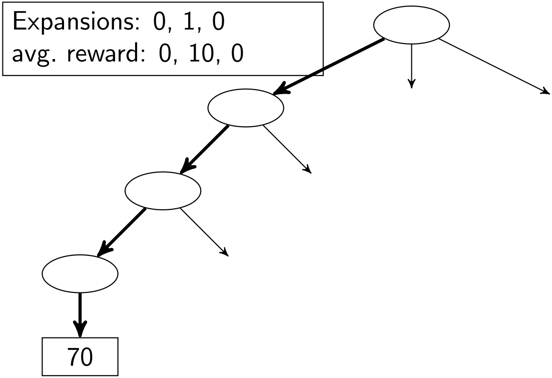
Monte-Carlo Sampling: Illustration of Sampling

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



Monte-Carlo Sampling: Illustration of Sampling

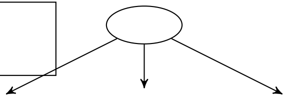
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.13 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Monte-Carlo Sampling: Illustration of Sampling

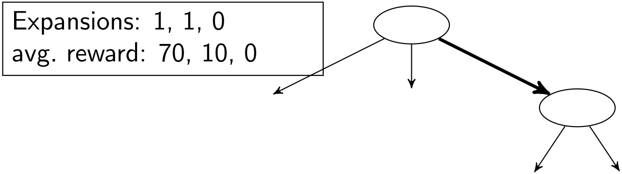
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.14 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

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



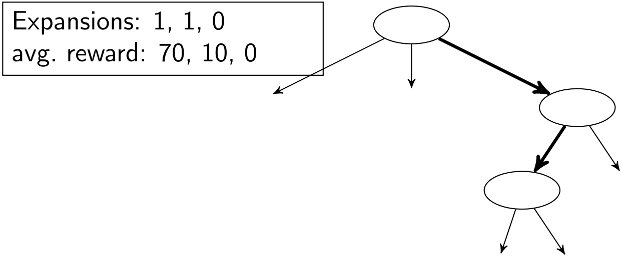
Monte-Carlo Sampling: Illustration of Sampling

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



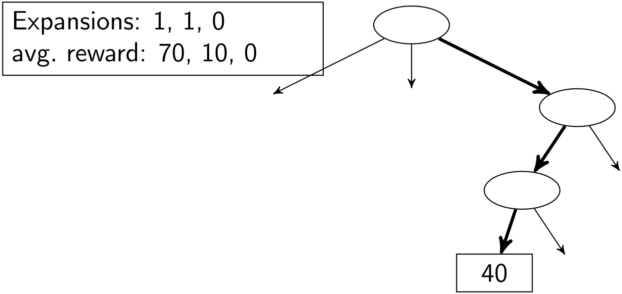
Monte-Carlo Sampling: Illustration of Sampling

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



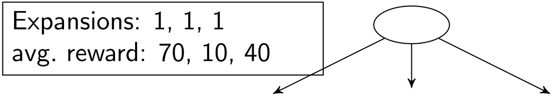
Monte-Carlo Sampling: Illustration of Sampling

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



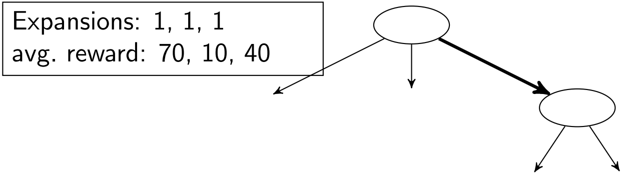
Monte-Carlo Sampling: Illustration of Sampling

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



Monte-Carlo Sampling: Illustration of Sampling

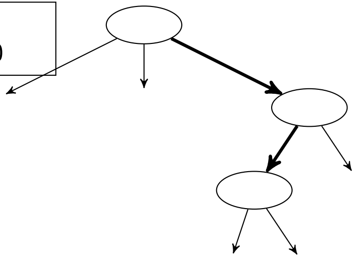
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.19 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



Monte-Carlo Sampling: Illustration of Sampling

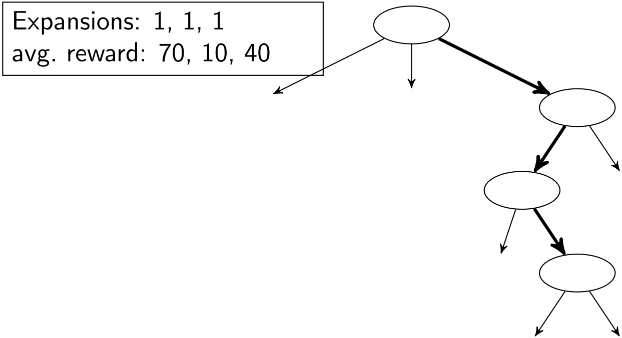
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.20 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

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



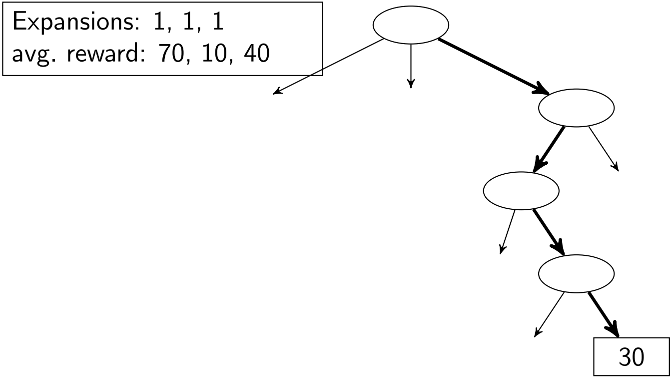
Monte-Carlo Sampling: Illustration of Sampling

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



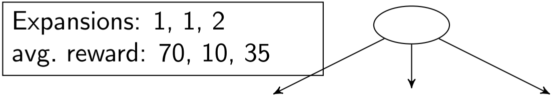
Monte-Carlo Sampling: Illustration of Sampling

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



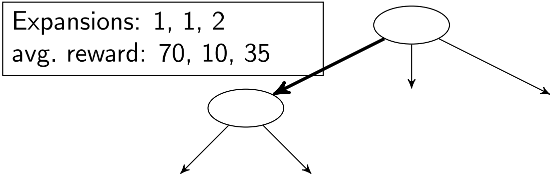
Monte-Carlo Sampling: Illustration of Sampling

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



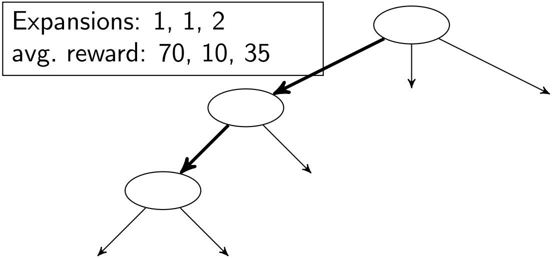
Monte-Carlo Sampling: Illustration of Sampling

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



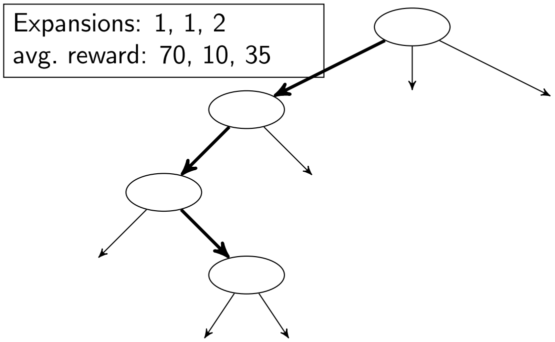
Monte-Carlo Sampling: Illustration of Sampling

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



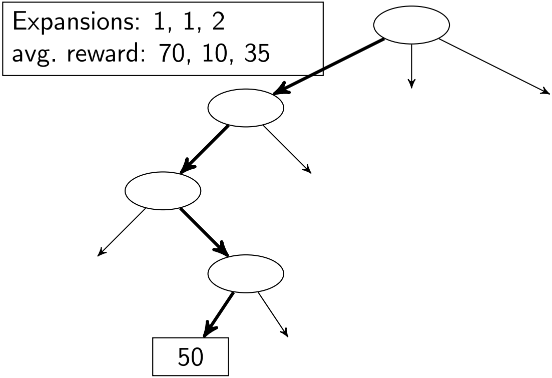
Monte-Carlo Sampling: Illustration of Sampling

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



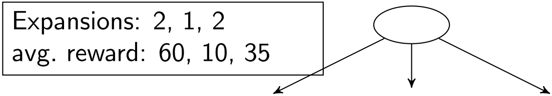
Monte-Carlo Sampling: Illustration of Sampling

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



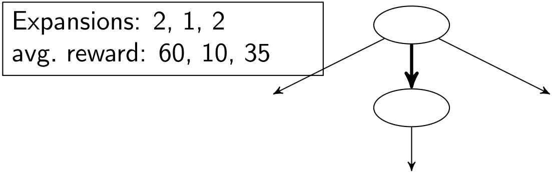
Monte-Carlo Sampling: Illustration of Sampling

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



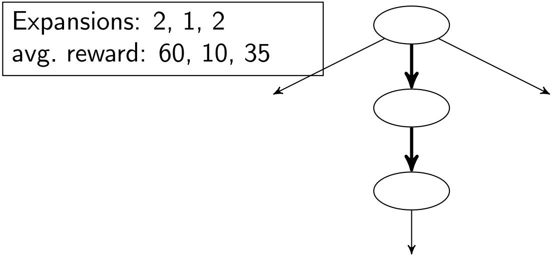
Monte-Carlo Sampling: Illustration of Sampling

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



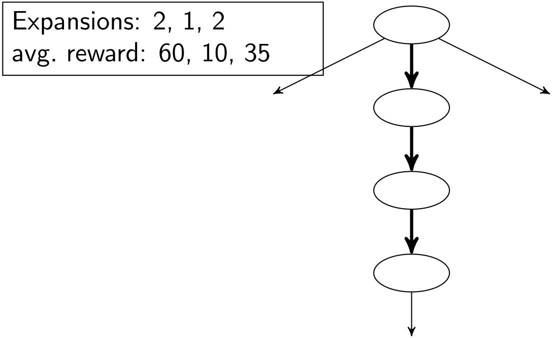
Monte-Carlo Sampling: Illustration of Sampling

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



Monte-Carlo Sampling: Illustration of Sampling

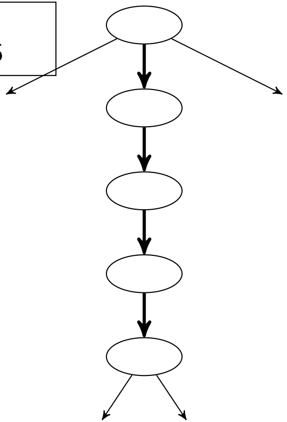
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.31 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



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)

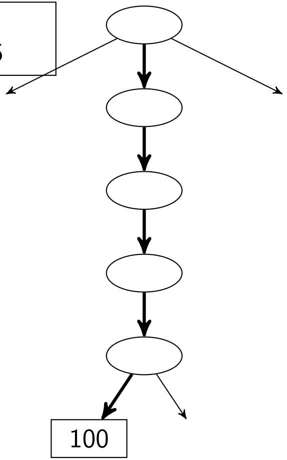
Expansions: 2, 1, 2
avg. reward: 60, 10, 35



Monte-Carlo Sampling: Illustration of Sampling

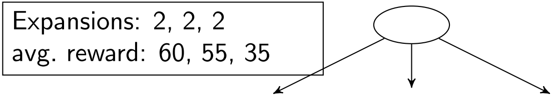
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.33 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

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



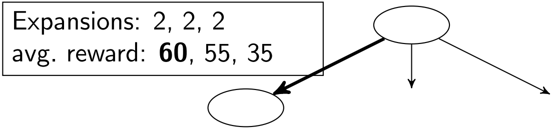
Monte-Carlo Sampling: Illustration of Sampling

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



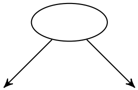
Monte-Carlo Sampling: Illustration of Sampling

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



Monte-Carlo Sampling: Illustration of Sampling

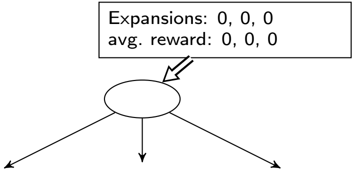
- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 5.36 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



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

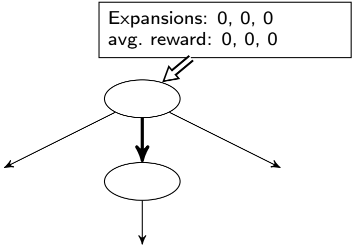
Monte-Carlo Tree Search: Building the Tree

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



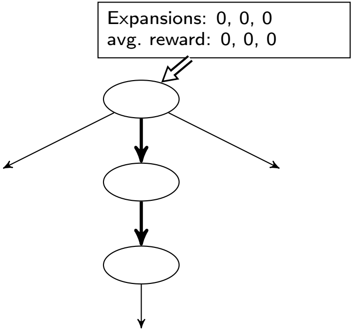
Monte-Carlo Tree Search: Building the Tree

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



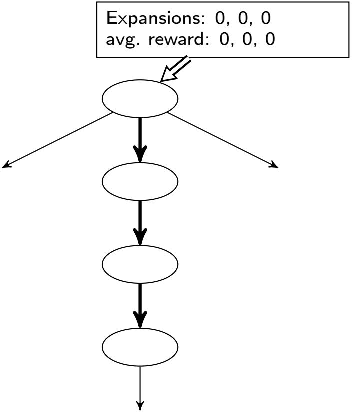
Monte-Carlo Tree Search: Building the Tree

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



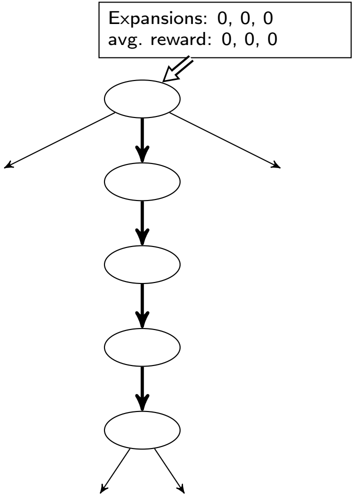
Monte-Carlo Tree Search: Building the Tree

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



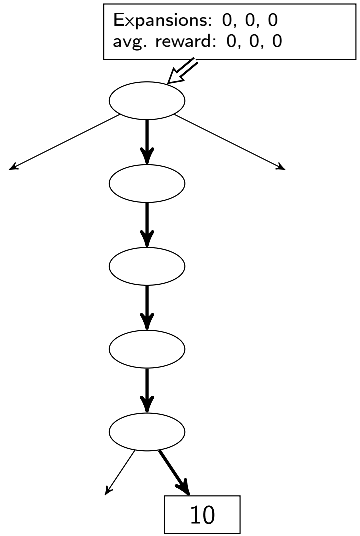
Monte-Carlo Tree Search: Building the Tree

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



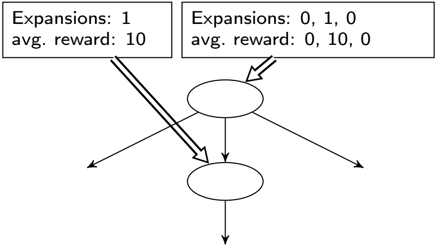
Monte-Carlo Tree Search: Building the Tree

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



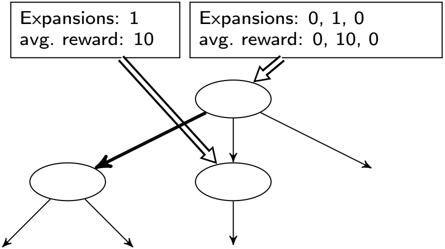
Monte-Carlo Tree Search: Building the Tree

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



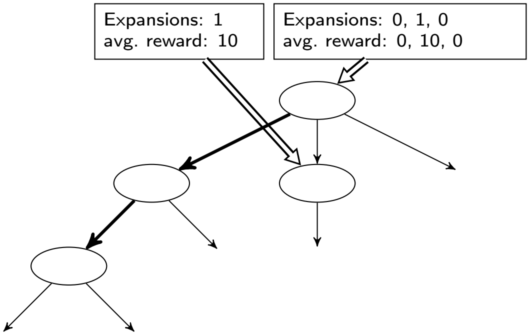
Monte-Carlo Tree Search: Building the Tree

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



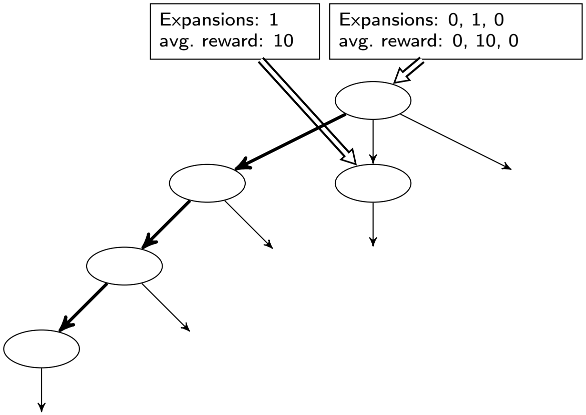
Monte-Carlo Tree Search: Building the Tree

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



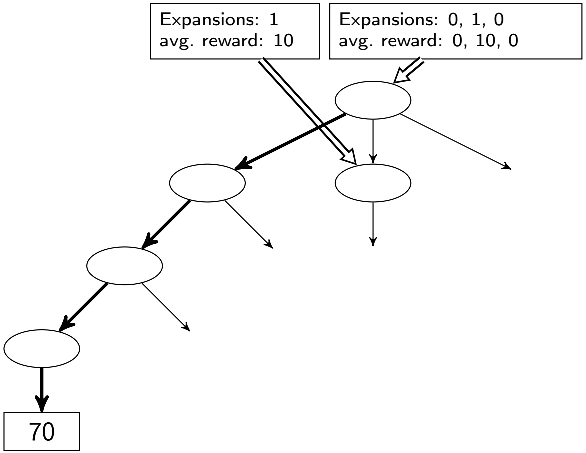
Monte-Carlo Tree Search: Building the Tree

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



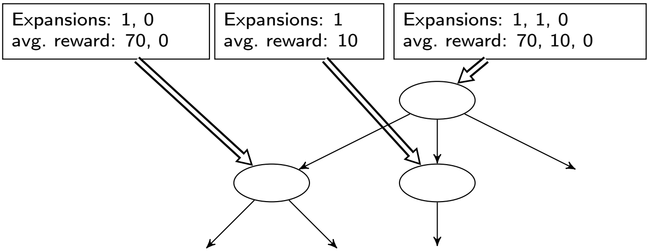
Monte-Carlo Tree Search: Building the Tree

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



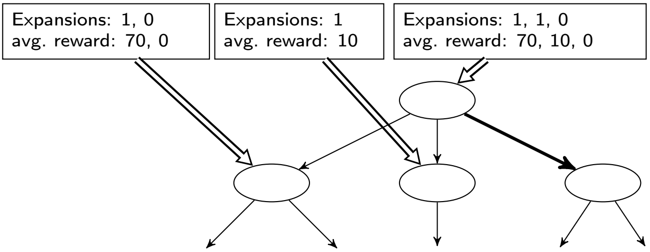
Monte-Carlo Tree Search: Building the Tree

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



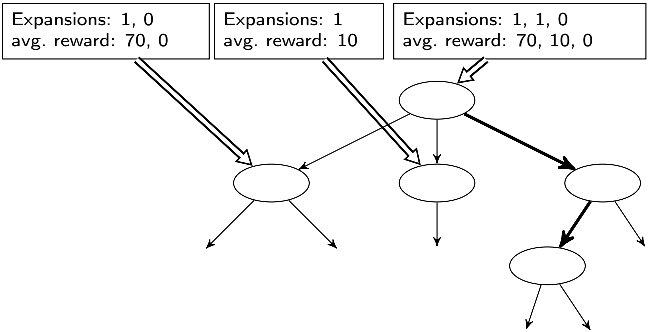
Monte-Carlo Tree Search: Building the Tree

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



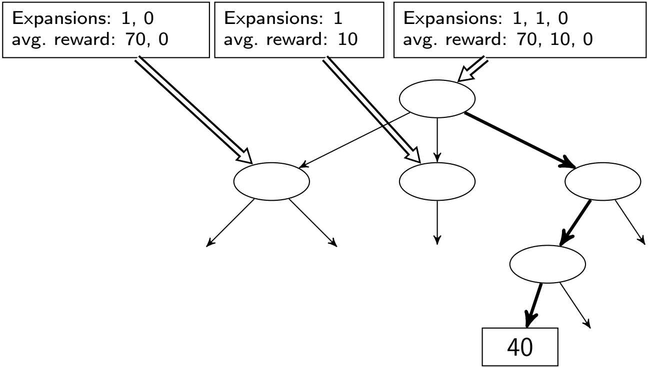
Monte-Carlo Tree Search: Building the Tree

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



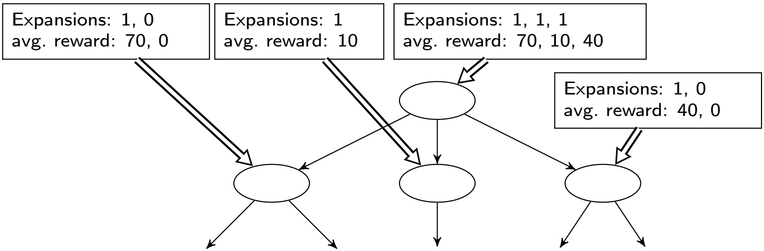
Monte-Carlo Tree Search: Building the Tree

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



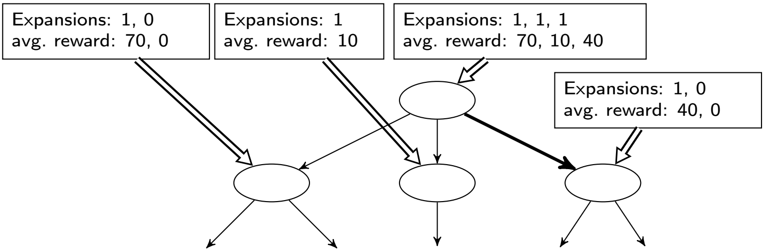
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.52 (Redoing the previous example).**



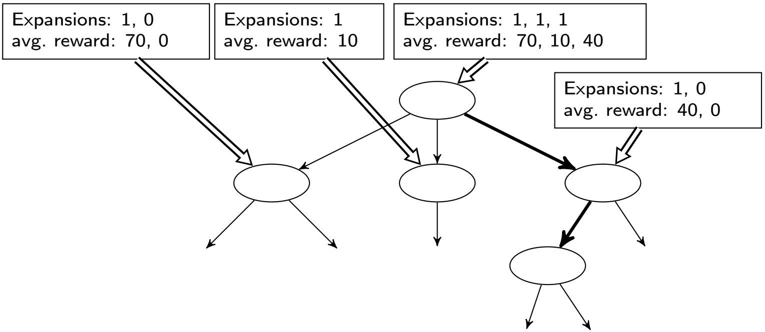
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.53 (Redoing the previous example).**



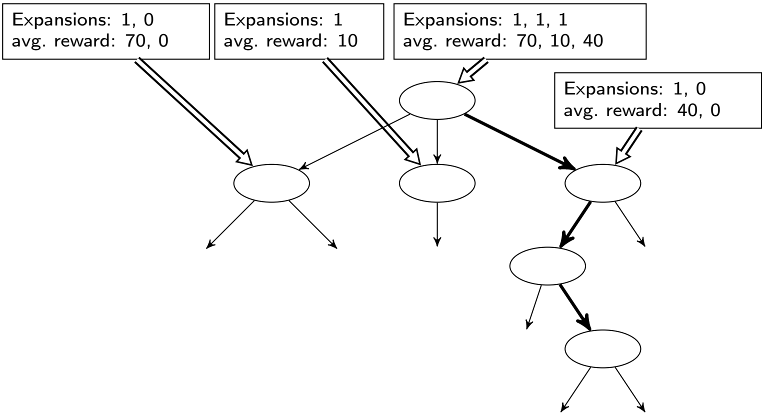
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.54 (Redoing the previous example).**



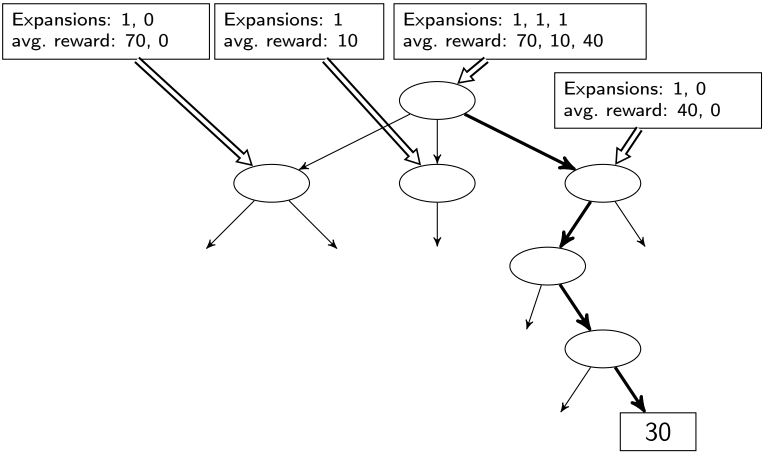
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.55 (Redoing the previous example).**



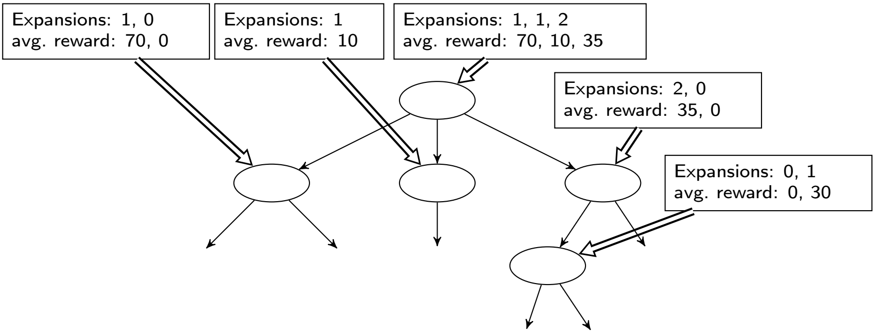
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.56 (Redoing the previous example).**



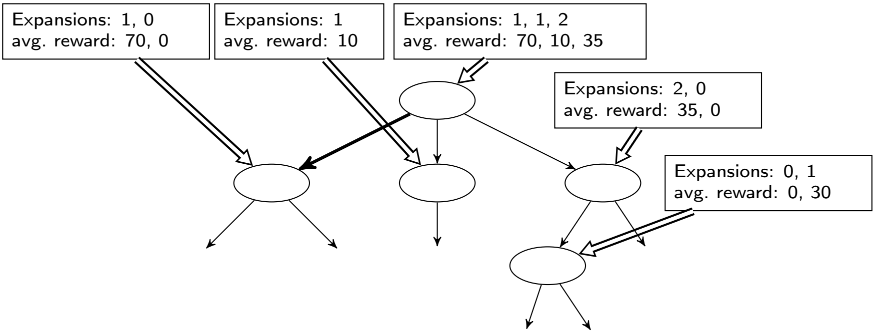
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.57 (Redoing the previous example).**



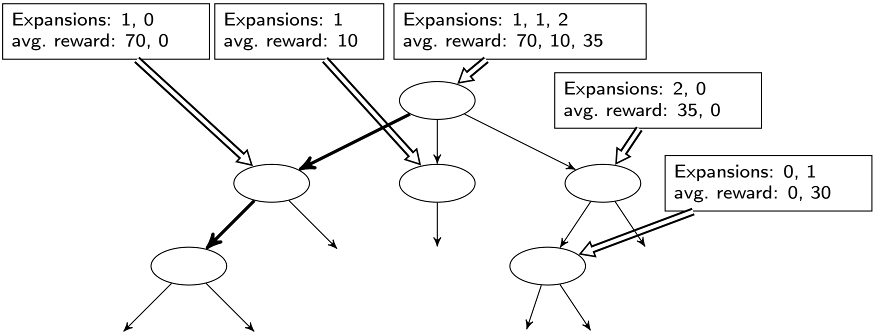
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.58 (Redoing the previous example).**



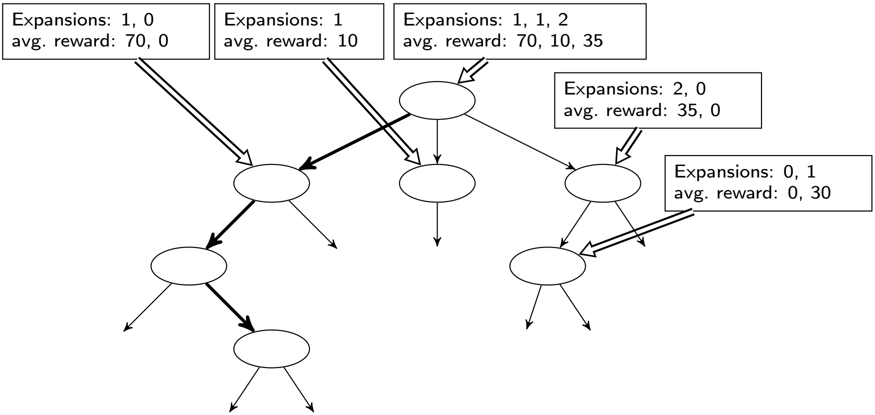
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.59 (Redoing the previous example).**



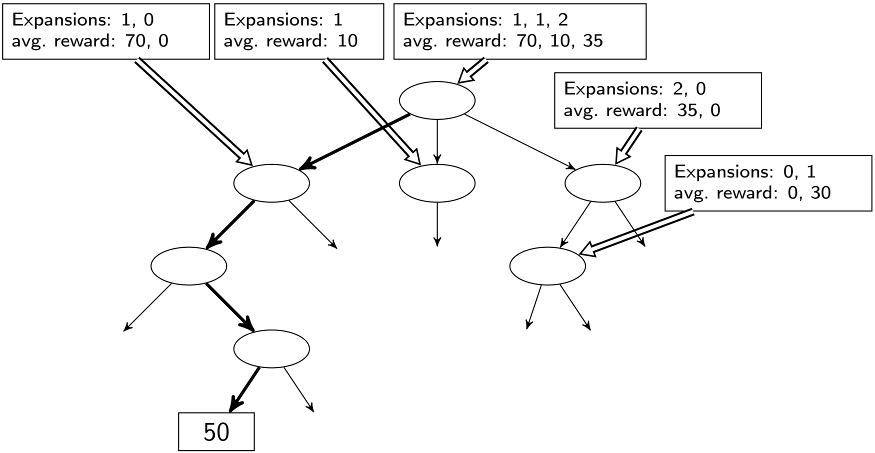
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.60 (Redoing the previous example).**



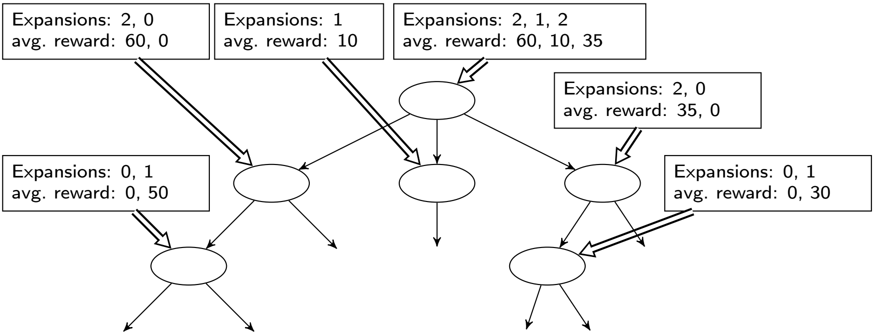
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.61 (Redoing the previous example).**



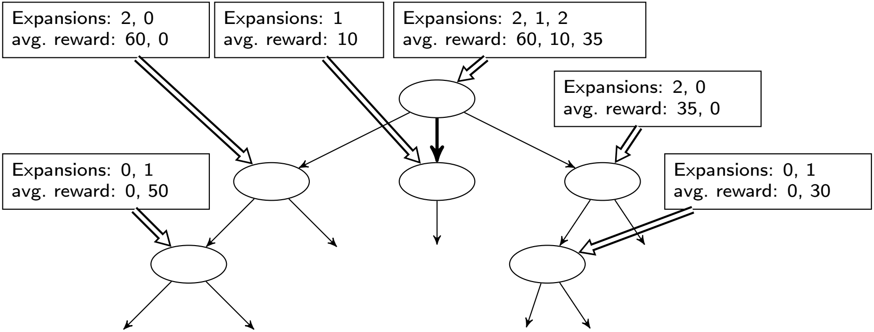
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.62 (Redoing the previous example).**



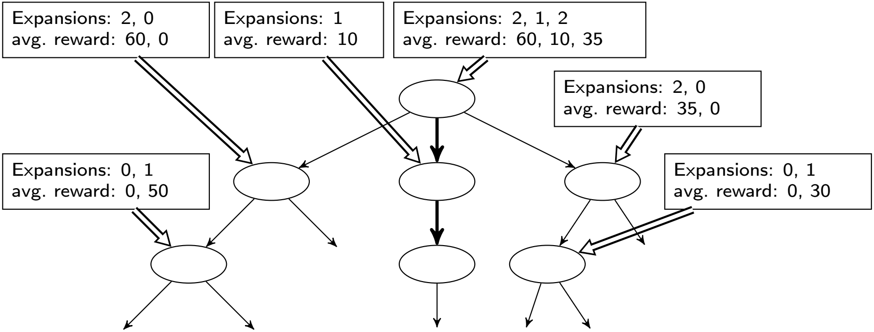
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.63 (Redoing the previous example).**



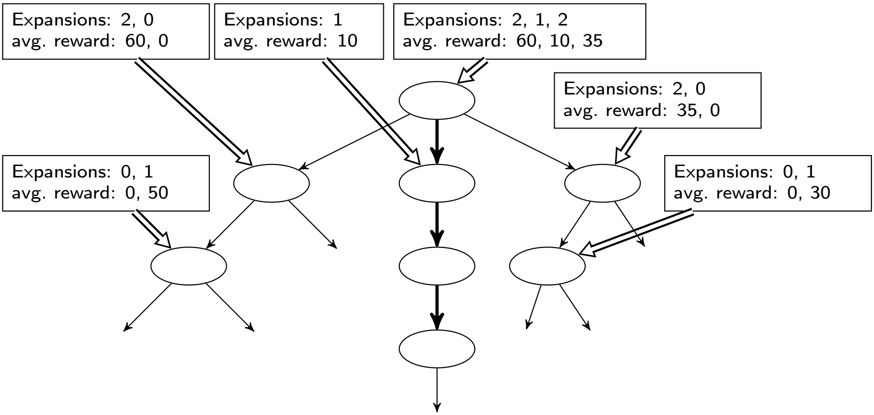
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.64 (Redoing the previous example).**



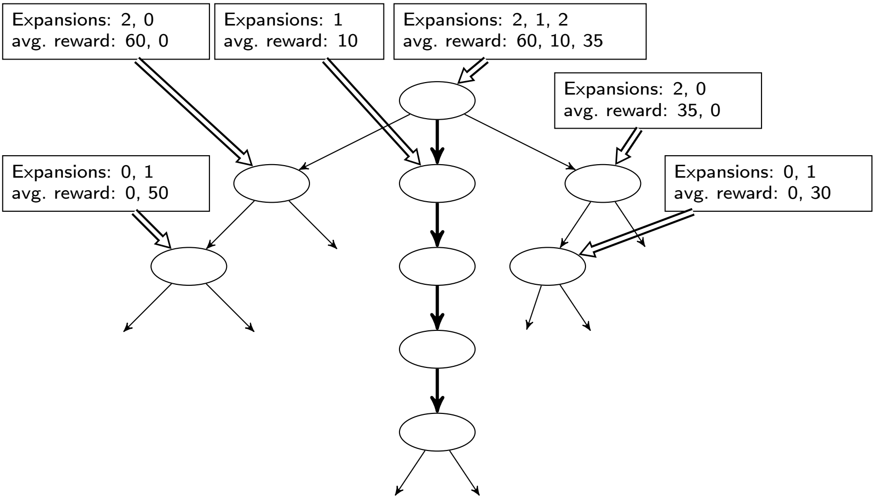
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.65 (Redoing the previous example).**



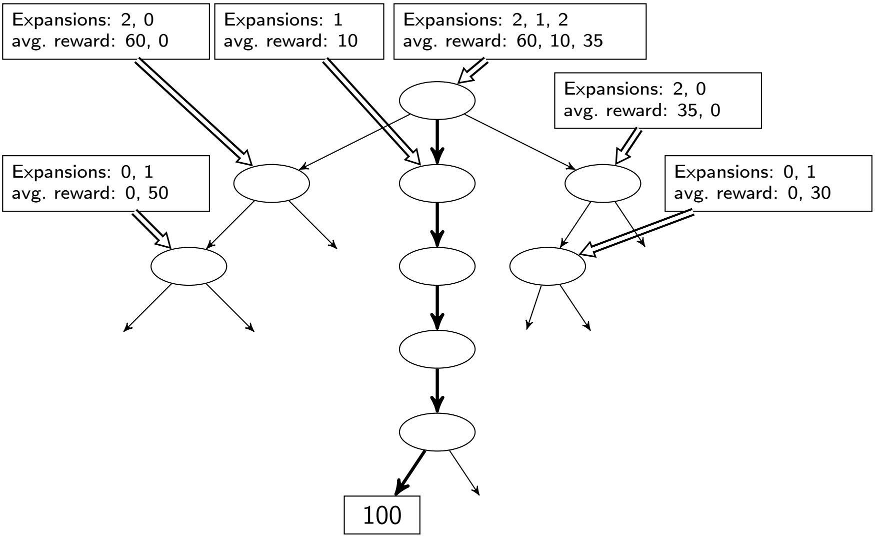
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.66 (Redoing the previous example).**



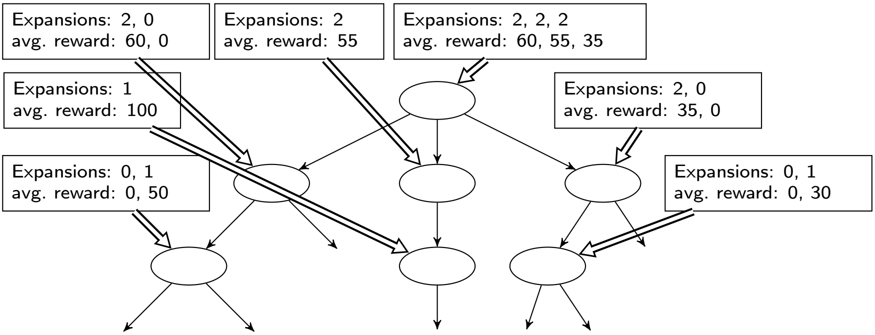
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.67 (Redoing the previous example).**



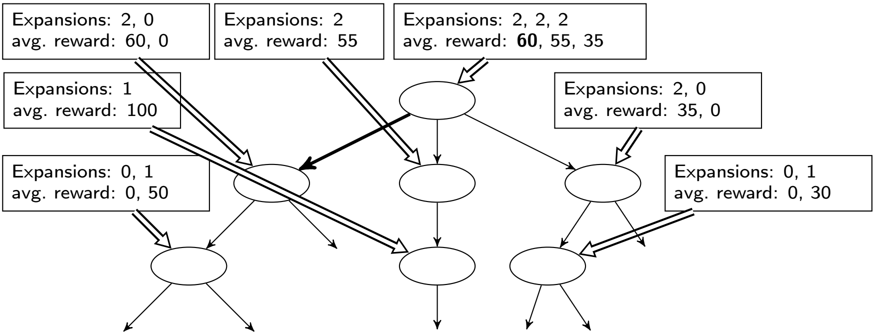
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.68 (Redoing the previous example).**



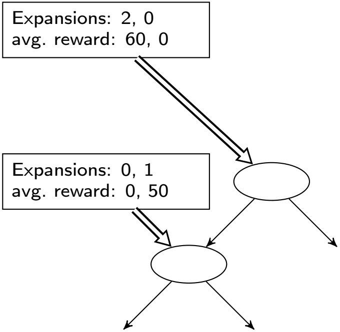
Monte-Carlo Tree Search: Building the Tree

- **Idea:** We can save work by building the tree as we go along.
- **Example 5.69 (Redoing the previous example).**



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).**



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” [KS06].
  - Inspired by Multi-Armed Bandit (as in: Casino) problems.
  - Basically a formula defining the balance. Very popular (buzzword).
  - Recent critics (e.g. [FD14]): 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).

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

Neural Networks in AlphaGo

- **Neural network training pipeline and architecture:**

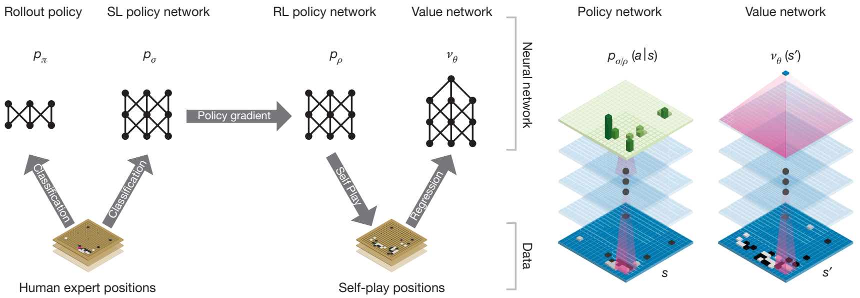


Illustration taken from [Sil+16] .

- 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”).

- **Monte Carlo tree search in AlphaGo:**

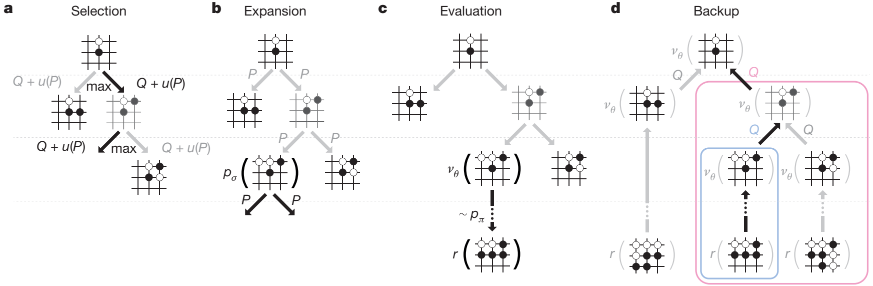


Illustration taken from [Sil+16]

- *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.

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.

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.

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))

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))

7.7 Conclusion

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").

Chapter 8
Constraint Satisfaction Problems

8.1 Constraint Satisfaction Problems: Motivation

A (Constraint Satisfaction) Problem

- **Example 1.1 (Tournament Schedule).** Who's going to play against who, when and where?

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 search problem, where the states are given by a finite set $V:=\{X_1,\ldots,X_n\}$ of variables and domains $\{D_v|v\in V\}$ and the goal state are specified by a set of constraints specifying allowable combinations of values for subsets of variables.
- **Definition 1.3.** A constraint network γ is satisfiable, iff it has a solution: a total, consistent variable assignment φ. We say that φ solves γ.
- **Definition 1.4.** The process of finding solutions to CSPs is called constraint solving.
- *Remark 1.5.* We are using factored representation for world states now.
- Simple example of a *formal representation language*
- Allows useful *general-purpose* algorithms with more power than standard tree search algorithm.

Another Constraint Satisfaction Problem

- **Example 1.6 (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

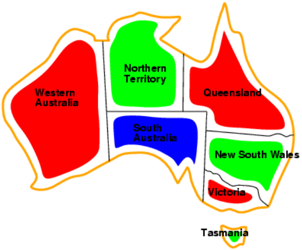
$\leadsto$

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 $1, \ldots, 9$.
  - Constraints: Each number only once in each row, column, block.

CSP Example: Map-Coloring

- **Definition 1.7.** 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 1.8 (Map coloring in Australia).**



- Variables: WA, NT, Q, NSW, V, SA, 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 $\langle\mathrm{WA}, \mathrm{NT}\rangle \in \{\langle\mathrm{red}, \mathrm{green}\rangle, \langle\mathrm{red}, \mathrm{blue}\rangle, \langle\mathrm{green}, \mathrm{red}$

- **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.
- ...

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 [Min]

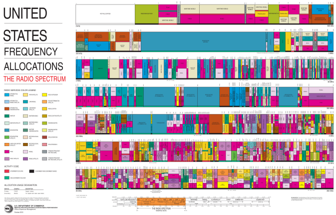
More Constraint Satisfaction Problems

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 [RN03]
- **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.

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.

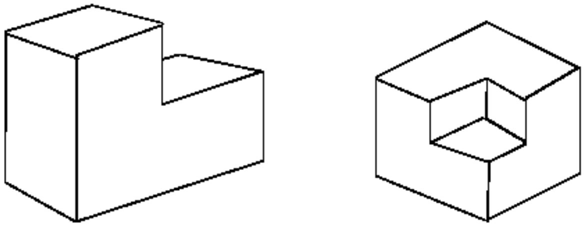
Our Agenda for This Chapter

- 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.

8.2 The Waltz Algorithm

The Waltz Algorithm

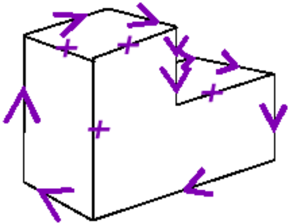
- **Remark:** One of the earliest examples of applied CSPs.
- **Motivation:** Interpret line drawings of polyhedra.

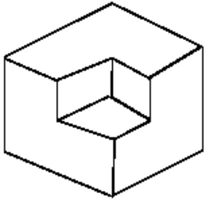


- **Problem:** Are intersections convex or concave? (interpret $\widehat{=}$ label as such)
- **Idea:** Adjacent intersections impose constraints on each other. Use CSP to find a unique set of labelings.

Waltz Algorithm on Simple Scenes

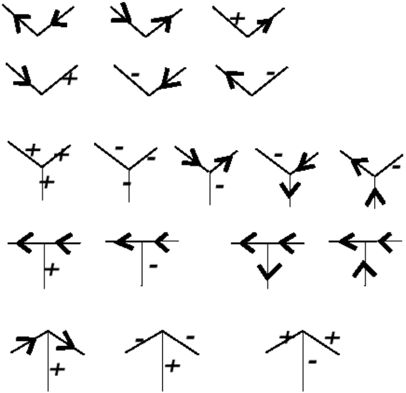
- **Assumptions:** All objects
  - have no shadows or cracks,
  - have only three-faced vertices,
  - are in “general position”, i.e. no junctions change with small movements of the eye.
- **Observation 2.1.** *Then each line on the images is one of the following:*
  - *a boundary line (edge of an object) (<) with right hand of arrow denoting “solid” and left hand denoting “space”*
  - *an interior convex edge* (label with “+”)
  - *an interior concave edge* (label with “-”)





18 Legal Kinds of Junctions

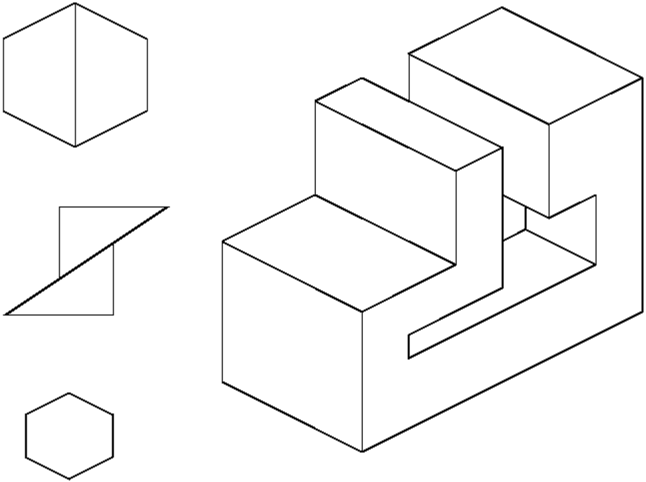
- **Observation 2.2.** *There are only 18 "legal" kinds of junctions:*



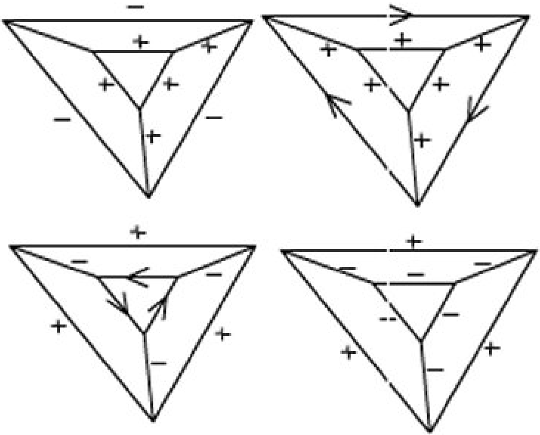
- **Idea:** given a representation of a diagram
  - label each junction in one of these manners (lots of possible ways)
  - junctions must be labeled, so that lines are labeled consistently
- **Fun Fact:** CSP always works perfectly! (early success story for CSP [Wal75])

Waltz's Examples

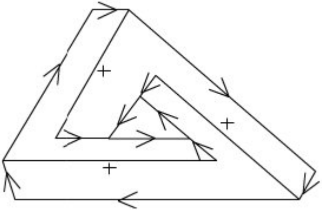
- In his dissertation 1972 [Wal75] David Waltz used the following examples



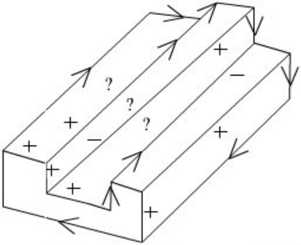
Waltz Algorithm (More Examples): Ambiguous Figures



Waltz Algorithm (More Examples): Impossible Figures



Consistent labelling for impossible figure



No consistent labelling possible

8.3 CSP: Towards a Formal Definition

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 complete)
  - 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.*

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 constraint) (e.g., red is better than green) are often representable by a cost for each variable assignment $\leadsto$ constrained optimization problems.

Non-Binary Constraints, e.g. "Send More Money"

- **Example 3.10 (Send More Money).** A student writes home:

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

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.

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.

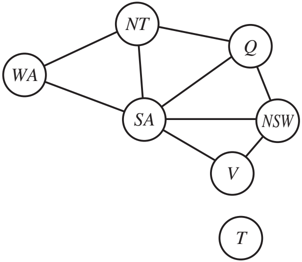
- **General Recipe:** For $n \geq 3$, encode $C(v_1, \ldots, v_{n-1}, v_n)$ as

$$C(p_1(x), \ldots, p_{n-1}(x), v_n) \wedge v_1 = p_1(x) \wedge \ldots \wedge v_{n-1} = p_{n-1}(x)$$

- **Problem:** The problem structure gets hidden. (search algorithms can get confused)

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!)

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.

8.4 Constrain Networks: Formalizing Binary CSPs

Constraint Networks (Formalizing binary CSPs)

- **Definition 4.1.** A constraint network is a triple $\langle V, D, C\rangle$, where
  - V is a finite set of variables,
  - $D{:=}\{D_v | v{\in}V\}$ the set of their domains, and
  - $C{:=}\{C_{uv} \subseteq D_u{\times}D_v | u, v{\in}V \text{ and } u \neq v\}$ is a set of constraints with $C_{uv} = C_{vu}^{-1}$.

  We call the undirected graph $\langle V, \{(u,v){\in}V^2 | C_{uv} \neq D_u \times D_v\}\rangle$, the constraint graph of γ.
- We will talk of CSPs and mean constraint networks.
- **Remarks:** The mathematical formulation gives us a lot of leverage:
  - $C_{uv} \subseteq D_u{\times}D_v \;\widehat{=}$ 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 $C_{uv} = \{(a,b) | a = b\}$ or "$u \neq v$".
  - We can express unary constraints C_u by restricting the domain of v: $D_v{:=}C_v$.

Example: SuDoKu as a Constraint Network

- **Example 4.2 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as 1.6.

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 1.6, but in constraint networks we have a language to formulate things precisely.

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 1.6.

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: $V = \{v_{ij} | 1 \leq i, j \leq 9\}$: v_{ij} =cell row i column j.
  - Domains

Note that the ideas are still the same as 1.6, but in constraint networks we have a language to formulate things precisely.

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 1.6.

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: $V = \{v_{ij} | 1 \leq i, j \leq 9\}$: v_{ij} =cell 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 1.6, but in constraint networks we have a language to formulate things precisely.

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 1.6.

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: $V = \{v_{ij} | 1 \leq i, j \leq 9\}$: v_{ij} =cell 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 1.6, but in constraint networks we have a language to formulate things precisely.

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 1.6.

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: $V = \{v_{ij} | 1 \leq i,j \leq 9\}$: v_{ij} =cell row i column j.
  - Domains
  - Unary constraint:
  - (Binary) constraint: $C_{v_{ij}v_{i'j'}} \mathrel{\widehat{=}}$ "$v_{ij} \neq v_{i'j'}$", i.e. $C_{v_{ij}v_{i'j'}} = \{(d,d') \in D \times D | d \neq d'\}$, 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 1.6, but in constraint networks we have a language to formulate things precisely.

Constraint Networks (Solutions)

- Let $\gamma := \langle V, D, C \rangle$ be a constraint network.
- **Definition 4.7.** We call a partial function $a\colon V \rightharpoonup \bigcup_{u \in V} D_u$ a variable assignment if $a(v) \in D_v$ for all $v \in \mathrm{dom}(a)$.
- **Definition 4.8.** Let $\mathcal{C} := \langle V, D, C \rangle$ 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 4.9.** The empty assignment ϵ is (trivially) consistent in any constraint network.
- **Definition 4.10.** 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 4.11.** We call a consistent (total) assignment a solution for γ and γ itself solvable or satisfiable.

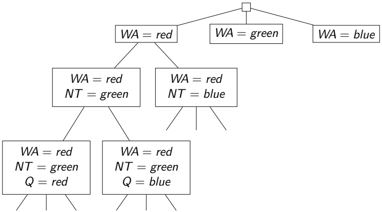
How it all fits together

- **Lemma 4.12.** *Higher-order constraints can be transformed into equi-satisfiable binary constraints using auxiliary variables.*
- **Corollary 4.13.** *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 4.14.** *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.

8.5 CSP as Search

Standard search formulation (incremental)

- **Idea:** Every constraint network induces a single state problem.
- State are defined by the values assigned so far
  - States are variable assignments
  - Initial state: the empty assignment, $\emptyset$
  - Actions: extend current assignment a by a pair (x,v) that does not conflicted with a.
  - $\rightsquigarrow$ fail if no consistent assignments exist (not fixable!)
  - Goal test: the current assignment is total.



- **Remark:** This is the same for all CSPs! ☺
- **Observation:** Every solution appears at depth n with n variables.
- **Idea:** Use depth first search!
- Path is irrelevant, so can also use complete-state formulation
- Branching factor $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!! ☹

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 5.1.** 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 $\approxeq n = 25$.

Backtracking Search (Implementation)

- **Definition 5.2.** The generic backtracking search algorithm

```
procedure Backtracking−Search(csp ) returns solution/failure
    return Recursive−Backtracking (∅, 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 ≠ failure then return result
      remove {var= value} from assignment
  return failure
```

Backtracking in Australia

- **Example 5.3.** We apply backtracking search for a map coloring problem:

Step 1:

Backtracking in Australia

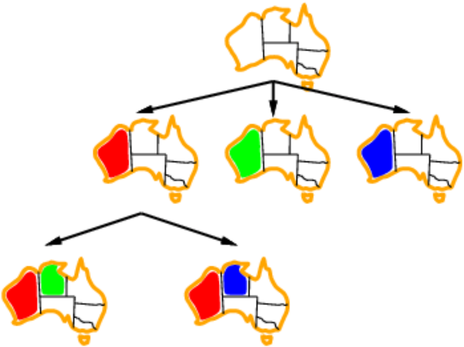
- **Example 5.4.** We apply backtracking search for a map coloring problem:

Step 2:

Backtracking in Australia

- **Example 5.5.** We apply backtracking search for a map coloring problem:

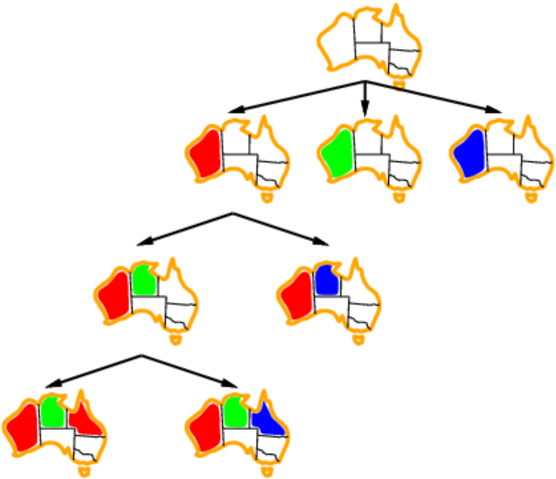
Step 3:



Backtracking in Australia

- **Example 5.6.** We apply backtracking search for a map coloring problem:

Step 4:

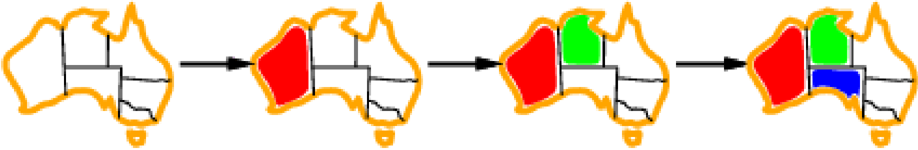


Improving Backtracking Efficiency

- 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? ($\leadsto$ inference)
- **Observation:** Questions 1/2 correspond to the missing subroutines Select−Unassigned−Variable and Order−Domain−Values from 5.2.

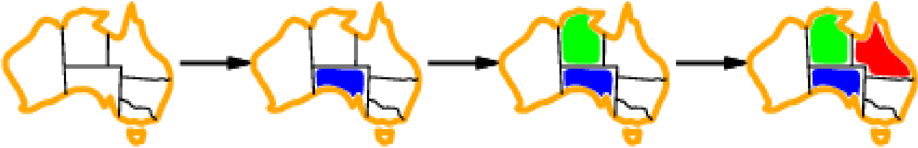
Heuristic: Minimum Remaining Values (Which Variable)

- **Definition 5.7.** 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 $\#(\{d \in D_v | a \cup \{v \mapsto d\} \text{ is consistent}\})$.
- **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 $\#(\{d \in D_v | a \cup \{v \mapsto d\} \text{ is consistent}\}) = 1$, then the value assignment to v is forced by our previous choices.
- **Example 5.8.** In step 3 of 5.3, there is only one remaining value for SA!



Degree Heuristic (Variable Order Tie Breaker)

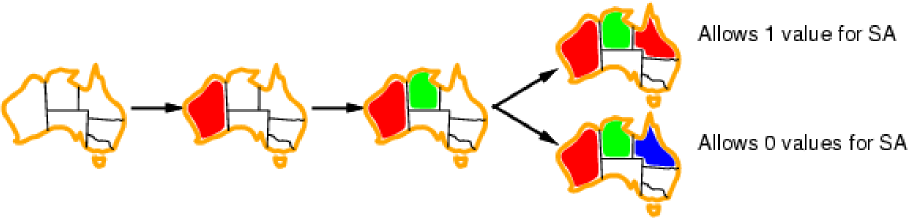
- **Problem:** Need a tie-breaker among MRV variables!(there was no preference in step 1,2)
- **Definition 5.9.** The degree heuristic in backtracking search always chooses a most constraining variable, i.e. given an initial assignment a always pick a variable v with $\#(\{v \in (V \backslash \mathrm{dom}(a)) | C_{uv} \in C\})$ 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 5.10.**



Degree heuristic: $\mathrm{SA} = 5$, $\mathrm{T} = 0$, all others 2 or 3.

Least Constraining Value Heuristic (Value Ordering)

- **Definition 5.11.** 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 $\#(\{e \in D_u | u \notin D_v, C_{uv} \in C, \text{ and } (e,d) \notin C_{uv}\})$
- By choosing the least constraining value first, we increase the chances to not rule out the solutions below the current node.
- **Example 5.12.**



- Combining these heuristics makes 1000 queens feasible.

8.6 Conclusion & Preview

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.

Chapter 9
Constraint Propagation

9.1 Introduction

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!

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.

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.

9.2 Constraint Propagation/Inference

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

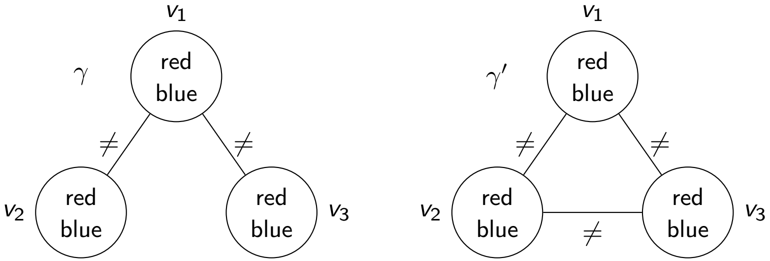
- **Formally:** Replace γ by an equivalent and strictly tighter constraint network γ'.

Equivalent Constraint Networks

- **Definition 2.3.** We say that two constraint networks $\gamma:=\langle V, D, C\rangle$ 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.

Equivalent Constraint Networks

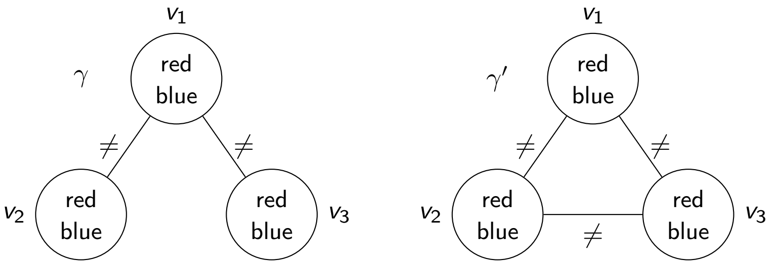
- **Definition 2.5.** We say that two constraint networks $\gamma{:=}\langle V, D, C\rangle$ and $\gamma'{:=}\langle V, D', C'\rangle$ sharing the same set of variables are equivalent, (write $\gamma'{\equiv}\gamma$), if they have the same solutions.
- **Example 2.6.**



Are these constraint networks equivalent?

Equivalent Constraint Networks

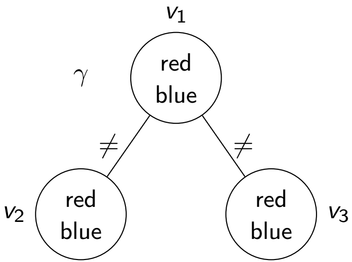
- **Definition 2.7.** We say that two constraint networks $\gamma{:=}\langle V, D, C\rangle$ and $\gamma'{:=}\langle V, D', C'\rangle$ sharing the same set of variables are equivalent, (write $\gamma'{\equiv}\gamma$), if they have the same solutions.
- **Example 2.8.**



Are these constraint networks equivalent? No.

Equivalent Constraint Networks

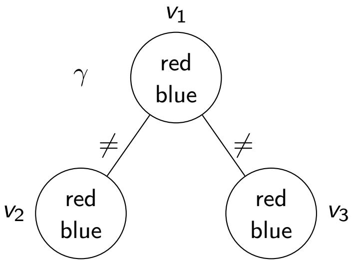
- **Definition 2.9.** We say that two constraint networks $\gamma:=\langle V, D, C\rangle$ and $\gamma':=\langle V, D', C'\rangle$ sharing the same set of variables are equivalent, (write $\gamma'\equiv\gamma$), if they have the same solutions.
- **Example 2.10.**



Are these constraint networks equivalent?

Equivalent Constraint Networks

- **Definition 2.11.** We say that two constraint networks $\gamma{:=}\langle V, D, C\rangle$ 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.

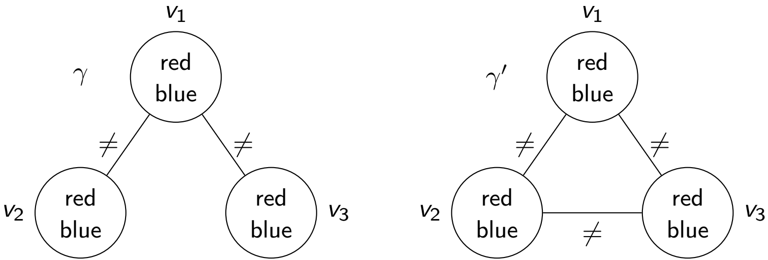
Tightness

- **Definition 2.13 (Tightness).** Let $\gamma{:=}\langle V, D, C\rangle$ and $\gamma' = \langle V, D', C'\rangle$ 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.

Tightness

- **Definition 2.15 (Tightness).** Let $\gamma{:=}\langle V, D, C\rangle$ and $\gamma' = \langle V, D', C'\rangle$ 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.**

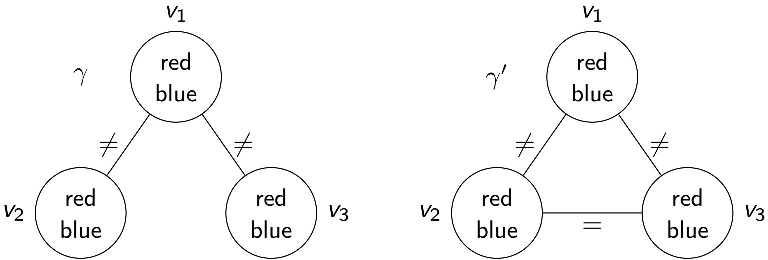


Here, we do have $\gamma'\sqsubseteq\gamma$.

Tightness

- **Definition 2.17 (Tightness).** Let $\gamma{:=}\langle V, D, C\rangle$ and $\gamma' = \langle V, D', C'\rangle$ 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.**

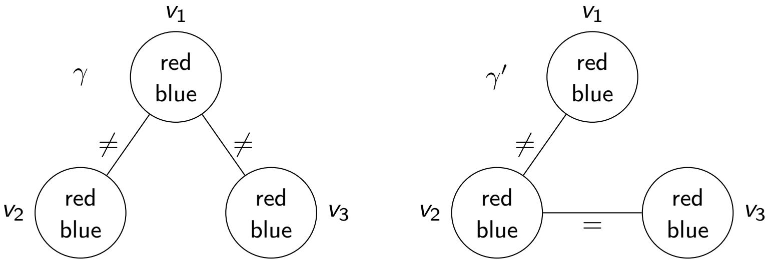


Here, we do have $\gamma'\sqsubseteq\gamma$.

Tightness

- **Definition 2.19 (Tightness).** Let $\gamma := \langle V, D, C\rangle$ and $\gamma' = \langle V, D', C'\rangle$ 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.**

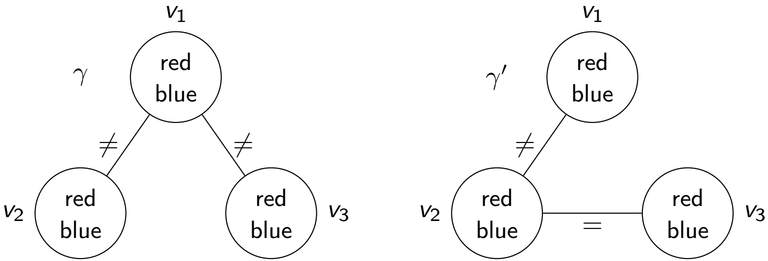


Here, we do not have $\gamma' \sqsubseteq \gamma$!.

Tightness

- **Definition 2.21 (Tightness).** Let $\gamma := \langle V, D, C\rangle$ and $\gamma' = \langle V, D', C'\rangle$ 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.**

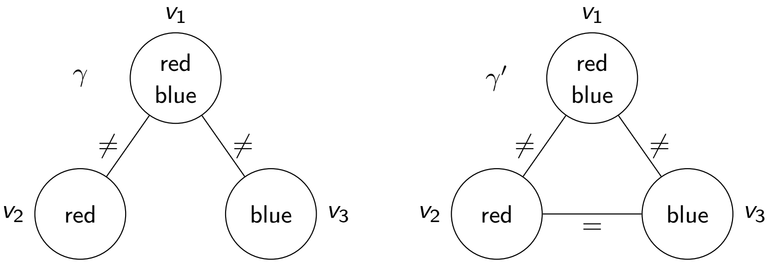


Here, we do not have $\gamma' \sqsubseteq \gamma$!.

- **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.

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*.

Backtracking With Inference

- **Definition 2.25.** The general algorithm for backtracking with inference is

1 **function** BacktrackingWithInference(γ,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   γ' := a copy of γ /* $\gamma' = (V_{\gamma'}, D_{\gamma'}, C_{\gamma'})$ */
5   γ' := **Inference**(γ')
6   **if** exists v with $D_v = \emptyset$ **then return** ‘‘inconsistent’’
7   select some variable v **for** which a is not defined
8   **for** each $d \in$ copy of D_v **in** some order **do**
9     $a' := a \cup \{v = d\}$; $D_v := \{d\}$ /* makes a explicit as a constraint */
10   a'' := BacktrackingWithInference(γ',a')
11   **if** $a'' \neq$ “inconsistent” **then return** a''
12   **return** ‘‘inconsistent’’

- Exactly the same as 5.2, only line 5 new!
- **Inference**(): Any procedure delivering a (tighter) equivalent network.
- **Inference**() typically prunes domains; indicate unsolvability by $D_v = \emptyset$.
- When backtracking out of a search branch, retract the inferred constraints: these were dependent on a, the search commitments so far.

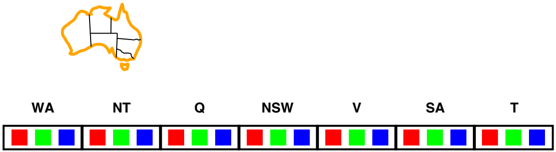
9.3 Forward Checking

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.

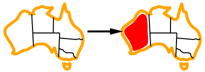
Forward Checking

- **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



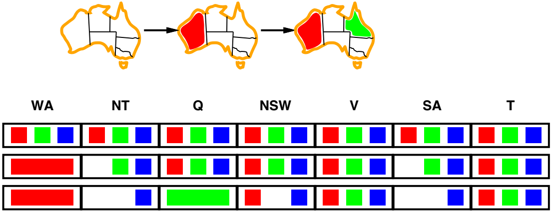
Forward Checking

- **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



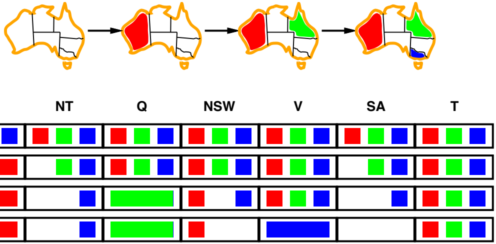
Forward Checking

- **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



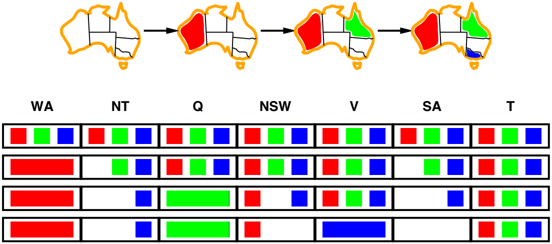
Forward Checking

- **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



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



- **Definition 3.18 (Inference, Version 1).** Forward checking implemented

function ForwardChecking(γ,a) **returns** modified γ
  for each v where $a(v) = d'$ is defined **do**
    for each u where $a(u)$ is undefined and $C_{uv} \in C$ **do**
      $D_u := \{d \in D_u | (d,d') \in C_{uv}\}$
  return γ

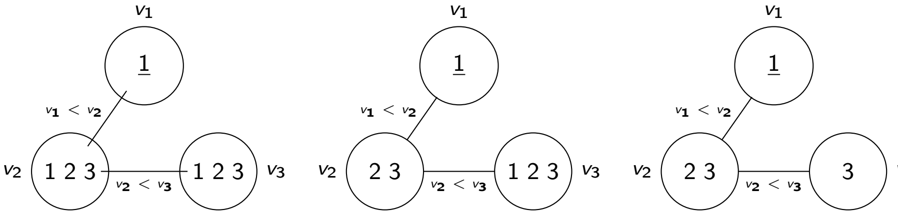
Forward Checking: Discussion

- **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.

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.

Arc Consistency: Definition

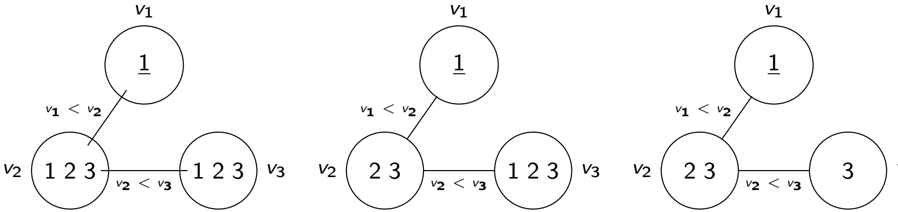
- **Definition 4.2 (Arc Consistency).** Let $\gamma:=\langle V, D, C\rangle$ be a constraint network.
  1. A variable $u\in V$ is arc consistent relative to another variable $v\in V$ if either $C_{uv}\notin C$, or for every value $d\in D_u$ there exists a value $d'\in D_v$ such that $(d,d')\in C_{uv}$.
  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.

Arc Consistency: Example

- **Definition 4.3 (Arc Consistency).** Let $\gamma := \langle V, D, C \rangle$ 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).**

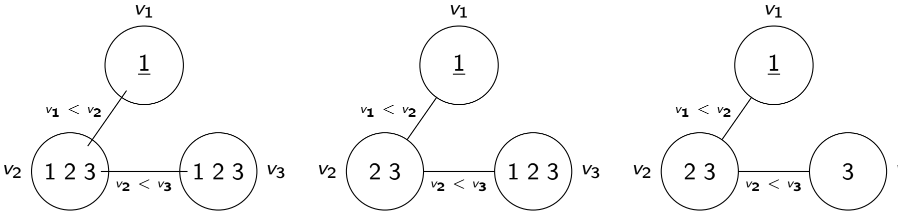


  - **Question**: On top, middle, is v_3 arc consistent relative to v_2?

Arc Consistency: Example

- **Definition 4.5 (Arc Consistency).** Let $\gamma{:=}\langle V, D, C\rangle$ 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).**

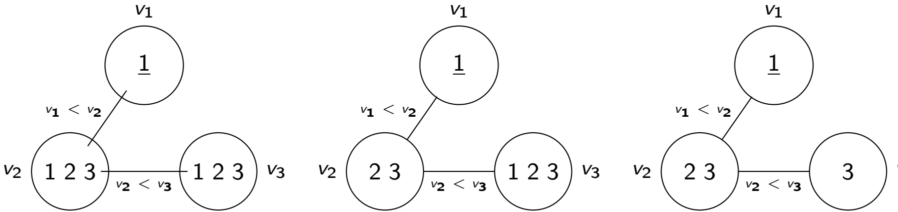


  - **Question**: On top, middle, is v_3 arc consistent relative to v_2?
  - **Answer**: No. For values 1 and 2, D_{v_2} does not have a value that works.
  - **Note**: Enforcing arc consistency for one variable may lead to further reductions on another variable!
  - **Question**: And on the right?

Arc Consistency: Example

- **Definition 4.7 (Arc Consistency).** Let $\gamma{:=}\langle V, D, C\rangle$ 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).**

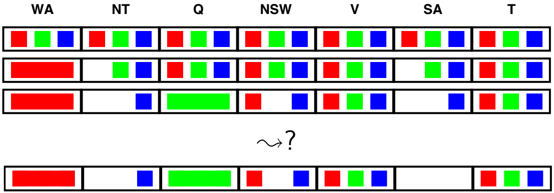


  - **Question**: On top, middle, is v_3 arc consistent relative to v_2?
  - **Answer**: No. For values 1 and 2, D_{v_2} does not have a value that works.
  - **Note**: Enforcing arc consistency for one variable may lead to further reductions on another variable!
  - **Question**: And on the right?
  - **Answer**: Yes. (But v_2 is not arc consistent relative to v_3)

Arc Consistency: Example

- **Definition 4.9 (Arc Consistency).** Let $\gamma:=\langle V, D, C\rangle$ 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.**



  - **Note**: SA is not arc consistent relative to NT in 3rd row.

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 279 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.

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(γ,u,v) **returns** modified γ
  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 γ

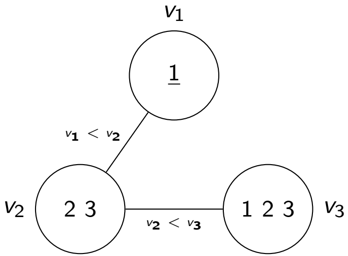
- **Lemma 4.14.** *If d is maximal domain size in γ and the test "$(d,d') \in C_{uv}$?" has time complexity $\mathcal{O}(1)$, then the running time of $\mathrm{Revise}(\gamma, u, v)$ is $\mathcal{O}(d^2)$.*

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(γ,u,v) **returns** modified γ
    for each $d \in D_u$ **do**
      if there is no $d' \in D_v$ with $(d,d') \in C_{uv}$ **then** $D_u := D_u \setminus \{d\}$
    return γ

- **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.** $\mathrm{Revise}(\gamma, v_3, v_2)$

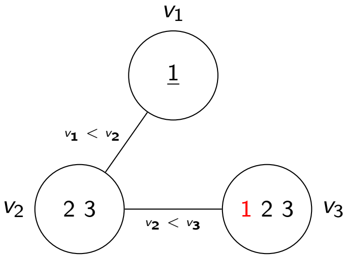


Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.19 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

function Revise(γ,u,v) **returns** modified γ
  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 γ

- **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.** $\mathrm{Revise}(\gamma, v_3, v_2)$

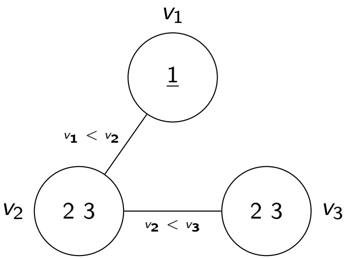


Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.22 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

function Revise(γ,u,v) **returns** modified γ
  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 γ

- **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.** $\mathrm{Revise}(\gamma, v_3, v_2)$

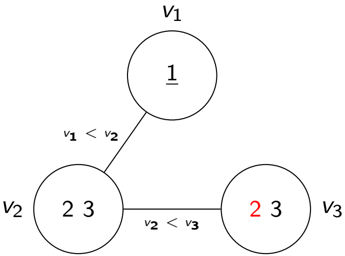


Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.25 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

function Revise(γ,u,v) **returns** modified γ
  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 γ

- **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.** $\mathrm{Revise}(\gamma, v_3, v_2)$

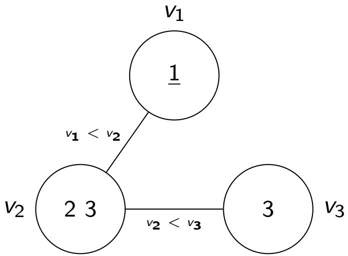


Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.28 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

  function Revise(γ,u,v) **returns** modified γ
    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 γ

- **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.** $\mathrm{Revise}(\gamma, v_3, v_2)$



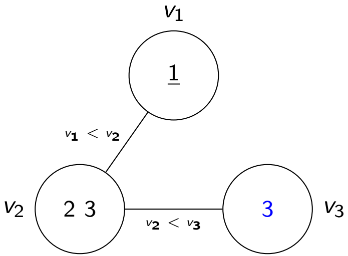
Enforcing Arc Consistency for *One* Pair of Variables

- **Definition 4.31 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

  function Revise(γ,u,v) **returns** modified γ
    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 γ

- **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(γ) returns modified γ
  repeat
    changesMade := False
    for each constraint C_{u0}v do
      Revise(γ,u,v) /* if D_u reduces, set changesMade := True */
      Revise(γ,v,u) /* if D_v reduces, set changesMade := True */
  until changesMade = False
  return γ
```

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(γ) returns modified γ
  repeat
    changesMade := False
    for each constraint C_u0v do
      Revise(γ,u,v) /∗ if D_u reduces, set changesMade := True ∗/
      Revise(γ,v,u) /∗ if D_v reduces, set changesMade := True ∗/
  until changesMade = False
  return γ
```

- **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.

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(γ) **returns** modified γ
  repeat
    changesMade := False
    for each constraint $C_{u0}v$ **do**
      Revise(γ,u,v) /∗ if D_u reduces, set changesMade := True ∗/
      Revise(γ,v,u) /∗ if D_v reduces, set changesMade := True ∗/
  until changesMade = False
  return γ

- **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?

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(γ) returns modified γ
  repeat
    changesMade := False
    for each constraint C_{u0}v do
      Revise(γ,u,v) /∗ if D_u reduces, set changesMade := True ∗/
      Revise(γ,v,u) /∗ if D_v reduces, set changesMade := True ∗/
  until changesMade = False
  return γ
```

- **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)

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(γ) returns modified γ
  M := ∅
  for each constraint C_uv ∈ C do
    M := M ∪ {(u,v),(v,u)}
  while M ≠ ∅ do
    remove any element (u,v) from M
    Revise(γ, u, v)
    if D_u has changed in the call to Revise then
      for each constraint C_wu ∈ C where w ≠ v do
        M := M ∪ {(w,u)}
  return γ
```

- **Question:** $\mathrm{AC}-3(\gamma)$ enforces arc consistency because?

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(γ) returns modified γ
  M := ∅
  for each constraint C_uv ∈ C do
    M := M ∪ {(u,v), (v,u)}
  while M ≠ ∅ do
    remove any element (u,v) from M
    Revise(γ, u, v)
    if D_u has changed in the call to Revise then
      for each constraint C_wu ∈ C where w ≠ v do
        M := M ∪ {(w,u)}
  return γ
```

- **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 $w \neq v$"?

AC-3: Enforcing Arc Consistency (Version 3)

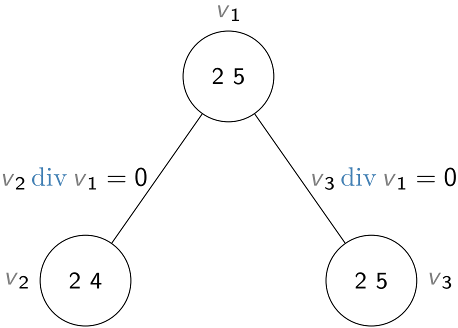
- **Idea:** Remember the potentially inconsistent variable pairs.
- **Definition 4.44.** AC-3 optimizes AC-1 for enforcing arc consistency.

```
function AC−3(γ) returns modified γ
  M := ∅
  for each constraint C_uv ∈ C do
    M := M ∪ {(u,v), (v,u)}
  while M ≠ ∅ do
    remove any element (u,v) from M
    Revise(γ, u, v)
    if D_u has changed in the call to Revise then
      for each constraint C_wu ∈ C where w ≠ v do
        M := M ∪ {(w,u)}
  return γ
```

- **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.

AC-3: Example

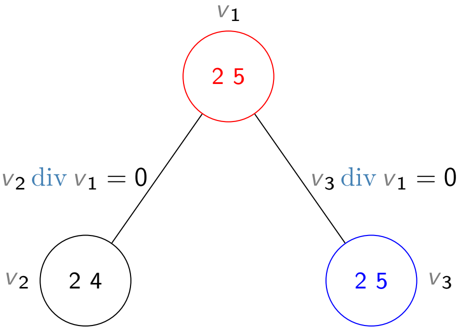
- **Example 4.45.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



M
(v_2, v_1)
(v_1, v_2)
(v_3, v_1)
(v_1, v_3)

AC-3: Example

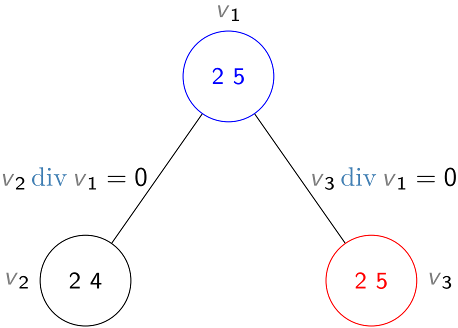
- **Example 4.46.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



M
(v_2, v_1)
(v_1, v_2)
(v_3, v_1)
(v_1, v_3)

AC-3: Example

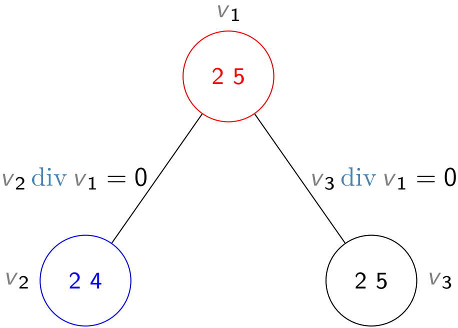
- **Example 4.47.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



M
(v_2, v_1)
(v_1, v_2)
(v_3, v_1)

AC-3: Example

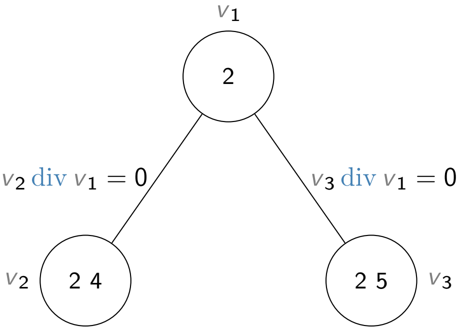
- **Example 4.48.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



M
(v_2, v_1)
(v_1, v_2)

AC-3: Example

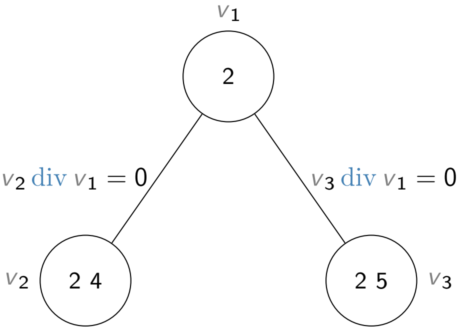
- **Example 4.49.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



$$\frac{M}{(v_2, v_1)}$$

AC-3: Example

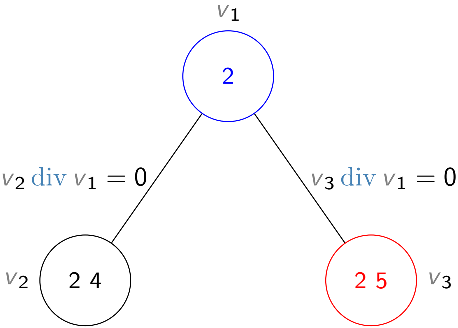
- **Example 4.50.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



M
(v_2, v_1)
(v_3, v_1)

AC-3: Example

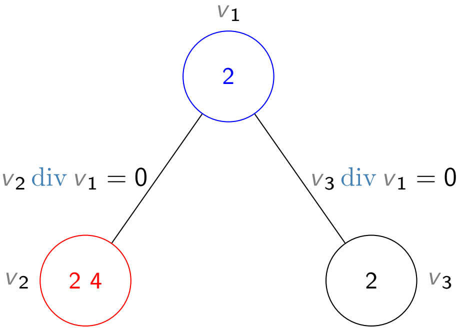
- **Example 4.51.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



M
(v_2, v_1)
(v_3, v_1)

AC-3: Example

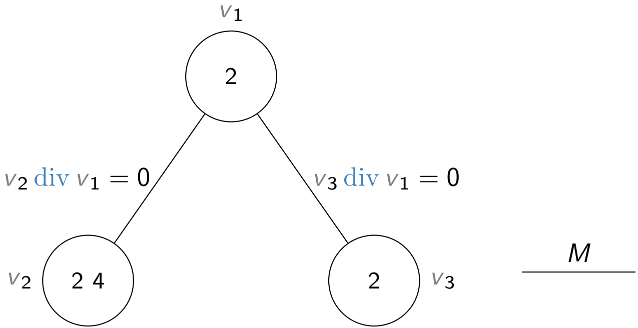
- **Example 4.52.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x



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



AC-3: Runtime

- **Theorem 4.54 (Runtime of AC-3).** *Let $\gamma := \langle V, D, C \rangle$ 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.

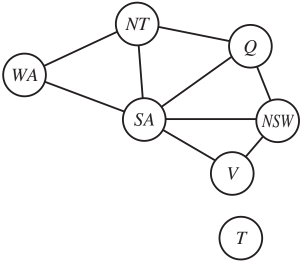
9.5 Decomposition: Constraint Graphs, and Three Simple Cases

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"?

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

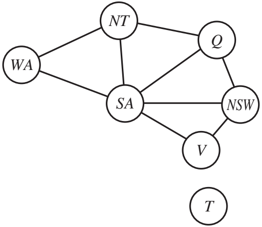
- **Theorem 5.2 (Disconnected Constraint Graphs).** *Let $\gamma := \langle V, D, C \rangle$ be a constraint network. Let a_i be a solution to each connected component γ_i of the constraint graph of γ. Then $a := \bigcup_i a_i$ is a solution to γ.*

"Decomposition" 1.0: Disconnected Constraint Graphs

- **Theorem 5.6 (Disconnected Constraint Graphs).** *Let $\gamma := \langle V, D, C \rangle$ 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.

"Decomposition" 1.0: Disconnected Constraint Graphs

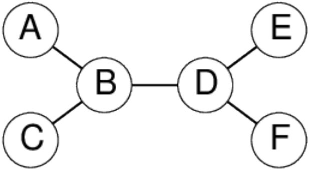
- **Theorem 5.10 (Disconnected Constraint Graphs).** *Let $\gamma := \langle V, D, C \rangle$ be a constraint network. Let a_i be a solution to each connected component γ_i of the constraint graph of γ. Then $a := \bigcup_i a_i$ is a solution to γ.*
- *Proof:*
  1. a satisfies all C_{uv} where u and 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



"Decomposition" 1.0: Disconnected Constraint Graphs

- **Theorem 5.14 (Disconnected Constraint Graphs).** *Let $\gamma := \langle V, D, C \rangle$ 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.

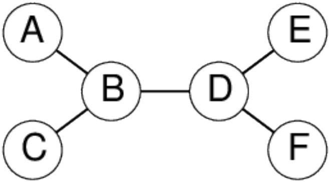
Tree-structured CSPs



- **Theorem 5.18.** *If the constraint graph has no cycles, the 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



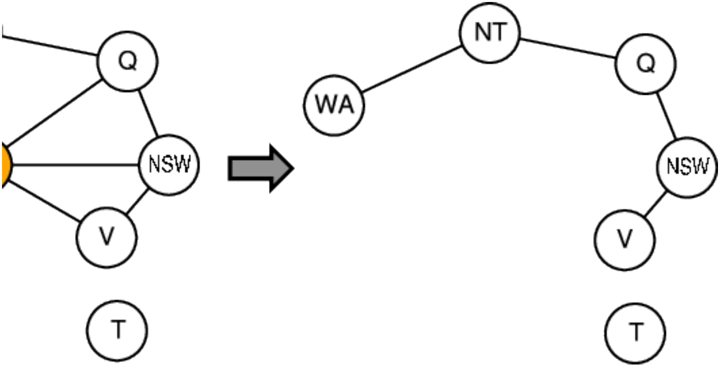
2. For j from n down to 2, apply

    RemoveInconsistent(Parent(X_j,X_j)

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

Nearly tree-structured CSPs

- **Definition 5.19.** Conditioning: instantiate a variable, prune its neighbors' domains.
- **Example 5.20.**



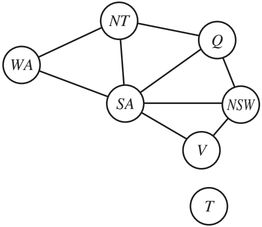
- **Definition 5.21.** Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree.
- Cutset size $c \rightsquigarrow$ running time $\mathcal{O}(d^c(n-c)d^2)$, very fast for small c.

“Decomposition” 2.0: Acyclic Constraint Graphs

- **Theorem 5.22 (Acyclic Constraint Graphs).** *Let $\gamma := \langle V, D, C \rangle$ 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 5.25 (Acyclic Constraint Graphs).** *Let $\gamma:=\langle V, D, C\rangle$ 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 5.26.** Australia is not acyclic. (But see next section)



"Decomposition" 2.0: Acyclic Constraint Graphs

- **Theorem 5.28 (Acyclic Constraint Graphs).** *Let $\gamma := \langle V, D, C \rangle$ 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 5.29.** Australia is not acyclic. (But see next section)
- **Example 5.30 (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}.

Acyclic Constraint Graphs: How To

- **Definition 5.31.** 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.

Acyclic Constraint Graphs: How To

- **Definition 5.33.** 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.

Acyclic Constraint Graphs: How To

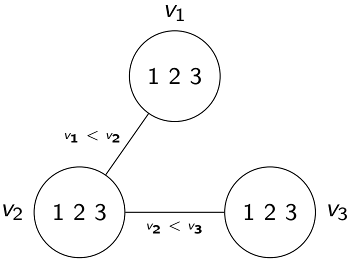
- **Definition 5.35.** Algorithm $\mathrm{AcyclicCG}(\gamma)$:
  1. Obtain a (directed) tree from γ's constraint graph, picking an arbitrary variable v as the root, and directing edges outwards.[1]
  2. Order the variables topologically, i.e., such that each node is ordered before its children; denote 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 5.36.** *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.

AcyclicCG(γ): Example

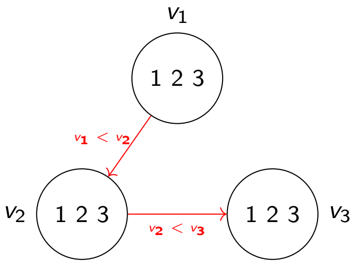
- **Example 5.37 (AcyclicCG() execution).**



Input network γ.

AcyclicCG(γ): Example

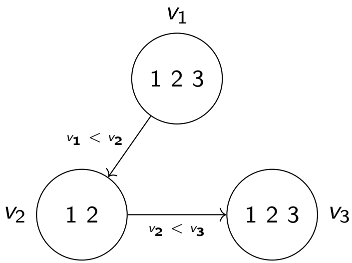
- **Example 5.38 (AcyclicCG() execution).**



Step 1: Directed tree for root v_1.
Step 2: Order v_1, v_2, v_3.

AcyclicCG(γ): Example

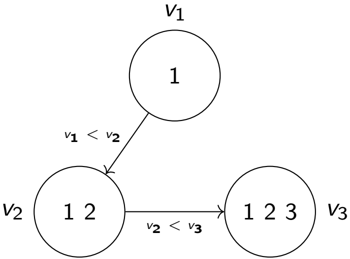
- **Example 5.39 (AcyclicCG() execution).**



Step 3: After Revise(γ, v_2, v_3).

AcyclicCG(γ): Example

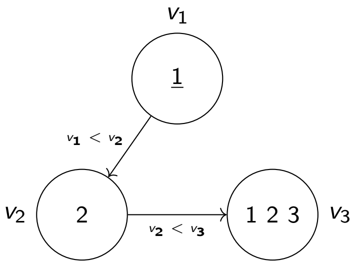
- **Example 5.40 (AcyclicCG() execution).**



Step 3: After Revise(γ, v_1, v_2).

AcyclicCG(γ): Example

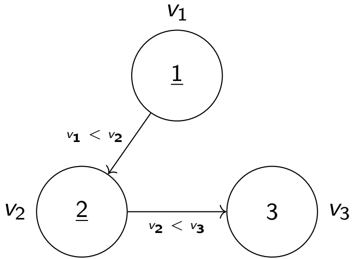
- **Example 5.41 (AcyclicCG() execution).**



Step 4: After $a(v_1) := 1$ and forward checking.

AcyclicCG(γ): Example

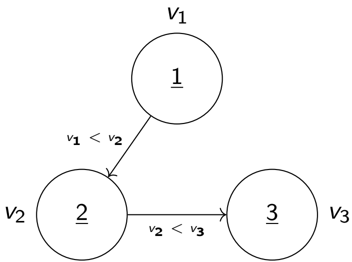
- **Example 5.42 (AcyclicCG() execution).**



Step 4: After $a(v_2) := 2$ and forward checking.

AcyclicCG(γ): Example

- **Example 5.43 (AcyclicCG() execution).**

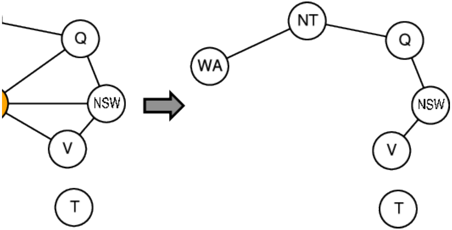


Step 4: After $a(v_3) := 3$ (and forward checking).

9.6 Cutset Conditioning

"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 $\{v \in V | a(v) \text{ is undefined}\}$ 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 . . . !

"Decomposition" 3.0: Cutset Conditioning

- **Definition 6.2 (Cutset).** Let $\gamma := \langle V, D, C \rangle$ 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(γ,V₀,a) returns a solution, or ''inconsistent''
  γ' := a copy of γ; γ' := ForwardChecking(γ',a)
  if ex. v with D_v = ∅ then return ''inconsistent''
  if ex. v∈V₀ s.t. a(v) is undefined then select such v
  else a' := AcyclicCG(γ');
  if a' ≠ "inconsistent" then return a ∪ a' else return ''inconsistent''
  for each d∈copy of D_v in some order do
    a' := a ∪ {v = d}; D_v := {d};
    a'' := CutsetConditioning(γ',V₀,a')
  if a'' ≠ "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.

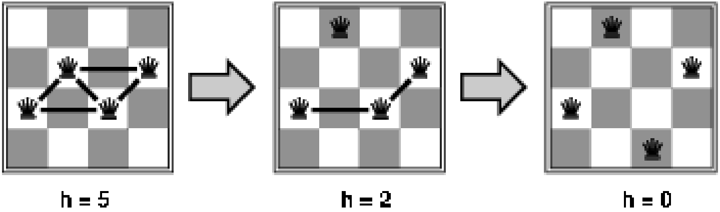
9.7 Constraint Propagation with Local Search

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.

Example: 4-Queens

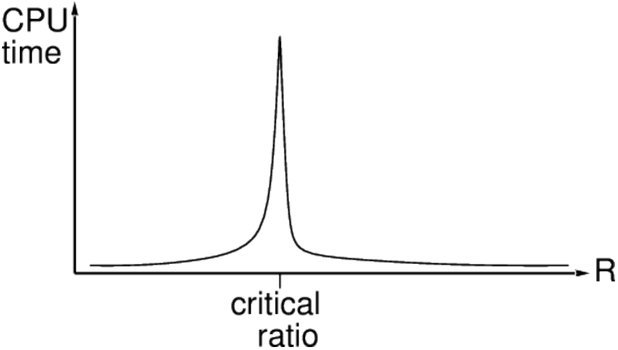
- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Actions: Move queen in column
- Goal state: No conflicts
- Heuristic: $h(n) \widehat{=}$ number of conflict



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

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



9.8 Conclusion & Summary

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.

Topics We Didn't Cover Here

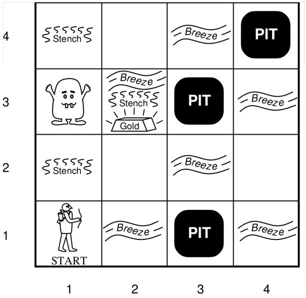
- **Path consistency, k-consistency:** Generalizes arc consistency to size k subsets of variables. Path consistency $\widehat{=}$ 3-consistency.
- **Tree decomposition:** Instead of instantiating variables until the leaf nodes are trees, distribute the variables and constraints over sub-CSPs whose connections 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.

Part 3
Knowledge and Inference

Chapter 10
Propositional Logic & Reasoning, Part I: Principles

10.1 Introduction

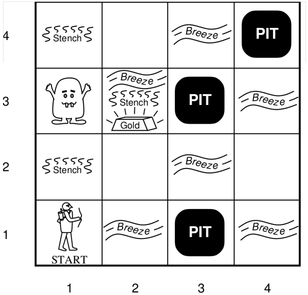
The Wumpus World



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.

- **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

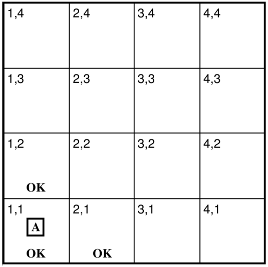


Definition 1.5. The Wumpus world is a simple game where an agent explores a cave with 16 cells that can contain pits, gold, and the Wumpus with the goal of getting back out alive with the gold.

- **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).

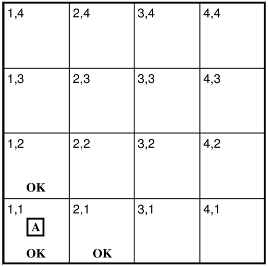
Reasoning in the Wumpus World

- **Example 1.9 (Reasoning in the Wumpus World).** **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.

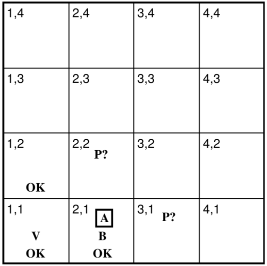


(1) Initial state

- **Example 1.10 (Reasoning in the Wumpus World).** **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



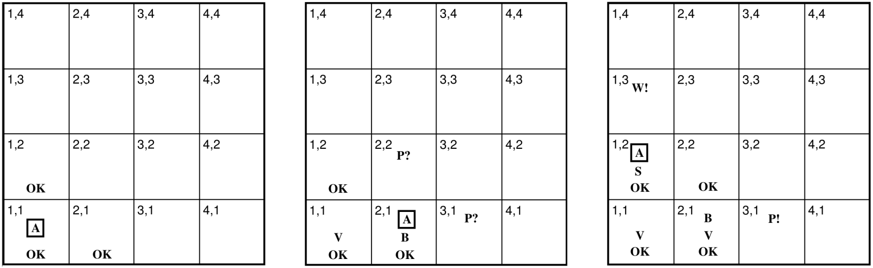
(1) Initial state



(2) One step to right

Reasoning in the Wumpus World

- **Example 1.11 (Reasoning in the Wumpus World).** **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.

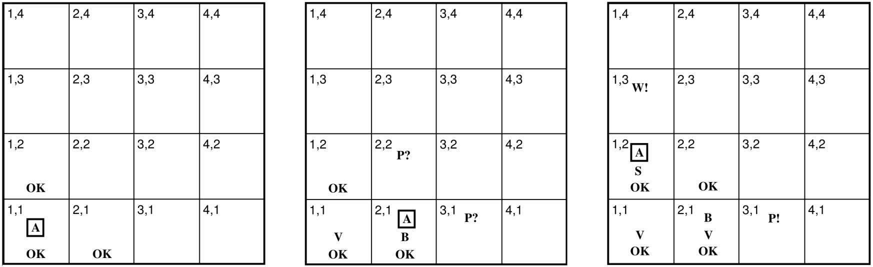


(1) Initial state (2) One step to right (3) Back, and up to [1,2]

  - *The Wumpus is in [1,3]*! How do we know?

Reasoning in the Wumpus World

- **Example 1.12 (Reasoning in the Wumpus World).** **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.

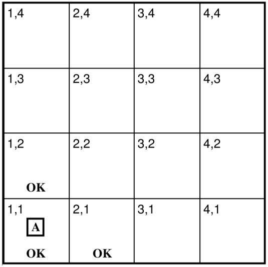


(1) Initial state (2) One step to right (3) Back, and up to [1,2]

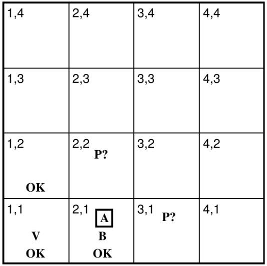
- *The Wumpus is in [1,3]*! How do we know?
- No stench in [2,1], so the stench in [1,2] can only come from [1,3].
- *There's a pit in [3,1]*! How do we know?

Reasoning in the Wumpus World

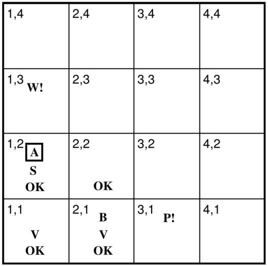
- **Example 1.13 (Reasoning in the Wumpus World).** **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



(1) Initial state



(2) One step to right

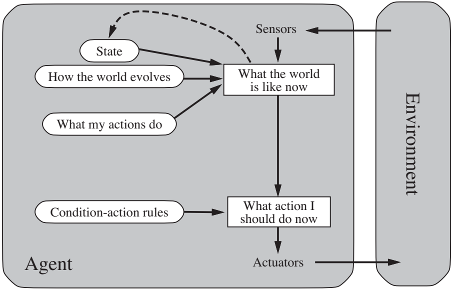


(3) Back, and up to [1,2]

  - *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].

Agents that Think Rationally

- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.
- **Definition 1.14.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about the state of the environment and its own actions.



Agents that Think Rationally

- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.
- **Definition 1.15.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about the state of the environment and its own 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*

Logic: Basic Concepts (Representing Knowledge)

- **Definition 1.16.** Syntax: What are legal statements (formulae) A in the logic?
- **Example 1.17.** "W" and "$W \Rightarrow S$". ($W \mathrel{\widehat{=}}$ *Wumpus is here*, $S \mathrel{\widehat{=}}$ *it stinks*)

Logic: Basic Concepts (Representing Knowledge)

- **Definition 1.20.** Syntax: What are legal statements (formulae) A in the logic?
- **Example 1.21.** "W" and "$W \Rightarrow S$". ($W \widehat{=}$ *Wumpus is here*, $S \widehat{=}$ *it stinks*)
- **Definition 1.22.** Semantics: Which formulas A are true under which assignment φ, written $\varphi \models$A?
- **Example 1.23.** 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!)

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.24.** 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

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.29.** 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 1.30.** Deduction: Which statements B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.31.** 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 1.34.** 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 1.35.** Deduction: Which statements B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.36.** 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.

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.39.** 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 1.40.** Deduction: Which statements B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.41.** 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.
- **Definition 1.42.** Soundness: whenever $A \vdash_{\mathcal{C}} B$, we also have $A \models B$.
- **Definition 1.43.** Completeness: whenever $A \models B$, we also have $A \vdash_{\mathcal{C}} B$.

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;)

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.

Our Agenda for This Topic

- **This section**: 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.

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 methods.
- **Killing a Wumpus**: How can we use all this to figure out where the Wumpus is?
  - Coming back to our introductory example.

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 $\mathit{wff}_0(\mathcal{V}_0)$ of well-formed propositional formula (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 biimplication). A $\Leftrightarrow$ B,

  where A, B$\in \mathit{wff}_0(\mathcal{V}_0)$ themselves.
- **Example 2.2.** $P \wedge Q, P \vee Q, (\neg P \vee Q) \Leftrightarrow (P \Rightarrow Q) \in \mathit{wff}_0(\mathcal{V}_0)$
- **Definition 2.3.** Propositional formulae without connectives are called atomic (or an atom) and complex otherwise.

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
		$\mid$	$\neg A$	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

Alternative Notations for Connectives

Here	Elsewhere
$\neg A$	$\sim A \quad \overline{A}$
$A \wedge B$	$A \,\&\, B \quad A \bullet B \quad A, B$
$A \vee B$	$A + B \quad A \mid B \quad A ; B$
$A \Rightarrow B$	$A \rightarrow B \quad A \supset B$
$A \Leftrightarrow B$	$A \leftrightarrow B \quad A \equiv B$
F	$\bot \quad 0$
T	$\top \quad 1$

Semantics of PL^0 (Models)

- **Definition 2.4.** A model $\mathcal{M}{:=}\langle\mathcal{D}_o,\mathcal{I}\rangle$ for propositional logic consists of
  - the universe $\mathcal{D}_o = \{\mathsf{T},\mathsf{F}\}$
  - the interpretation $\mathcal{I}$ that assigns values to essential connectives.
  - $\mathcal{I}(\neg)\colon \mathcal{D}_o{\rightarrow}\mathcal{D}_o; \mathsf{T}{\mapsto}\mathsf{F}, \mathsf{F}{\mapsto}\mathsf{T}$
  - $\mathcal{I}(\wedge)\colon \mathcal{D}_o \times \mathcal{D}_o{\rightarrow}\mathcal{D}_o; \langle\alpha,\beta\rangle{\mapsto}\mathsf{T}$, iff $\alpha = \beta = \mathsf{T}$

  We call a constructor a logical constant, iff its value is fixed by the interpretation.

- Treat the other connectives as abbreviations, e.g. $A \vee B \widehat{=} \neg(\neg A \wedge \neg B)$ and $A \Rightarrow B \widehat{=} \neg A \vee B$, and $T \widehat{=} P \vee \neg P$ (only need to treat $\neg, \wedge$ directly)

Semantics of PL^0 (Evaluation)

- **Problem:** The interpretation function only assigns meaning to connectives.
- **Definition 2.5.** A variable assignment $\varphi\colon \mathcal{V}_0 \to \mathcal{D}_o$ assigns values to propositional variables.
- **Definition 2.6.** The value function $\mathcal{I}_\varphi\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ 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 that $\mathcal{I}_\varphi(\mathrm{A} \vee \mathrm{B}) = \mathcal{I}_\varphi(\neg(\neg \mathrm{A} \wedge \neg \mathrm{B}))$ 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.
- **Definition 2.7.** Two formulae A and B are called equivalent, iff $\mathcal{I}_\varphi(\mathrm{A}) = \mathcal{I}_\varphi(\mathrm{B})$ for all variable assignments φ.

Computing Semantics

- **Example 2.8.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

  $\mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)$

Computing Semantics

- **Example 2.9.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned} & \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\ = \quad & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \end{aligned}$$

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

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4)))
\end{aligned}
$$

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

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4))))
\end{aligned}$$

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

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4))))
\end{aligned}
$$

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

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F})))
\end{aligned}
$$

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

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F}))
\end{aligned}
$$

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

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F}))
\end{aligned}
$$

Computing Semantics

- **Example 2.16.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F}, \mathsf{F})), \mathsf{F}))
\end{aligned}
$$

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

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \lor P_2 \lor \neg(\neg P_1 \land P_2) \lor P_3 \land P_4) \\
= \; & \mathcal{I}(\lor)(\mathcal{I}_\varphi(P_1 \lor P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \land P_2) \lor P_3 \land P_4)) \\
= \; & \mathcal{I}(\lor)(\mathcal{I}(\lor)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\lor)(\mathcal{I}_\varphi(\neg(\neg P_1 \land P_2)), \mathcal{I}_\varphi(P_3 \land P_4))) \\
= \; & \mathcal{I}(\lor)(\mathcal{I}(\lor)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\lor)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \land P_2)), \mathcal{I}(\land)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\lor)(\mathcal{I}(\lor)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\lor)(\mathcal{I}(\neg)(\mathcal{I}(\land)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\land)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\lor)(\mathsf{T}, \mathcal{I}(\lor)(\mathcal{I}(\neg)(\mathcal{I}(\land)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\land)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\lor)(\mathsf{T}, \mathcal{I}(\lor)(\mathcal{I}(\neg)(\mathcal{I}(\land)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\lor)(\mathsf{T}, \mathcal{I}(\lor)(\mathcal{I}(\neg)(\mathcal{I}(\land)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\lor)(\mathsf{T}, \mathcal{I}(\lor)(\mathcal{I}(\neg)(\mathcal{I}(\land)(\mathsf{F}, \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\lor)(\mathsf{T}, \mathcal{I}(\lor)(\mathcal{I}(\neg)(\mathsf{F}), \mathsf{F}))
\end{aligned}
$$

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

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \quad & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F}, \mathsf{F})), \mathsf{F})) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathsf{F}), \mathsf{F})) \\
= \quad & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}))
\end{aligned}
$$

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

$$
\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F}, \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathsf{F}), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathsf{T}, \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathsf{T})
\end{aligned}
$$

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

$$
\begin{array}{rl}
& \mathcal{I}_\varphi(P_1\vee P_2\vee\neg(\neg P_1\wedge P_2)\vee P_3\wedge P_4)\\
= & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1\vee P_2),\mathcal{I}_\varphi(\neg(\neg P_1\wedge P_2)\vee P_3\wedge P_4))\\
= & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1),\mathcal{I}_\varphi(P_2)),\mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1\wedge P_2)),\mathcal{I}_\varphi(P_3\wedge P_4)))\\
= & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1),\varphi(P_2)),\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1\wedge P_2)),\mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3),\mathcal{I}_\varphi(P_4))))\\
= & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T},\mathsf{F}),\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1),\mathcal{I}_\varphi(P_2))),\mathcal{I}(\wedge)(\varphi(P_3),\varphi(P_4))))\\
= & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)),\varphi(P_2))),\mathcal{I}(\wedge)(\mathsf{T},\mathsf{F})))\\
= & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)),\mathsf{F})),\mathsf{F}))\\
= & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}),\mathsf{F})),\mathsf{F}))\\
= & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F},\mathsf{F})),\mathsf{F}))\\
= & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathsf{F}),\mathsf{F}))\\
= & \mathcal{I}(\vee)(\mathsf{T},\mathcal{I}(\vee)(\mathsf{T},\mathsf{F}))\\
= & \mathcal{I}(\vee)(\mathsf{T},\mathsf{T})\\
= & \mathsf{T}
\end{array}
$$

- What a mess!

Propositional Identities

- We have the following identities in propositional logic:

Name	for $\wedge$	for $\vee$
Idenpotence	$\varphi \wedge \varphi = \varphi$	$\varphi \vee \varphi = \varphi$
Identity	$\varphi \wedge T = \varphi$	$\varphi \vee F = \varphi$
Absorption I	$\varphi \wedge F = F$	$\varphi \vee T = T$
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 II	$\varphi \wedge (\varphi \vee \theta) = \varphi$	$\varphi \vee \varphi \wedge \theta = \varphi$
De Morgan	$\neg(\varphi \wedge \psi) = \neg\varphi \vee \neg\psi$	$\neg(\varphi \vee \psi) = \neg\varphi \wedge \neg\psi$
Double negation	$\neg\neg\varphi = \varphi$	
Definitions	$\varphi \Rightarrow \psi = \neg\varphi \vee \psi$	$\varphi \Leftrightarrow \psi = (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$

Semantic Properties of Propositional Formulae

- **Definition 2.21.** Let $\mathcal{M}:=\langle\mathcal{U},\mathcal{I}\rangle$ be our model, then we call A
  - true under φ (φ satisfies A) in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathrm{A})=\mathsf{T}$, (write $\mathcal{M}\models^\varphi \mathrm{A}$)
  - false under φ (φ falsifies A) 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 2.22.** $x\vee x$ is satisfiable and falsifiable.
- **Example 2.23.** $x\vee\neg x$ is valid and $x\wedge\neg x$ is unsatisfiable.
- **Alternative Notation:** Write $[\![\mathrm{A}]\!]_\varphi^\mathcal{I}$ for $\mathcal{I}_\varphi(\mathrm{A})$, if $\mathcal{M}=\langle\mathcal{U},\mathcal{I}\rangle$.(and $[\![\mathrm{A}]\!]^\mathcal{I}$, if A is ground, and $[\![\mathrm{A}]\!]^\mathcal{I}$, if $\mathcal{M}$ is clear)
- **Definition 2.24 (Entailment).** (aka. logical consequence)
  We say that A entails B ($\mathrm{A}\models\mathrm{B}$), 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)

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

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

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.

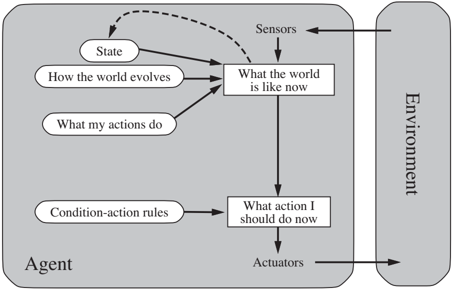
  We first express what we know: For every $x \in \{S, N, J\}$ (Stefan, Nicole, Jochen) we have
  1. $bla(x) \vee red(x) \vee gre(x)$; (note: three formulae)
  2. $ai(x) \vee phy(x) \vee chi(x)$ and $ai(S) \vee ai(N) \vee ai(J)$
     - 2.1 $ai(x) \Rightarrow \neg red(x) \wedge \neg gre(x)$.
     - 2.2 $phy(x) \Rightarrow \neg bla(x)$ 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 3. and 2.2 and
  5. so $\neg ai(S) \wedge \neg ai(J)$ by 3. and 4.
  6. With 2.3 the latter entails $ai(N)$.

10.3 Inference in Propositional Logics

Agents that Think Rationally

- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.
- **Definition 3.1.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about the state of the environment and its own actions.



Agents that Think Rationally

- **Idea:** Think Before You Act!
  "Thinking" = Inference about knowledge represented using logic.
- **Definition 3.2.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about the state of the environment and its own 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
```

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 $\mathcal{I}_\varphi(\mathsf{A} \Rightarrow \mathsf{B}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathsf{A}) = \mathsf{F}$ or $\mathcal{I}_\varphi(\mathsf{B}) = \mathsf{T}$.
- **Definition 3.3.** The Hilbert calculus $\mathcal{H}^0$ consists of the inference rules:

$$\frac{}{P \Rightarrow Q \Rightarrow P}\,\mathrm{K} \qquad \frac{}{(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R}\,\mathrm{S}$$

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

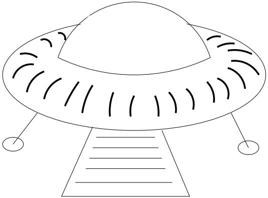
- **Example 3.4.** A $\mathcal{H}^0$ theorem $\mathsf{C} \Rightarrow \mathsf{C}$ and its proof

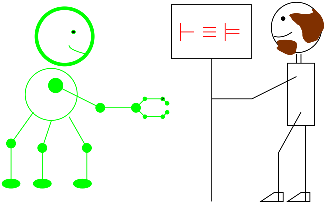
Proof: We show that $\emptyset \vdash_{\mathcal{H}^0} \mathsf{C} \Rightarrow \mathsf{C}$

1. $(\mathsf{C} \Rightarrow (\mathsf{C} \Rightarrow \mathsf{C}) \Rightarrow \mathsf{C}) \Rightarrow (\mathsf{C} \Rightarrow \mathsf{C} \Rightarrow \mathsf{C}) \Rightarrow \mathsf{C} \Rightarrow \mathsf{C}$ (S with $[\mathsf{C}/P], [\mathsf{C} \Rightarrow \mathsf{C}/Q], [\mathsf{C}/R]$)
2. $\mathsf{C} \Rightarrow (\mathsf{C} \Rightarrow \mathsf{C}) \Rightarrow \mathsf{C}$ (K with $[\mathsf{C}/P], [\mathsf{C} \Rightarrow \mathsf{C}/Q]$)
3. $(\mathsf{C} \Rightarrow \mathsf{C} \Rightarrow \mathsf{C}) \Rightarrow \mathsf{C} \Rightarrow \mathsf{C}$ (MP on P.1 and P.2)
4. $\mathsf{C} \Rightarrow \mathsf{C} \Rightarrow \mathsf{C}$ (K with $[\mathsf{C}/P], [\mathsf{C}/Q]$)
5. $\mathsf{C} \Rightarrow \mathsf{C}$ (MP on P.3 and P.4)

Soundness and Completeness

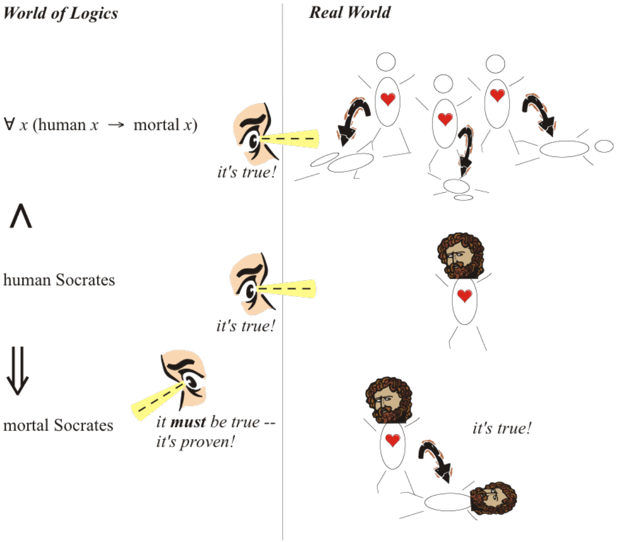
- **Definition 3.5.** Let $\mathcal{L}{:=}\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system, then we call a calculus $\mathcal{C}$ for $\mathcal{L}$,
  - sound (or correct), iff $\mathcal{H}\models\mathrm{A}$, whenever $\mathcal{H}\vdash_{\mathcal{C}}\mathrm{A}$, and
  - complete, iff $\mathcal{H}\vdash_{\mathcal{C}}\mathrm{A}$, 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])

    - 



The miracle of logics

- Purely formal derivations are true in the real world!



10.4 Propositional Natural Deduction Calculus

Calculi: Natural Deduction ($\mathcal{ND}_0$; Gentzen [Gen34])

- **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
$\dfrac{A \quad B}{A \wedge B}\ \mathcal{ND}_0{\wedge}I$	$\dfrac{A \wedge B}{A}\ \mathcal{ND}_0{\wedge}E_l$	$\dfrac{A \wedge B}{B}\ \mathcal{ND}_0{\wedge}E_r$	$\dfrac{}{A \vee \neg A}\ \mathcal{ND}_0\mathrm{TND}$
$\dfrac{\begin{array}{c}[A]^1\\ \hline\hline \\ B\end{array}}{A \Rightarrow B}\ \mathcal{ND}_0{\Rightarrow}I^1$	$\dfrac{A \Rightarrow B \quad A}{B}\ \mathcal{ND}_0{\Rightarrow}E$		

$\Rightarrow I$ 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; $\Rightarrow I$ then discharges (get rid of A, which can only be used in $\mathcal{D}$) the 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:** $\mathcal{ND}_0\mathrm{TND}$ is used only in classical logic. (otherwise constructive/intuitionistic)

Natural Deduction: Examples

- **Example 4.3 (Inference with Local Hypotheses).**

$$\dfrac{\dfrac{[A \wedge B]^1}{B}\,\mathcal{ND}_0{\wedge}E_r \quad \dfrac{[A \wedge B]^1}{A}\,\mathcal{ND}_0{\wedge}E_l}{\dfrac{B \wedge A}{A \wedge B \Rightarrow B \wedge A}\,\mathcal{ND}_0{\Rightarrow}I^1}\,\mathcal{ND}_0{\wedge}I$$

$$\begin{array}{c} [A]^1 \\ [B]^2 \\ \dfrac{\dfrac{A}{B \Rightarrow A}\,\mathcal{ND}_0{\Rightarrow}I^2}{A \Rightarrow B \Rightarrow A}\,\mathcal{ND}_0{\Rightarrow}I^1 \end{array}$$

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 $\mathcal{ND}_0 \Rightarrow I$, 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 $\mathcal{ND}_0 \Rightarrow E$.

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.

$$\frac{A}{A \vee B}\ \mathcal{ND}_0{\vee}I_l \qquad \frac{B}{A \vee B}\ \mathcal{ND}_0{\vee}I_r \qquad \frac{A \vee B \quad \begin{array}{c}[A]^1\\ \vdots\\ C\end{array} \quad \begin{array}{c}[B]^1\\ \vdots\\ C\end{array}}{C}\ \mathcal{ND}_0{\vee}E^1$$

$$\frac{\begin{array}{c}[A]^1\\ \vdots\\ C\end{array} \quad \begin{array}{c}[A]^1\\ \vdots\\ \neg C\end{array}}{\neg A}\ \mathcal{ND}_0{\neg}I^1 \qquad \frac{\neg\neg A}{A}\ \mathcal{ND}_0{\neg}E$$

$$\frac{\neg A \quad A}{F}\ \mathcal{ND}_0 FI \qquad \frac{F}{A}\ \mathcal{ND}_0 FE$$

- **Again:** $\mathcal{ND}_0{\neg}E$ is used only in classical logic (otherwise constructive/intuitionistic)

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:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A\wedge B\vdash A\wedge B}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{A\wedge B\vdash B}\,\mathcal{ND}^0_\vdash\wedge E_r \qquad \dfrac{\dfrac{}{A\wedge B\vdash A\wedge B}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{A\wedge B\vdash A}\,\mathcal{ND}^0_\vdash\wedge E_l}{A\wedge B\vdash B\wedge A}\,\mathcal{ND}^0_\vdash\wedge I}{\vdash A\wedge B\Rightarrow B\wedge A}\,\mathcal{ND}^0_\vdash\Rightarrow I$$

$$\dfrac{\dfrac{\dfrac{}{A, B\vdash A}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{A\vdash B\Rightarrow A}\,\mathcal{ND}^0_\vdash\Rightarrow I}{\vdash A\Rightarrow B\Rightarrow A}\,\mathcal{ND}^0_\vdash\Rightarrow I$$

- **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 $\mathcal{ND}^0_\vdash$:

$$\frac{}{\Gamma, A \vdash A}\ \mathcal{ND}^0_\vdash\mathrm{Ax} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ \mathcal{ND}^0_\vdash\mathrm{weaken} \qquad \frac{}{\Gamma \vdash A \vee \neg A}\ \mathcal{ND}^0_\vdash\mathrm{TND}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \mathcal{ND}^0_\vdash \wedge I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \mathcal{ND}^0_\vdash \wedge E_l \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \mathcal{ND}^0_\vdash \wedge E_r$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \mathcal{ND}^0_\vdash \vee I_l \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \mathcal{ND}^0_\vdash \vee I_r \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \mathcal{ND}^0_\vdash \vee E$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \mathcal{ND}^0_\vdash \Rightarrow I \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \mathcal{ND}^0_\vdash \Rightarrow E$$

$$\frac{\Gamma, A \vdash F}{\Gamma \vdash \neg A}\ \mathcal{ND}^0_\vdash \neg I \qquad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}\ \mathcal{ND}^0_\vdash \neg E$$

$$\mathcal{ND}^0_\vdash FI\ \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash F} \qquad \mathcal{ND}^0_\vdash FE\ \frac{\Gamma \vdash F}{\Gamma \vdash A}$$

Linearized Notation for (Sequent-Style) ND Proofs

- Linearized notation for sequent-style ND proofs

$$\begin{array}{llll} 1. & \mathcal{H}_1 & \vdash & A_1 & (\mathcal{J}_1) \\ 2. & \mathcal{H}_2 & \vdash & A_2 & (\mathcal{J}_2) \\ 3. & \mathcal{H}_3 & \vdash & A_3 & (\mathcal{J}_3 1, 2) \end{array} \quad \text{corresponds to} \quad \dfrac{\mathcal{H}_1 \vdash A_1 \quad \mathcal{H}_2 \vdash A_2}{\mathcal{H}_3 \vdash A_3}\,\mathcal{R}$$

- **Example 4.10.** We show a linearized version of the $\mathcal{ND}_0$ examples 4.8

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\,\mathcal{ND}^0_\vdash \mathrm{Ax}}{A \wedge B \vdash B}\,\mathcal{ND}^0_\vdash \wedge E_r \quad \dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\,\mathcal{ND}^0_\vdash \mathrm{Ax}}{A \wedge B \vdash A}\,\mathcal{ND}^0_\vdash \wedge E_l}{A \wedge B \vdash B \wedge A}\,\mathcal{ND}^0_\vdash \wedge I}{\vdash A \wedge B \Rightarrow B \wedge A}\,\mathcal{ND}^0_\vdash \Rightarrow I$$

$$\dfrac{\dfrac{\dfrac{}{A, B \vdash A}\,\mathcal{ND}^0_\vdash A}{A \vdash B \Rightarrow A}\,\mathcal{ND}^0_\vdash}{\vdash A \Rightarrow B \Rightarrow A}\,\mathcal{ND}$$

#	*hyp*	$\vdash$	*formula*	*NDjust*
1.	1	$\vdash$	$A \wedge B$	$\mathcal{ND}^0_\vdash \mathrm{Ax}$
2.	1	$\vdash$	B	$\mathcal{ND}^0_\vdash \wedge E_r$ 1
3.	1	$\vdash$	A	$\mathcal{ND}^0_\vdash \wedge E_l$ 1
4.	1	$\vdash$	$B \wedge A$	$\mathcal{ND}^0_\vdash \wedge I$ 2, 3
5.		$\vdash$	$A \wedge B \Rightarrow B \wedge A$	$\mathcal{ND}^0_\vdash \Rightarrow I$ 4

#	*hyp*	$\vdash$	*formula*	*NDjust*
1.	1	$\vdash$	A	$\mathcal{ND}^0_\vdash \mathrm{Ax}$
2.	2	$\vdash$	B	$\mathcal{ND}^0_\vdash \mathrm{Ax}$
3.	1, 2	$\vdash$	A	$\mathcal{ND}^0_\vdash \mathrm{weaken}$
4.	1	$\vdash$	$B \Rightarrow A$	$\mathcal{ND}^0_\vdash \Rightarrow I$ 3
5.		$\vdash$	$A \Rightarrow B \Rightarrow A$	$\mathcal{ND}^0_\vdash \Rightarrow I$ 4

10.5 Predicate Logic Without Quantifiers

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/Relations

- **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/Relations

- **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 5.3.** A first-order signature $\langle\Sigma^f,\Sigma^p\rangle$ consists of
  - $\Sigma^f{:=}\bigcup_{k\in\mathbb{N}}\Sigma_k^f$ of function constants, where members of Σ_k^f denote k-ary functions on individuals,
  - $\Sigma^p{:=}\bigcup_{k\in\mathbb{N}}\Sigma_k^p$ of predicate constants, where members of Σ_k^p denote k-ary relations among individuals,

  where Σ_k^f and Σ_k^p are pairwise disjoint, countable sets of symbols for each $k\in\mathbb{N}$.

Individuals and their Properties/Relations

- **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 5.5.** 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 Σ^p_k denote k-ary relations among individuals,

  where Σ^f_k and Σ^p_k are pairwise disjoint, countable sets of symbols for each $k\in\mathbb{N}$.
- **Definition 5.6.** 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	$\in$	Σ^p_k	
terms	t	$::=$	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

$\mathrm{PL}^{\mathrm{nq}}$ Semantics

- **Definition 5.7.** Domains $\mathcal{D}_0 = \{\mathsf{T}, \mathsf{F}\}$ of truth values and $\mathcal{D}_\iota \neq \emptyset$ of individuals.
- **Definition 5.8.** 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)
  - $\mathcal{I} \colon \Sigma_k^p \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicate constants as relations)
- **Definition 5.9.** 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 5.10.** Model: $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$ varies in $\mathcal{D}_\iota$ and $\mathcal{I}$.
- **Theorem 5.11.** *$\mathrm{PL}^{\mathrm{nq}}$ is isomorphic to PL^0* (*interpret atoms as prop. variables*)

A Model for $\mathrm{PL}^{\mathrm{nq}}$

- **Example 5.12.** 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 5.13 (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, $\mathcal{I}(R(a, b) \wedge P(c)) = \mathsf{F}$.

$\mathrm{PL^{nq}}$ and $\mathrm{PL^0}$ are Isomorphic

- **Observation:** For every choice of Σ of signature, the set $\mathcal{A}_\Sigma$ of atomic $\mathrm{PL^{nq}}$ 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 formulae as $\mathrm{PL^{nq}}$ and $\mathrm{PL^0}$ share connectives.
- **Lemma 5.14.** *For every model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$, 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 $\varphi_\mathcal{M}(X) := \mathcal{I}(\theta_\Sigma^{-1}(X))$
- **Lemma 5.15.** *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 5.16.** *$\mathrm{PL^{nq}}$ is isomorphic to $\mathrm{PL^0}$, i.e. the following diagram commutes:*

$$\begin{tikzcd}
\langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle & & \mathcal{V}_\Sigma \to \{\mathsf{T}, \mathsf{F}\} \arrow[ll, "\psi \mapsto \mathcal{M}^\psi"'] \\
\mathrm{PL^{nq}}(\Sigma) \arrow[u, "\mathcal{I}^\psi()"] \arrow[rr, "\theta_\Sigma"] & & \mathrm{PL^0}(\mathcal{A}_\Sigma) \arrow[u, "\mathcal{I}_{\varphi_\mathcal{M}}()"']
\end{tikzcd}$$

- **Note:** This constellation with a language isomorphism and a corresponding model isomorphism (in converse direction) is typical for a logic isomorphism.

Valuation and Satisfiability

- **Lemma 5.17.** *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 $\mathcal{I}_\psi(\mathrm{A}) = \mathcal{I}^\psi(\mathrm{A})$.*
- *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
     - $\mathcal{I}^\psi(f)\colon {\mathcal{D}_\iota}^k \to {\mathcal{D}^\psi}^k; \langle \mathrm{A}_1, \ldots, \mathrm{A}_k \rangle \mapsto f(\mathrm{A}_1, \ldots, \mathrm{A}_k)$ for $f \in \Sigma_k^f$
     - $\mathcal{I}^\psi(p) := \{\langle \mathrm{A}_1, \ldots, \mathrm{A}_k \rangle \mid \psi({\theta_\psi}^{-1} p(\mathrm{A}_1, \ldots, \mathrm{A}_k)) = \mathsf{T}\}$ for $p \in \Sigma^p$.
  2. We show $\mathcal{I}^\psi(\mathrm{A}) = \mathrm{A}$ for terms A by induction on A
     2.1. If $\mathrm{A} = c$, then $\mathcal{I}^\psi(\mathrm{A}) = \mathcal{I}^\psi(c) = c = \mathrm{A}$
     2.2. If $\mathrm{A} = f(\mathrm{A}_1, \ldots, \mathrm{A}_n)$ then
     $\mathcal{I}^\psi(\mathrm{A}) = \mathcal{I}^\psi(f)(\mathcal{I}(\mathrm{A}_1), \ldots, \mathcal{I}(\mathrm{A}_n)) = \mathcal{I}^\psi(f)(\mathrm{A}_1, \ldots, \mathrm{A}_k) = \mathrm{A}$.
  3. For a $\mathrm{PL}^{\mathrm{nq}}$ formula A we show that $\mathcal{I}^\psi(\mathrm{A}) = \mathcal{I}_\psi(\mathrm{A})$ by induction on A.
     3.1. If $\mathrm{A} = p(\mathrm{A}_1, \ldots, \mathrm{A}_k)$, then $\mathcal{I}^\psi(\mathrm{A}) = \mathcal{I}^\psi(p)(\mathcal{I}(\mathrm{A}_1), \ldots, \mathcal{I}(\mathrm{A}_n)) = \mathsf{T}$, iff $\langle \mathrm{A}_1, \ldots, \mathrm{A}_k \rangle \in \mathcal{I}^\psi(p)$, iff $\psi({\theta_\psi}^{-1}\mathrm{A}) = \mathsf{T}$, so $\mathcal{I}^\psi(\mathrm{A}) = \mathcal{I}_\psi(\mathrm{A})$ as desired.
     3.2. If $\mathrm{A} = \neg \mathrm{B}$, then $\mathcal{I}^\psi(\mathrm{A}) = \mathsf{T}$, iff $\mathcal{I}^\psi(\mathrm{B}) = \mathsf{F}$, iff $\mathcal{I}^\psi(\mathrm{B}) = \mathcal{I}_\psi(\mathrm{B})$, iff $\mathcal{I}^\psi(\mathrm{A}) = \mathcal{I}_\psi(\mathrm{A})$.
     3.3. If $\mathrm{A} = \mathrm{B} \wedge \mathrm{C}$ then we argue similarly
  4. Hence $\mathcal{I}^\psi(\mathrm{A}) = \mathcal{I}_\psi(\mathrm{A})$ for all $\mathrm{PL}^{\mathrm{nq}}$ formulae and we have concluded the proof.

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 formulas are built from atomic propositions, with the connectives *and*, *or*, *not*.

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 $\rightsquigarrow$ 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 encodings are generated by computer programs. This mitigates (but does not remove!) the inconveniences of propositional modeling.

Chapter 11
Machine-Oriented Calculi for Propositional Logic

Automated Deduction as an Agent Inference Procedure

- **Recall:** Our knowledge of the cave entails a definite Wumpus position! (slide 312)
- **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.

Unsatisfiability Theorem

- **Theorem 0.1 (Unsatisfiability Theorem).** *$\mathcal{H} \models A$ iff $\mathcal{H} \cup \{\neg A\}$ is unsatisfiable.*
- *Proof:* We prove both directions separately
  1. "$\Rightarrow$": Say $\mathcal{H} \models A$
     1.1. For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \models A$ and thus $\varphi \not\models \neg A$.
  2. "$\Leftarrow$": Say $\mathcal{H} \cup \{\neg A\}$ is unsatisfiable.
     2.1. For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \not\models \neg A$ and thus $\varphi \models A$.
- **Observation 0.2.** *Entailment can be tested via satisfiability.*

Test Calculi: A Paradigm for Automating Inference

- **Definition 0.3.** Given a logical system $\mathcal{L}$ and a conjecture C, theorem proving consists of finding a calculus for $\mathcal{L}$ and establising that C is valid in the induced formal system: Given a formal system $\langle\mathcal{L},\mathcal{K},\models,\mathcal{C}\rangle$, 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 0.4.** Automated theorem proving (ATP) is the automation of theorem proving: Given a logical system $\mathcal{L}:=\langle\mathcal{L},\mathcal{K},\models\rangle$, the task of automated theorem proving consists of developing calculi for $\mathcal{L}$ and programs – called (automated) theorem provers – that given a set $\mathcal{H}\subseteq\mathcal{L}$ of hypotheses and a conjecture $\mathrm{A}\in\mathcal{L}$ determine whether $\mathcal{H}\models\mathrm{A}$ (usually by searching for $\mathcal{C}$-derivations $\mathcal{H}\vdash_{\mathcal{C}}\mathrm{A}$ in a calculus $\mathcal{C}$).

Test Calculi: A Paradigm for Automating Inference

- **Definition 0.6.** Given a logical system $\mathcal{L}$ and a conjecture C, theorem proving consists of finding a calculus for $\mathcal{L}$ and establising that C is valid in the induced formal system: Given a formal system $\langle\mathcal{L},\mathcal{K},\models,\mathcal{C}\rangle$, 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 0.7.** Automated theorem proving (ATP) is the automation of theorem proving: Given a logical system $\mathcal{L}:=\langle\mathcal{L},\mathcal{K},\models\rangle$, the task of automated theorem proving consists of developing calculi for $\mathcal{L}$ and programs – called (automated) theorem provers – that given a set $\mathcal{H}\subseteq\mathcal{L}$ of hypotheses and a conjecture $\mathrm{A}\in\mathcal{L}$ determine whether $\mathcal{H}\models\mathrm{A}$ (usually by searching for $\mathcal{C}$-derivations $\mathcal{H}\vdash_{\mathcal{C}}\mathrm{A}$ in a calculus $\mathcal{C}$).
- **Idea:** ATP with a calculus $\mathcal{C}$ for $\langle\mathcal{L},\mathcal{K},\models\rangle$ induces a search problem $\Pi:=\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, where the states $\mathcal{S}$ are sets of formulae in $\mathcal{L}$, 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.

Test Calculi: A Paradigm for Automating Inference

- **Definition 0.9.** Given a logical system $\mathcal{L}$ and a conjecture C, theorem proving consists of finding a calculus for $\mathcal{L}$ and establising that C is valid in the induced formal system: Given a formal system $\langle \mathcal{L}, \mathcal{K}, \models, \mathcal{C} \rangle$, 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 0.10.** Automated theorem proving (ATP) is the automation of theorem proving: Given a logical system $\mathcal{L} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$, the task of automated theorem proving consists of developing calculi for $\mathcal{L}$ and programs – called (automated) theorem provers – that given a set $\mathcal{H} \subseteq \mathcal{L}$ of hypotheses and a conjecture $\mathrm{A} \in \mathcal{L}$ determine whether $\mathcal{H} \models \mathrm{A}$ (usually by searching for $\mathcal{C}$-derivations $\mathcal{H} \vdash_{\mathcal{C}} \mathrm{A}$ in a calculus $\mathcal{C}$).
- **Idea:** ATP with a calculus $\mathcal{C}$ for $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ induces a search problem $\Pi := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, where the states $\mathcal{S}$ are sets of formulae in $\mathcal{L}$, 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 (0.1).
- **Definition 0.11.** For a given conjecture A and hypotheses $\mathcal{H}$ a test calculus $\mathcal{T}$ tries to derive $\mathcal{H}, \overline{A} \vdash_{\mathcal{T}} \bot$ instead of $\mathcal{H} \vdash A$, where $\overline{A}$ is unsatisfiable iff A is valid

Test Calculi: A Paradigm for Automating Inference

- **Definition 0.12.** Given a logical system $\mathcal{L}$ and a conjecture C, theorem proving consists of finding a calculus for $\mathcal{L}$ and establising that C is valid in the induced formal system: Given a formal system $\langle \mathcal{L}, \mathcal{K}, \models, \mathcal{C} \rangle$, 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 0.13.** Automated theorem proving (ATP) is the automation of theorem proving: Given a logical system $\mathcal{L} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$, the task of automated theorem proving consists of developing calculi for $\mathcal{L}$ and programs – called (automated) theorem provers – that given a set $\mathcal{H} \subseteq \mathcal{L}$ of hypotheses and a conjecture $\mathsf{A} \in \mathcal{L}$ determine whether $\mathcal{H} \models \mathsf{A}$ (usually by searching for $\mathcal{C}$-derivations $\mathcal{H} \vdash_{\mathcal{C}} \mathsf{A}$ in a calculus $\mathcal{C}$).
- **Idea:** ATP with a calculus $\mathcal{C}$ for $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ induces a search problem $\Pi := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, where the states $\mathcal{S}$ are sets of formulae in $\mathcal{L}$, 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 (0.1).
- **Definition 0.14.** For a given conjecture A and hypotheses $\mathcal{H}$ a test calculus $\mathcal{T}$ tries to derive $\mathcal{H}, \overline{A} \vdash_{\mathcal{T}} \bot$ instead of $\mathcal{H} \vdash A$, where $\overline{A}$ is unsatisfiable iff A is valid and $\bot$ an “obviously” unsatisfiable formula.

11.1 Normal Forms

Recap: Atoms and Literals

- **Definition 1.1.** A formula is called atomic (or an atom) if it does not contain logical constants, else it is called complex.
- **Definition 1.2.** We call a pair A^{α} of a formula and a truth value $\alpha \in \{T, F\}$ a labeled formula. For a set Φ of formulae we use $\Phi^{\alpha} := \{A^{\alpha} | A \in \Phi\}$.
- **Definition 1.3.** A labeled atom A^{α} is called a (positive if $\alpha = T$, else negative) 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 1.4.** For a literal A^{α}, we call the literal A^{β} with $\alpha \neq \beta$ the opposite literal (or partner literal).

Alternative Definition: Literals

- **Note:** Literals are often defined without recurring to labeled formulae:
- **Definition 1.5.** 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.

Normal Forms

- There are two quintessential normal forms for propositional formulae: (there are others as well)
- **Definition 1.6.** 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 $\bigwedge_{i=1}^{n}\bigvee_{j=1}^{m_i} l_{ij}$
- **Definition 1.7.** A formula is in disjunctive normal form (DNF) if it is 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 1.8.** *Every formula has equivalent formulae in CNF and DNF.*

11.2 Analytical Tableaux

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:

Tableau refutation (Validity)	Model generation (Satisfiability)
$\models P \wedge Q \Rightarrow Q \wedge P$	$\models P \wedge (Q \vee \neg R) \wedge \neg Q$
$(P \wedge Q \Rightarrow Q \wedge P)^F$ $(P \wedge Q)^T$ $(Q \wedge P)^F$ P^T Q^T $P^F \mid Q^F$ $\perp \mid \perp$	$(P \wedge (Q \vee \neg R) \wedge \neg Q)^T$ $(P \wedge (Q \vee \neg R))^T$ $\neg Q^T$ Q^F P^T $(Q \vee \neg R)^T$ $Q^T \mid \neg R^T$ $\perp \mid R^F$
No Model	Herbrand Model $\{P^T, Q^F, R^F\}$ $\varphi := \{P \mapsto T, Q \mapsto F, R \mapsto F\}$

- **Idea:** Open branches in saturated tableaux yield models.
- **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 2.2.** The propositional tableau calculus $\mathcal{T}_0$ has two inference rules per connective (one for each possible label)

$$\frac{(A \wedge B)^T}{\begin{array}{c}A^T\\ B^T\end{array}}\mathcal{T}_0\wedge \qquad \frac{(A \wedge B)^F}{A^F \mid B^F}\mathcal{T}_0\vee \qquad \frac{\neg A^T}{A^F}\mathcal{T}_0\neg^T \qquad \frac{\neg A^F}{A^T}\mathcal{T}_0\neg^F \qquad \frac{\begin{array}{c}A^\alpha\\ A^\beta\end{array} \quad \alpha \neq \beta}{\perp}\mathcal{T}_0\perp$$

  Use rules exhaustively as long as they contribute new material ($\rightsquigarrow$ termination)

- **Definition 2.3.** We call any tree ($\mid$ 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 $\perp$, else open. A tableau is closed, iff all of its branches are.

Analytical Tableaux ($\mathcal{T}_0$ continued)

- **Definition 2.5 ($\mathcal{T}_0$-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 $\Phi \vdash_{\mathcal{T}_0} A$.

A Valid Real-World Example

- **Example 2.6.** *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 2.7.** *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 2.8.** * *If Mary loves Bill or John loves Mary, then John loves Mary*
  Try proving the implication (this fails)

$$((\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary})) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$\neg(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary})) \wedge \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary})) \wedge \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\neg\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}$$
$$\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}$$
$$\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\begin{array}{c|c} \mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} & \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\ & \bot \end{array}$$

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 2.9.** Does *Mary loves Bill or John loves Mary* entail that *John loves Mary*?

$$
\begin{array}{c}
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}} \\
\begin{array}{c|c}
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} & \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\
 & \perp
\end{array}
\end{array}
$$

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.

11.3 Practical Enhancements for Tableaux

Derived Rules of Inference

- **Definition 3.1.** 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 3.2.** We have the following derivable inference rules in $\mathcal{T}_0$:

$$\frac{(A \Rightarrow B)^T}{A^F \mid B^T} \qquad \frac{(A \Rightarrow B)^F}{\begin{array}{c} A^T \\ B^F \end{array}} \qquad \frac{\begin{array}{c} A^T \\ (A \Rightarrow B)^T \end{array}}{B^T}$$

$$\frac{(A \vee B)^T}{A^T \mid B^T} \qquad \frac{(A \vee B)^F}{\begin{array}{c} A^F \\ B^F \end{array}} \qquad \frac{A \Leftrightarrow B^T}{\begin{array}{c|c} A^T & A^F \\ B^T & B^F \end{array}} \qquad \frac{A \Leftrightarrow B^F}{\begin{array}{c|c} A^T & A^F \\ B^F & B^T \end{array}}$$

$$\begin{array}{c} A^T \\ (A \Rightarrow B)^T \\ (\neg A \vee B)^T \\ \neg(\neg\neg A \wedge \neg B)^T \\ (\neg\neg A \wedge \neg B)^F \\ \begin{array}{c|c} \neg\neg A^F & \neg B^F \\ \neg A^T & B^T \\ A^F & \\ \bot & \end{array} \end{array}$$

Tableaux with derived Rules (example)

Example 3.3.

$$\begin{array}{c}
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}} \\
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}} \\
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\
\bot
\end{array}$$

11.4 Soundness and Termination of Tableaux

Soundness (Tableau)

- **Idea:** A test calculus is refutation sound, iff its inference rules preserve satisfiability and the goal formulae are unsatisfiable.
- **Definition 4.1.** A labeled formula A^α is valid under φ, iff $\mathcal{I}_\varphi(A) = \alpha$.
- **Definition 4.2.** 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 4.3.** *$\mathcal{T}_0$ rules transform satisfiable tableaux into satisfiable ones.*
- **Theorem 4.4 (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 isfalsifiable $\widehat{=}$ not valid.
  2. Then the initial tableau is satisfiable, (Φ^F satisfiable)
  3. so $\mathcal{T}$ is satisfiable, by 4.3.
  4. Thus there is a satisfiable branch (by definition)
  5. but all branches are closed ($\mathcal{T}$ closed)
- **Theorem 4.5 (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 Corollary A.2.2 (A Completeness Proof for Propositional Tableaux) in the AI lecture notes

Termination for Tableaux

- **Lemma 4.6.** *$\mathcal{T}_0$ terminates, i.e. every $\mathcal{T}_0$ tableau becomes saturated after finitely many rule applications.*

Termination for Tableaux

- **Lemma 4.8.** *$\mathcal{T}_0$ 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.

Termination for Tableaux

- **Lemma 4.10.** *$\mathcal{T}_0$ 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 4.11.** *$\mathcal{T}_0$ induces a decision procedure for validity in PL^0.*

  Proof: We combine the results so far
- 1. By 4.6 it is decidable whether $\vdash_{\mathcal{T}_0} A$
  2. By soundness (4.4) and completeness (4.5), $\vdash_{\mathcal{T}_0} A$ iff A is valid.

11.5 Resolution for Propositional Logic

Another Test Calculus: Resolution

- **Definition 5.1.** 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 5.2 (Resolution Calculus).** The resolution calculus $\mathcal{R}_0$ operates a 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 5.3 (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 $\mathcal{D}\colon S \vdash_{\mathcal{R}_0} \Box$.

Clause Normal Form Transformation (A calculus)

- **Definition 5.4.** 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 5.5 (Transformation into Clause Normal Form).** The CNF transformation calculus CNF_0 consists of the following four inference rules on sets of labeled formulae.

$$\frac{C \vee (A \vee B)^T}{C \vee A^T \vee B^T} \quad \frac{C \vee (A \vee B)^F}{C \vee A^F ; C \vee B^F} \qquad \frac{C \vee \neg A^T}{C \vee A^F} \quad \frac{C \vee \neg A^F}{C \vee A^T}$$

- **Definition 5.6.** We write $CNF_0(A^\alpha)$ for the set of all clauses derivable from A^α via the rules above.

Derived Rules of Inference

- **Definition 5.7.** 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 proofs shorter.
- **Example 5.8.**

$$\dfrac{\dfrac{\dfrac{C \vee (A \Rightarrow B)^{T}}{C \vee (\neg A \vee B)^{T}}}{C \vee \neg A^{T} \vee B^{T}}}{C \vee A^{F} \vee B^{T}} \quad \rightsquigarrow \quad \dfrac{C \vee (A \Rightarrow B)^{T}}{C \vee A^{F} \vee B^{T}}$$

- **Other Derived CNF Rules:**

$$\frac{C \vee (A \Rightarrow B)^{T}}{C \vee A^{F} \vee B^{T}} \quad \frac{C \vee (A \Rightarrow B)^{F}}{C \vee A^{T}\,;\, C \vee B^{F}} \qquad \frac{C \vee (A \wedge B)^{T}}{C \vee A^{T}\,;\, C \vee B^{T}} \quad \frac{C \vee (A \wedge B)^{F}}{C \vee A^{F} \vee B^{F}}$$

Example: Proving Axiom S with Resolution

- **Example 5.9.** Clause Normal Form transformation

$$\frac{\frac{\frac{((P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R)^{\mathsf{F}}}{(P \Rightarrow Q \Rightarrow R)^{\mathsf{T}} ; ((P \Rightarrow Q) \Rightarrow P \Rightarrow R)^{\mathsf{F}}}}{P^{\mathsf{F}} \vee (Q \Rightarrow R)^{\mathsf{T}} ; (P \Rightarrow Q)^{\mathsf{T}} ; (P \Rightarrow R)^{\mathsf{F}}}}{P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} ; P^{\mathsf{T}} ; R^{\mathsf{F}}}$$

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 5.10.** 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

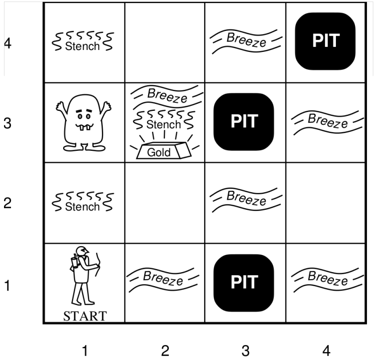
- **Observation:** Let Δ be a clause set, l a literal, and Δ' be Δ where
  - all clauses $l \vee C$ have been removed and
  - and all clauses $\bar{l} \vee C$ have been shortened to C.

  Then Δ is satisfiable, iff Δ' is. We call Δ' the clause set simplification of Δ wrt. l.
- **Corollary 5.11.** *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)

11.6 Killing a Wumpus with Propositional Inference

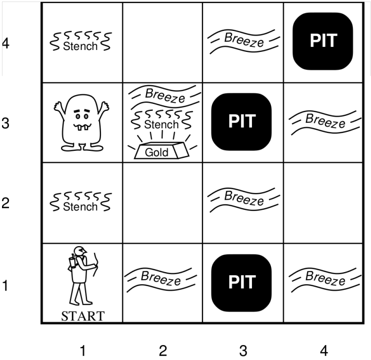
Applying Propositional Inference: Where is the Wumpus?

- **Example 6.1 (Finding the Wumpus).** The situation



Applying Propositional Inference: Where is the Wumpus?

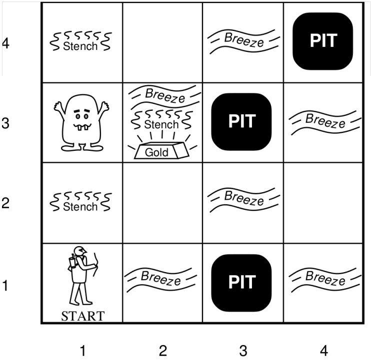
▶ **Example 6.2 (Finding the Wumpus).** The situation and what the agent knows

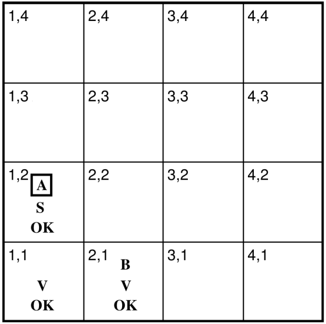


1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

Applying Propositional Inference: Where is the Wumpus?

- **Example 6.3 (Finding the Wumpus).** The situation and what the agent knows

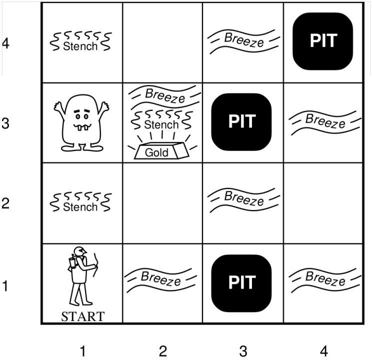




  - What should the agent do next and why?

Applying Propositional Inference: Where is the Wumpus?

- **Example 6.4 (Finding the Wumpus).** The situation and what the agent knows

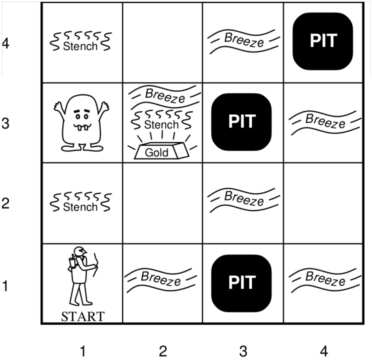


1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 $\boxed{\mathbf{A}}$ **S** **OK**	2,2	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1	4,1

  - 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 6.5 (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)

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)

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**: $W_{1,3}{}^{F}$

Resolution Proof Killing the Wumpus!

- **Example 6.6 (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: ${W_{1,3}}^{F}$ and ${S_{1,2}}^{F} \vee {W_{1,3}}^{T} \vee {W_{2,2}}^{T} \vee {W_{1,1}}^{T}$.
    - Resolvent: ${S_{1,2}}^{F} \vee {W_{2,2}}^{T} \vee {W_{1,1}}^{T}$.
  - *There's a stench in $(1,2)$, so it must be another neighbor.*
    - Parents: ${S_{1,2}}^{T}$ and ${S_{1,2}}^{F} \vee {W_{2,2}}^{T} \vee {W_{1,1}}^{T}$.
    - Resolvent: ${W_{2,2}}^{T} \vee {W_{1,1}}^{T}$.
  - *We've been to $(1,1)$, and there's no Wumpus there, so it can't be $(1,1)$.*
    - Parents: ${W_{1,1}}^{F}$ and ${W_{2,2}}^{T} \vee {W_{1,1}}^{T}$.
    - Resolvent: ${W_{2,2}}^{T}$.
  - *There is no stench in $(2,1)$ so it can't be $(2,2)$ either, in contradiction.*
    - Parents: ${S_{2,1}}^{F}$ and ${S_{2,1}}^{T} \vee {W_{2,2}}^{F}$.
    - Resolvent: ${W_{2,2}}^{F}$.
    - Parents: ${W_{2,2}}^{F}$ and ${W_{2,2}}^{T}$.
    - Resolvent: $\Box$.

As resolution is sound, we have shown that indeed $R_1, R_2, R_3, R_4 \models W_{1,3}$.

Where does the Conjecture ${W_{1,3}}^{\mathsf{F}}$ come from?

- **Question:** Where did the ${W_{1,3}}^{\mathsf{F}}$ come from?
- **Observation 6.7.** *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 function and a test calculus for testing all dangers (pits), opportunities (gold) and the Wumpus.
- **Example 6.8 (Back to the Wumpus).** In 6.1, 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.

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.

Chapter 12
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 NOR and defined $\wedge$, $\vee$, and $\neg$ via $\uparrow$ and NOR respectively.

$\uparrow$	$\top$	$\bot$
$\top$	F	T
$\bot$	T	T

NOR	$\top$	$\bot$
$\top$	F	F
$\bot$	F	T

$\neg a$	$a \uparrow a$	a NOR a
ab	$a \uparrow b \uparrow a \uparrow b$	a NOR ab NOR b
ab	$a \uparrow a \uparrow b \uparrow b$	a NOR b NOR a NOR b

- **Observation:** The set $\mathit{wff}_0(\mathcal{V}_0)$ 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 $\mathcal{M} \models^{\varphi} \mathrm{A}$, and
  - entailment i.e whether $\mathrm{A} \models \mathrm{B}$.
- **Observation:** In particular, the inductive/compositional nature of $\mathit{wff}_0(\mathcal{V}_0)$ and $\mathcal{I}_\varphi \colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_0$ are secondary.
- **Idea:** Concentrate on language, models $(\mathcal{M}, \varphi)$, and satisfiability.

Logical Systems

- **Definition 0.1.** A logical system (or simply a logic) is a triple $\mathcal{L}:=\langle\mathcal{L},\mathcal{K},\models\rangle$, where $\mathcal{L}$ is a formal language, $\mathcal{K}$ is a set and $\models\subseteq\mathcal{K}\times\mathcal{L}$. Members of $\mathcal{L}$ are called formulae of $\mathcal{L}$, members of $\mathcal{K}$ models for $\mathcal{L}$, and $\models$ the satisfaction relation.
- **Example 0.2 (Propositional Logic).** $\langle wff(\Sigma_{PL^0},\mathcal{V}_{PL^0}),\mathcal{K},\models\rangle$ is a logical system, if we define $\mathcal{K}:=\mathcal{V}_0\rightharpoonup\mathcal{D}_0$ (the set of variable assignments) and $\varphi\models\mathrm{A}$ iff $\mathcal{I}_\varphi(\mathrm{A})=\mathsf{T}$.
- **Definition 0.3.** Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system, $\mathcal{M}\in\mathcal{K}$ be a model and $\mathrm{A}\in\mathcal{L}$ a formula, then we say that A is
  - satisfied by $\mathcal{M}$, iff $\mathcal{M}\models\mathrm{A}$.
  - falsified by $\mathcal{M}$, iff $\mathcal{M}\not\models\mathrm{A}$.
  - satisfiable in $\mathcal{K}$, iff $\mathcal{M}\models\mathrm{A}$ for some $\mathcal{M}\in\mathcal{K}$.
  - valid in $\mathcal{K}$ (write $\models\mathrm{A}$), iff $\mathcal{M}\models\mathrm{A}$ for all $\mathcal{M}\in\mathcal{K}$.
  - falsifiable in $\mathcal{K}$, iff $\mathcal{M}\not\models\mathrm{A}$ for some $\mathcal{M}\in\mathcal{K}$.
  - unsatisfiable in $\mathcal{K}$, iff $\mathcal{M}\not\models\mathrm{A}$ for all $\mathcal{M}\in\mathcal{K}$.

Derivation Relations and Inference Rules

- **Definition 0.4.** Let $\mathcal{L}:=\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system, then we call a relation $\vdash\subseteq\mathcal{P}(\mathcal{L})\times\mathcal{L}$ a derivation relation for $\mathcal{L}$, if
  - $\mathcal{H}\vdash \mathrm{A}$, if $\mathrm{A}\in\mathcal{H}$ ($\vdash$ is proof reflexive),
  - $\mathcal{H}\vdash \mathrm{A}$ and $\mathcal{H}'\cup\{\mathrm{A}\}\vdash \mathrm{B}$ imply $\mathcal{H}\cup\mathcal{H}'\vdash \mathrm{B}$ ($\vdash$ is proof transitive),
  - $\mathcal{H}\vdash \mathrm{A}$ and $\mathcal{H}\subseteq\mathcal{H}'$ imply $\mathcal{H}'\vdash \mathrm{A}$ ($\vdash$ is monotonic or admits weakening).
- **Definition 0.5.** We call $\langle\mathcal{L},\mathcal{K},\models,\mathcal{C}\rangle$ a formal system, iff $\mathcal{L}:=\langle\mathcal{L},\mathcal{K},\models\rangle$ is a logical system, and $\mathcal{C}$ a calculus for $\mathcal{L}$.

Derivation Relations and Inference Rules

- **Definition 0.9.** Let $\mathcal{L}{:=}\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system, 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 0.10.** We call $\langle\mathcal{L},\mathcal{K},\models,\mathcal{C}\rangle$ a formal system, iff $\mathcal{L}{:=}\langle\mathcal{L},\mathcal{K},\models\rangle$ is a logical system, and $\mathcal{C}$ a calculus for $\mathcal{L}$.
- **Definition 0.11.** Let $\mathcal{L}$ be the formal language of a logical system, then an inference rule over $\mathcal{L}$ is a decidable $n+1$ ary relation on $\mathcal{L}$. Inference rules are traditionally written as

$$\frac{A_1 \;\; \ldots \;\; A_n}{C}\,\mathcal{N}$$

  where $A_1,\ldots,A_n$ and C are formula schemata for $\mathcal{L}$ and $\mathcal{N}$ is a name.
  The A_i are called assumptions of $\mathcal{N}$, and C is called its conclusion.
- **Definition 0.12.** An inference rule without assumptions is called an axiom.
- **Definition 0.13.** Let $\mathcal{L}{:=}\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system, then we call a set $\mathcal{C}$ of inference rules over $\mathcal{L}$ a calculus (or inference system) for $\mathcal{L}$.

Derivations

- **Definition 0.14.** Let $\mathcal{L}:=\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system and $\mathcal{C}$ a calculus for $\mathcal{L}$, then a $\mathcal{C}$-derivation of a formula $\mathrm{C}\in\mathcal{L}$ 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 $\mathcal{L}$-formulae, such that
  - $\mathrm{A}_m=\mathrm{C}$, (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 $\frac{\mathrm{A}_{l_1}\quad\ldots\quad\mathrm{A}_{l_k}}{\mathrm{A}_i}$ 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_k.

- **Example 0.15.**
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.

$$\dfrac{\dfrac{}{P\Rightarrow Q\Rightarrow P}K \qquad P}{Q\Rightarrow P}MP$$

Formal Systems

- Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system and $\mathcal{C}$ a calculus, then $\vdash_{\mathcal{C}}$ is a derivation relation and thus $\langle\mathcal{L},\mathcal{K},\models,\vdash_{\mathcal{C}}\rangle$ a derivation system.
- Therefore we will sometimes also call $\langle\mathcal{L},\mathcal{K},\models,\mathcal{C}\rangle$ a formal system, iff $\mathcal{L}{:=}\langle\mathcal{L},\mathcal{K},\models\rangle$ is a logical system, and $\mathcal{C}$ a calculus for $\mathcal{L}$.
- **Definition 0.16.** 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 0.17. The act of finding a proof for a formula A is called proving A.
- **Definition 0.18.** 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 0.19.** An inference rule $\dfrac{\mathrm{A}_1 \quad \ldots \quad \mathrm{A}_n}{\mathrm{C}}$ 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 0.20.** *Derivable inference rules are admissible, but not the other way around.*

Chapter 13
Propositional Reasoning: SAT Solvers

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.

SAT: The Propositional Satisfiability Problem

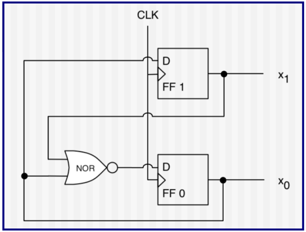
- **Definition 1.1.** The SAT problem (SAT): Given a propositional formula A, decide whether or not A is satisfiable. We denote the class of all SAT problems with SAT
- The SAT problem was the first problem proved to be NP-complete!
- A is commonly assumed to be in CNF. This is without loss of generality, 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 $\mathrm{KB} \models \mathrm{A}$, decide satisfiability of $\theta := \mathrm{KB} \cup \{\neg \mathrm{A}\}$: θ is unsatisfiable iff $\mathrm{KB} \models \mathrm{A}$.
- **Consequence:** Deduction can be performed using SAT solvers.

SAT vs. CSP

- **Recall:** Constraint network $\langle V, D, C\rangle$ has variables $v \in V$ with finite domains $D_v \in D$, and binary constraints $C_{uv} \in C$ which are relations over u, 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 arity.*
- **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.

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).

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.

13.2 The Davis-Putnam (Logemann-Loveland) Procedure

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(Δ,I) **returns** a partial assignment I, or ''unsatisfiable''
  /∗ Unit Propagation (UP) Rule: ∗/
  Δ' := a copy of Δ; I' := I
  while Δ' contains a unit clause $C = P^\alpha$ **do**
    extend I' with $[\alpha/P]$, clause−set simplify Δ'
   /∗ Termination Test: ∗/
   if $\Box \in \Delta'$ **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; Δ'' := a copy of Δ'; simplify Δ''
  if I''' := DPLL(Δ'',I'') $\neq$ ''unsatisfiable'' **then return** I'''
  I'' := I' extended with the other truth value **for** P; Δ'' := Δ'; simplify Δ''
  return DPLL(Δ'',I'')

- In practice, of course one uses flags etc. instead of "copy".

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 "$R \mapsto \mathsf{T}, P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}$"

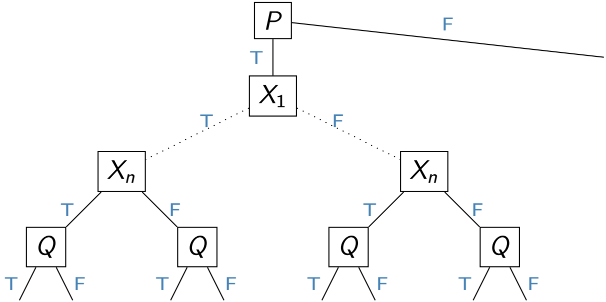
DPLL: Example (Vanilla2)

- **Observation:** Sometimes UP is all we need.
- **Example 2.3.** Let $\Delta := (Q^F \vee P^F \,;\, P^T \vee Q^F \vee R^F \vee S^F \,;\, Q^T \vee S^F \,;\, R^T \vee S^F \,;\, S^T)$

  1. UP Rule: $S \mapsto T$
     $Q^F \vee P^F \,;\, P^T \vee Q^F \vee R^F \,;\, Q^T \,;\, R^T$
  2. UP Rule: $Q \mapsto T$
     $P^F \,;\, P^T \vee R^F \,;\, R^T$
  3. UP Rule: $R \mapsto T$
     $P^F \,;\, P^T$
  4. UP Rule: $P \mapsto T$
     $\Box$

DPLL: Example (Redundance1)

- **Example 2.4.** We introduce some nasty redundance to make DPLL slow.
  $\Delta := (P^F \vee Q^F \vee R^T ; P^F \vee Q^F \vee R^F ; P^F \vee Q^T \vee R^T ; P^F \vee Q^T \vee R^F)$
  DPLL on $\Delta ; \Theta$ with $\Theta := ({X_1}^T \vee \ldots \vee {X_n}^T ; {X_1}^F \vee \ldots \vee {X_n}^F)$



$R^T ; \Box \; R^T ; \Box \; R^T ; \Box \; R^T ; \Box \; R^T ; \Box \; R^T ; \Box \; R^T ; \Box \; R^T ; \Box$

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.)

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?

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.

13.3 DPLL $\widehat{=}$ (A Restricted Form of) Resolution

UP $\widehat{=}$ Unit Resolution

- **Observation:** The unit propagation (UP) rule corresponds to a calculus:

while Δ' contains a unit clause $\{l\}$ **do**
  extend $\mathcal{I}'$ with the respective truth value **for** the proposition underlying l
  simplify Δ' /* 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.

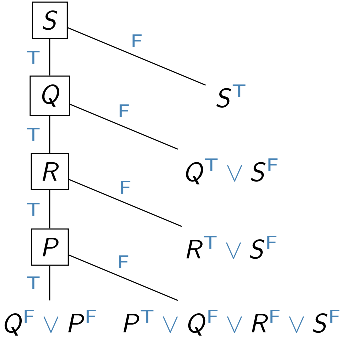
DPLL vs. Resolution

- **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.

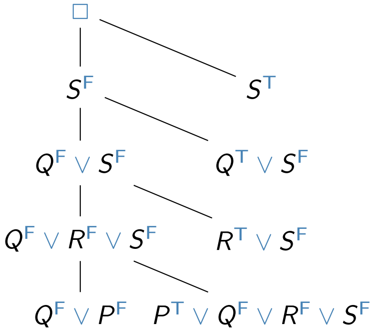
DPLL vs. Resolution: Example (Vanilla2)

- **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 $\Delta{:=}(Q^{\mathsf{F}} \vee P^{\mathsf{F}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{F}} ; Q^{\mathsf{T}} \vee S^{\mathsf{F}} ; R^{\mathsf{T}} \vee S^{\mathsf{F}} ; S^{\mathsf{T}})$

  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.

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)

13.4 Conclusion

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.

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.

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 (Δ, MaxFlips MaxTries
  for i :=1 to MaxTries
    I := a randomly−generated truth assignment
    for j :=1 to MaxFlips
    if I satisfies Δ 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 398) 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)?

Chapter 14
First-Order Predicate Logic

14.1 Motivation: A more Expressive Language

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?

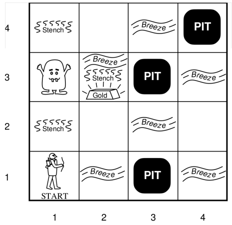
- **You say:** "All blocks are red"; "All blocks are on the table"; "A is a block".
- **And now:** Say it in propositional logic!
- **Answer:** "isRedA","isRedB", ..., "onTableA", "onTableB", ..., "isBlockA", ...
- **Wait a sec!:** Why don't we just say, e.g., "AllBlocksAreRed" and "isBlockA"?
- **Problem:** 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.

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.** "$\forall x.\mathrm{block}(x) \Rightarrow \mathrm{red}(x)$"; "$\mathrm{block}(\mathsf{A})$"

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:** "$\forall x.\mathrm{Wumpus}(x) \Rightarrow (\forall y.\mathrm{adj}(x, y) \Rightarrow \mathrm{stench}(y))$"

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: "$\forall x.\mathrm{even}(x) \Rightarrow \mathrm{even}(\mathrm{succ}(\mathrm{succ}(x)))$".

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 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`

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."*

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.)

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.

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.

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)

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.*

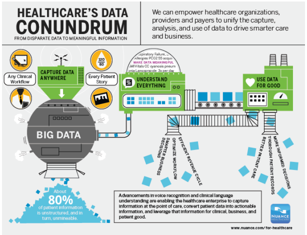
(A Few) Semantic Technology Applications

Web Queries

Jeopardy (IBM Watson)

Context-Aware Apps

Healthcare



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.

14.2 First-Order Logic

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,...*)

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 computer science. 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: $\Sigma_k^f = \{f, g, h, \ldots\}$ (k-ary functions on individuals)
  - predicate constants: $\Sigma_k^p = \{p, q, r, \ldots\}$ (k-ary relations among individuals.)
  - (Skolem constants: $\Sigma_k^{sk} = \{f_k^1, f_k^2, \ldots\}$) (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 ∞)

PL^1 Syntax (Formulae)

- **Definition 2.4.** Terms: $\mathbf{A} \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ (denote individuals)
  - $\mathcal{V}_\iota \subseteq wff_\iota(\Sigma_1, \mathcal{V}_\iota)$,
  - if $f \in \Sigma_k^f$ and $\mathbf{A}^i \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $f(\mathbf{A}^1, \ldots, \mathbf{A}^k) \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$.
- **Definition 2.5.** Propositions: $\mathbf{A} \in wff_o(\Sigma_1, \mathcal{V}_\iota)$: (denote truth values)
  - if $p \in \Sigma_k^p$ and $\mathbf{A}^i \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $p(\mathbf{A}^1, \ldots, \mathbf{A}^k) \in wff_o(\Sigma_1, \mathcal{V}_\iota)$,
  - if $\mathbf{A}, \mathbf{B} \in wff_o(\Sigma_1, \mathcal{V}_\iota)$ and $X \in \mathcal{V}_\iota$, then $T, \mathbf{A} \wedge \mathbf{B}, \neg \mathbf{A}, \forall X.\mathbf{A} \in wff_o(\Sigma_1, \mathcal{V}_\iota)$.
    $\forall$ is a binding operator called the universal quantifier.
- **Definition 2.6.** We define the connectives $F, \vee, \Rightarrow, \Leftrightarrow$ via the abbreviations $\mathbf{A} \vee \mathbf{B} := \neg(\neg \mathbf{A} \wedge \neg \mathbf{B})$, $\mathbf{A} \Rightarrow \mathbf{B} := \neg \mathbf{A} \vee \mathbf{B}$, $\mathbf{A} \Leftrightarrow \mathbf{B} := (\mathbf{A} \Rightarrow \mathbf{B}) \wedge (\mathbf{B} \Rightarrow \mathbf{A})$, and $F := \neg T$. We will use them like the primary connectives $\wedge$ and $\neg$
- **Definition 2.7.** We use $\exists X.\mathbf{A}$ as an abbreviation for $\neg(\forall X.\neg \mathbf{A})$. $\exists$ is a binding operator called the existential quantifier.
- **Definition 2.8.** Call formulae without connectives or quantifiers atomic else complex.

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 $\forall X.B$ of A.
  For a formula A, we will use $\mathrm{BVar}(A)$ (and $\mathrm{free}(A)$) 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 $\mathrm{free}(A)$ of free variables of a formula A:

$$
\begin{aligned}
&\mathrm{free}(X){:=}\{X\}\\
&\mathrm{free}(f(A_1,\ldots,A_n)){:=}\bigcup_{1\leq i\leq n}\mathrm{free}(A_i)\\
&\mathrm{free}(p(A_1,\ldots,A_n)){:=}\bigcup_{1\leq i\leq n}\mathrm{free}(A_i)\\
&\mathrm{free}(\neg A){:=}\mathrm{free}(A)\\
&\mathrm{free}(A\wedge B){:=}\mathrm{free}(A)\cup\mathrm{free}(B)\\
&\mathrm{free}(\forall X.A){:=}\mathrm{free}(A)\backslash\{X\}
\end{aligned}
$$

- **Definition 2.11.** We call a formula A closed or ground, iff $\mathrm{free}(A)=\emptyset$. 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 $\forall X.B$ of a formula A can be replaced by $A'{:=}(\forall Y.B')$, where B' arises from B by replacing all $X\in\mathrm{free}(B)$ 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.*

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)
  - $\mathcal{I}\colon \Sigma_k^f \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (interpret function symbols as arbitrary functions)
  - $\mathcal{I}\colon \Sigma_k^p \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (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)
  - $\mathcal{I}_\varphi\colon \mathit{wff}_\iota(\Sigma_1,\mathcal{V}_\iota)\to\mathcal{D}_\iota$ assigns values to terms.
    - $\mathcal{I}_\varphi(X):=\varphi(X)$ and
    - $\mathcal{I}_\varphi(f(\mathrm{A}_1,\ldots,\mathrm{A}_k)):=\mathcal{I}(f)(\mathcal{I}_\varphi(\mathrm{A}_1),\ldots,\mathcal{I}_\varphi(\mathrm{A}_k))$
  - $\mathcal{I}_\varphi\colon \mathit{wff}_o(\Sigma_1,\mathcal{V}_\iota)\to\mathcal{D}_0$ 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.

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 $\Sigma_2^p{:=}\{o\}$
  - 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 $\psi{:=}\varphi,[a/X]$ for $a{\in}\mathcal{D}_\iota$ we have

$$
\begin{array}{lll}
\mathcal{I}_\varphi(\forall X.o(f(X), X)) = \mathsf{T} & \text{iff} & \mathcal{I}_\psi(o(f(X), X)) = \mathsf{T} \text{ for all } a{\in}\mathcal{D}_\iota \\
& \text{iff} & (\mathcal{I}_\psi(f(X)),\mathcal{I}_\psi(X)){\in}\mathcal{I}(o) \text{ for all } a{\in}\{J, M\} \\
& \text{iff} & (\mathcal{I}(f)(\mathcal{I}_\psi(X)),\psi(X)){\in}\{(M,J)\} \text{ for all } a{\in}\{J, M\} \\
& \text{iff} & (\mathcal{I}(f)(\psi(X)),a) = (M,J) \text{ for all } a{\in}\{J, M\} \\
& \text{iff} & \mathcal{I}(f)(a) = M \text{ and } a = J \text{ for all } a{\in}\{J, M\}
\end{array}
$$

  But $a \neq J$ for $a = M$, so $\mathcal{I}_\varphi(\forall X.o(f(X), X)) = \mathsf{F}$ in the model $\langle\mathcal{D}_\iota,\mathcal{I}\rangle$.

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 $[\mathrm{B}/X](\mathrm{A})$.
- **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)
- **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 $\mathrm{supp}(\sigma){:=}\{X|(X,\mathrm{A})\in\sigma, X \neq \mathrm{A}\}$ of σ is finite. We denote the empty substitution with ϵ.
- **Definition 2.21.** We can discharge a variable X from a substitution σ by setting $\sigma_{-X}{:=}\sigma,[X/X]$.
- **Definition 2.22 (Substitution Application).** We define substitution application by
  - $\sigma(c) = c$ for $c\in\Sigma$
  - $\sigma(X) = \mathrm{A}$, iff $X\in\mathcal{V}$ and $(X,\mathrm{A})\in\sigma$.
  - $\sigma(f(\mathrm{A}_1,\ldots,\mathrm{A}_n)) = f(\sigma(\mathrm{A}_1),\ldots,\sigma(\mathrm{A}_n))$,
  - $\sigma(\forall X.\mathrm{A}) = \forall X.\sigma_{-X}(\mathrm{A})$. ($\exists$ analogous)
- **Example 2.23.** $[a/x],[f(b)/y],[a/z]$ instantiates $g(x,y,h(z))$ to $g(a,f(b),h(a))$.

Substitution Extension

- **Definition 2.24 (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 2.25.** We can discharge a variable X from a substitution σ by setting $\sigma_{-X}:=\sigma,[X/X]$.

Substitutions on Propositions

- **Problem:** We want to extend substitutions to propositions, in particular to quantified formulae: What is $\sigma(\forall X.\mathrm{A})$?
- **Idea:** σ should not instantiate bound variables. ($[\mathrm{A}/X](\forall X.\mathrm{B}) = \forall \mathrm{A}.\mathrm{B}'$ ill-formed)
- **Definition 2.26.** $\sigma(\forall X.\mathrm{A}) := (\forall X.\sigma_{-X}(\mathrm{A}))$.
- **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 2.27 (Capture-Avoiding Substitution Application).** Let σ be a substitution, A a formula, and A′ an alphabetic variant of A, such that $\mathrm{intro}(\sigma) \cap \mathrm{BVar}(\mathrm{A}) = \emptyset$. Then we define capture-avoiding substitution application via $\sigma(\mathrm{A}) := \sigma(\mathrm{A}')$.

Substitution Value Lemma for Terms

- **Lemma 2.28.** *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
     $\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A})) = \mathcal{I}_\varphi([\mathrm{B}/X](X)) = \mathcal{I}_\varphi(\mathrm{B}) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(\mathrm{A})$.
     1.2. $\mathrm{A} = Y \neq X$
     1.2.1. then $\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A})) = \mathcal{I}_\varphi([\mathrm{B}/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(\mathrm{A})$.
     1.3. A is a constant
     1.3.1. Analogous to the preceding case ($Y \neq X$).
     1.4. This completes the base case (depth $= 0$).
  2. depth> 0
     2.1. then $\mathrm{A} = f(\mathrm{A}_1, \ldots, \mathrm{A}_n)$ and we have

$$\begin{aligned}\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A})) &= \mathcal{I}(f)(\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A}_1)), \ldots, \mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A}_n))) \\ &= \mathcal{I}(f)(\mathcal{I}_\psi(\mathrm{A}_1), \ldots, \mathcal{I}_\psi(\mathrm{A}_n)) \\ &= \mathcal{I}_\psi(\mathrm{A}).\end{aligned}$$

by induction hypothesis

Substitution Value Lemma for Propositions

- **Lemma 2.29.** $\mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A})) = \mathcal{I}_\psi(\mathrm{A})$, *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.
  2. $n>0$ and $\mathrm{A} = \neg\mathrm{B}$ or $\mathrm{A} = \mathrm{C} \circ \mathrm{D}$
     2.1. Here we argue like in the induction step of the term lemma as well.
  3. $n>0$ and $\mathrm{A} = \forall Y.\mathrm{C}$ where (WLOG) $X \neq Y$ *(otherwise rename)*
     3.1. 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 $a \in \mathcal{D}_\iota$.
     3.2. But $\mathcal{I}_{(\psi,[a/Y])}(\mathrm{C}) = \mathcal{I}_{(\varphi,[a/Y])}([\mathrm{B}/X](\mathrm{C})) = \mathsf{T}$, by induction hypothesis.
     3.3. So $\mathcal{I}_\psi(\mathrm{A}) = \mathcal{I}_\varphi(\forall Y.[\mathrm{B}/X](\mathrm{C})) = \mathcal{I}_\varphi([\mathrm{B}/X](\forall Y.\mathrm{C})) = \mathcal{I}_\varphi([\mathrm{B}/X](\mathrm{A}))$

14.3 First-Order Natural Deduction

First-Order Natural Deduction ($\mathcal{ND}^1$; Gentzen [Gen34])

- 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:

$$\frac{\mathbf{A}}{\forall X.\mathbf{A}}\ \mathcal{ND}^1\forall I^* \qquad \frac{\forall X.\mathbf{A}}{[\mathbf{B}/X](\mathbf{A})}\ \mathcal{ND}^1\forall E$$

$$\frac{[\mathbf{B}/X](\mathbf{A})}{\exists X.\mathbf{A}}\ \mathcal{ND}^1\exists I \qquad \frac{\exists X.\mathbf{A} \qquad \begin{array}{c}[[c/X](\mathbf{A})]^1\\ \vdots\\ \mathbf{C}\end{array} \qquad c\in\Sigma_0^{sk}\ \text{new}}{\mathbf{C}}\ \mathcal{ND}^1\exists E^1$$

* means that A does not depend on any hypothesis in which X is free.

A Complex $\mathcal{ND}^1$ Example

- **Example 3.2.** We prove $\neg(\forall X.P(X)) \vdash_{\mathcal{ND}^1} \exists X.\neg P(X)$.

$$
\dfrac{\dfrac{\neg(\forall X.P(X)) \qquad \dfrac{\dfrac{\dfrac{\dfrac{[\neg(\exists X.\neg P(X))]^1 \qquad \dfrac{[\neg P(X)]^2}{\exists X.\neg P(X)}\,\mathcal{ND}^1\exists I}{F}\,\mathcal{ND}_0 FI}{\neg\neg P(X)}\,\mathcal{ND}_0\neg I^2}{P(X)}\,\mathcal{ND}_0\neg E}{\forall X.P(X)}\,\mathcal{ND}^1\forall I}{F}\,\mathcal{ND}_0 FI}{\dfrac{\neg\neg(\exists X.\neg P(X))}{\exists X.\neg P(X)}\,\mathcal{ND}_0\neg E}\,\mathcal{ND}_0\neg I^1
$$

First-Order Natural Deduction in Sequent Formulation

- Rules for connectives from $\mathcal{ND}^0_\vdash$
- **Definition 3.3 (New Quantifier Rules).** The inference rules of the first-order sequent calculus $\mathcal{ND}^1_\vdash$ consist of those from $\mathcal{ND}^0_\vdash$ plus the following quantifier rules:

$$\frac{\Gamma\vdash A \quad X\notin\mathrm{free}(\Gamma)}{\Gamma\vdash\forall X.A}\ \mathcal{ND}^1_\vdash\forall I \qquad \frac{\Gamma\vdash\forall X.A}{\Gamma\vdash[B/X](A)}\ \mathcal{ND}^1_\vdash\forall E$$

$$\frac{\Gamma\vdash[B/X](A)}{\Gamma\vdash\exists X.A}\ \mathcal{ND}^1_\vdash\exists I \qquad \frac{\Gamma\vdash\exists X.A \quad \Gamma,[c/X](A)\vdash C \quad c\in\Sigma_0^{sk}\ \text{new}}{\Gamma\vdash C}\ \mathcal{ND}^1_\vdash\exists E$$

Natural Deduction with Equality

- **Definition 3.4 (First-Order Logic with Equality).** We extend PL^1 with a new logical constant for equality $= \in \Sigma_2^p$ and fix its interpretation to $\mathcal{I}(=) := \{(x,x) | x \in \mathcal{D}_\iota\}$. We call the extended logic first-order logic with equality ($\mathrm{PL}^1_=$)
- We now extend natural deduction as well.
- **Definition 3.5.** 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:

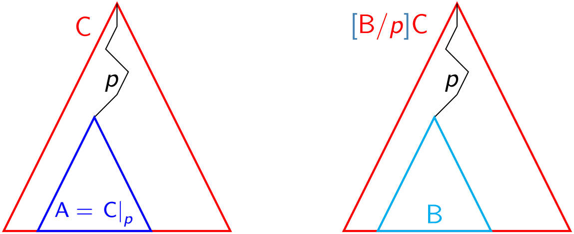
$$\frac{}{\mathrm{A} = \mathrm{A}} =I \qquad \frac{\mathrm{A} = \mathrm{B} \quad \mathrm{C}\,[\mathrm{A}]_p}{[\mathrm{B}/p]\mathrm{C}} =E$$

  where $\mathrm{C}\,[\mathrm{A}]_p$ if the formula C has a subterm A at position p and $[\mathrm{B}/p]\mathrm{C}$ 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 3.6.** $\Leftrightarrow I$ is derivable and $\Leftrightarrow E$ is admissible in $\mathcal{ND}^1$:

$$\frac{}{\mathrm{A} \Leftrightarrow \mathrm{A}} \Leftrightarrow I \qquad \frac{\mathrm{A} \Leftrightarrow \mathrm{B} \quad \mathrm{C}\,[\mathrm{A}]_p}{[\mathrm{B}/p]\mathrm{C}} \Leftrightarrow E$$

Positions in Formulae

- **Idea:** Formulae are (naturally) trees, so we can use tree positions to talk about subformulae
- **Definition 3.8.** A position p is a tuple of natural numbers that in each node of a expression (tree) specifies into which child to descend. For a expression A we denote the subexpression at p with $\mathrm{A}|_p$.
  We will sometimes write a 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 3.9.** 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 3.10 (Schematically).**



$\mathcal{ND}^1_=$ Example: $\sqrt{2}$ is Irrational

- We can do real mathematics with $\mathcal{ND}^1_=$:
- **Theorem 3.11.** $\sqrt{2}$ *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

$\mathcal{ND}^1_=$ 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		$\mathrm{irr}(\sqrt{2}) \Leftrightarrow (\neg(\exists p, q. \sqrt{2} = p/q))$	$\mathcal{ND}^1_\vdash \forall E(4)$
6	6	$\neg \mathrm{irr}(\sqrt{2})$	$\mathcal{ND}^0_\vdash \mathrm{Ax}$
7	6	$\neg\neg(\exists p, q. \sqrt{2} = p/q)$	$\Leftrightarrow E(6,5)$
8	6	$\exists p, q. \sqrt{2} = p/q$	$\mathcal{ND}^0_\vdash \neg E(7)$
9	6,9	$\sqrt{2} = p/q$	$\mathcal{ND}^0_\vdash \mathrm{Ax}$
10	6,9	$2q^2 = p^2$	arith(9)
11	6,9	$\#(p^2) = 2\#(p)$	$\mathcal{ND}^1_\vdash \forall E^2(2)$
12	6,9	$\mathrm{prime}(2) \Rightarrow \#(2q^2) = (\#(q^2)+1)$	$\mathcal{ND}^1_\vdash \forall E^2(1)$

$\mathcal{ND}^1_=$ Example: $\sqrt{2}$ is Irrational (the Proof continued)

13		$\mathrm{prime}(2)$	lemma
14	6,9	$\#(2q^2) = \#(q^2) + 1$	$\mathcal{ND}_0 \Rightarrow E(13, 12)$
15	6,9	$\#(q^2) = 2\#(q)$	$\mathcal{ND}^1 \forall E^2(2)$
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	$\neg(2\#(q) + 1) = (2\#(p))$	$\mathcal{ND}^1 \forall E^2(1)$
22	6,9	F	$\mathcal{ND}_0 FI(20, 21)$
23	6	F	$\mathcal{ND}^1 \exists E^6(22)$
24		$\neg\neg\mathrm{irr}(\sqrt{2})$	$\mathcal{ND}_0 \neg I^6(23)$
25		$\mathrm{irr}(\sqrt{2})$	$\mathcal{ND}_0 \neg E^2(23)$

14.4 Conclusion

Summary (Predicate Logic)

- *Predicate 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*.
- *First-order predicate logic (PL1)* allows *universal* and *existential quantification* over objects.
- A PL1 *interpretation* consists of a *universe* U and a function I mapping *constant symbols/predicate symbols/function symbols* to elements/relations/functions on U.

Chapter 15
Automated Theorem Proving in First-Order Logic

15.1 First-Order Inference with Tableaux

15.1.1 First-Order Tableau Calculi

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:

Tableau refutation (Validity)	Model generation (Satisfiability)
$\models P \wedge Q \Rightarrow Q \wedge P$	$\models P \wedge (Q \vee \neg R) \wedge \neg Q$
$(P \wedge Q \Rightarrow Q \wedge P)^F$ $(P \wedge Q)^T$ $(Q \wedge P)^F$ P^T Q^T $P^F \mid Q^F$ $\bot \mid \bot$	$(P \wedge (Q \vee \neg R) \wedge \neg Q)^T$ $(P \wedge (Q \vee \neg R))^T$ $\neg Q^T$ Q^F P^T $(Q \vee \neg R)^T$ $Q^T \mid \neg R^T$ $\bot \mid R^F$
No Model	Herbrand Model $\{P^T, Q^F, R^F\}$ $\varphi := \{P \mapsto T, Q \mapsto F, R \mapsto F\}$

- **Idea:** Open branches in saturated tableaux yield models.
- **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)

$$\frac{(A \wedge B)^{T}}{\begin{array}{c}A^{T}\\ B^{T}\end{array}}\,\mathcal{T}_0\wedge \qquad \frac{(A \wedge B)^{F}}{A^{F} \mid B^{F}}\,\mathcal{T}_0\vee \qquad \frac{\neg A^{T}}{A^{F}}\,\mathcal{T}_0\neg^{T} \qquad \frac{\neg A^{F}}{A^{T}}\,\mathcal{T}_0\neg^{F} \qquad \frac{\begin{array}{c}A^{\alpha}\\ A^{\beta}\end{array} \quad \alpha \neq \beta}{\perp}\,\mathcal{T}_0\perp$$

  Use rules exhaustively as long as they contribute new material ($\leadsto$ termination)
- **Definition 1.3.** We call any tree ($\mid$ 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.

Analytical Tableaux ($\mathcal{T}_0$ continued)

- **Definition 1.5 ($\mathcal{T}_0$-Theorem/Derivability).** A is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0} \mathrm{A}$), iff there is a closed tableau with A^{F} at the root.
  $\Phi \subseteq wff_0(\mathcal{V}_0)$ derives A in $\mathcal{T}_0$ ($\Phi \vdash_{\mathcal{T}_0} \mathrm{A}$), 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 $\Phi \vdash_{\mathcal{T}_0} \mathrm{A}$.

First-Order Standard Tableaux ($\mathcal{T}_1$)

- **Definition 1.6.** The standard tableau calculus ($\mathcal{T}_1$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the following quantifier rules:

$$\frac{(\forall X.A)^{\mathsf{T}} \quad C \in cwff_\iota(\Sigma_\iota)}{([C/X](A))^{\mathsf{T}}} \; \mathcal{T}_1\forall \qquad \frac{(\forall X.A)^{\mathsf{F}} \quad c \in \Sigma_0^{sk} \text{ new}}{([c/X](A))^{\mathsf{F}}} \; \mathcal{T}_1\exists$$

- **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 1.7.** The free variable tableau calculus ($\mathcal{T}_1^f$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the quantifier rules:

$$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \quad Y \text{ new}}{([Y/X](\mathbf{A}))^{\mathsf{T}}}\,\mathcal{T}_1^f\forall \qquad \frac{(\forall X.\mathbf{A})^{\mathsf{F}} \quad \mathrm{free}(\forall X.\mathbf{A}) = \{X^1,\ldots,X^k\} \quad f\in\Sigma_k^{sk} \text{ new}}{([f(X^1,\ldots,X^k)/X](\mathbf{A}))^{\mathsf{F}}}\,\mathcal{T}_1^f\exists$$

and generalizes its cut rule $\mathcal{T}_0\bot$ to:

$$\frac{\begin{array}{l}\mathbf{A}^\alpha \\ \mathbf{B}^\beta\end{array} \quad \alpha\neq\beta \quad \sigma(\mathbf{A})=\sigma(\mathbf{B})}{\bot : \sigma}\,\mathcal{T}_1^f\bot$$

$\mathcal{T}_1^f\bot$ 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)

Free variable Tableaux ($\mathcal{T}_1^f$): Derivable Rules

- **Definition 1.8.** Derivable quantifier rules in $\mathcal{T}_1^f$:

$$\frac{(\exists X.\mathrm{A})^\mathsf{T} \quad \mathrm{free}(\forall X.\mathrm{A}) = \{X^1, \ldots, X^k\} \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X^1, \ldots, X^k)/X](\mathrm{A}))^\mathsf{T}}$$

$$\frac{(\exists X.\mathrm{A})^\mathsf{F} \quad Y \text{ new}}{([Y/X](\mathrm{A}))^\mathsf{F}}$$

Tableau Reasons about Blocks

- **Example 1.9 (Reasoning about Blocks).** Returing to slide 400

Can we prove $\mathrm{red}(\mathsf{A})$ from $\forall x.\mathrm{block}(x) \Rightarrow \mathrm{red}(x)$ and $\mathrm{block}(\mathsf{A})$?

$$
\begin{array}{c|c}
\multicolumn{2}{c}{(\forall X.\mathrm{block}(X) \Rightarrow \mathrm{red}(X))^{\mathsf{T}}} \\
\multicolumn{2}{c}{\mathrm{block}(\mathsf{A})^{\mathsf{T}}} \\
\multicolumn{2}{c}{\mathrm{red}(\mathsf{A})^{\mathsf{F}}} \\
\multicolumn{2}{c}{(\mathrm{block}(Y) \Rightarrow \mathrm{red}(Y))^{\mathsf{T}}} \\
\mathrm{block}(Y)^{\mathsf{F}} & \mathrm{red}(\mathsf{A})^{\mathsf{T}} \\
\bot : [\mathsf{A}/Y] & \bot
\end{array}
$$

15.1.2 First-Order Unification

Unification (Definitions)

- **Definition 1.10.** For given terms A and B, unification is the problem of finding a substitution σ, such that $\sigma(\mathrm{A}) = \sigma(\mathrm{B})$.
- **Notation:** We write term pairs as $\mathrm{A}{=}^{?}\mathrm{B}$ e.g. $f(X){=}^{?}f(g(Y))$.
- **Definition 1.11.** Solutions (e.g. $[g(a)/X],[a/Y]$, $[g(g(a))/X],[g(a)/Y]$, or $[g(Z)/X],[Z/Y]$) are called unifiers, $\mathrm{U}(\mathrm{A}{=}^{?}\mathrm{B}):=\{\sigma|\sigma(\mathrm{A}) = \sigma(\mathrm{B})\}$.
- **Idea:** Find representatives in $\mathrm{U}(\mathrm{A}{=}^{?}\mathrm{B})$, that generate the set of solutions.
- **Definition 1.12.** 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 1.13.** σ is called a most general unifier (mgu) of A and B, iff it is minimal in $\mathrm{U}(\mathrm{A}{=}^{?}\mathrm{B})$ wrt. $\leq[(\mathrm{free}(\mathrm{A}) \cup \mathrm{free}(\mathrm{B}))]$.

Unification Problems ($\widehat{=}$ Equational Systems)

- **Idea:** Unification is equation solving.
- **Definition 1.14.** We call a formula $\mathrm{A}^1 =^? \mathrm{B}^1 \wedge \ldots \wedge \mathrm{A}^n =^? \mathrm{B}^n$ an unification problem iff $\mathrm{A}^i, \mathrm{B}^i \in wff_\iota(\Sigma_\iota, \mathcal{V}_\iota)$.
- **Note:** We consider unification problems as sets of equations ($\wedge$ is ACI), and equations as two-element multisets ($=^?$ is C).
- **Definition 1.15.** A substitution is called a unifier for a unification problem $\mathcal{E}$ (and thus $\mathcal{D}$ unifiable), iff it is a (simultaneous) unifier for all pairs in $\mathcal{E}$.

Solved forms and Most General Unifiers

- **Definition 1.16.** We call a pair $A{=}^{?}B$ solved in a unification problem $\mathcal{E}$, iff $A = X$, $\mathcal{E} = X{=}^{?}A \wedge \mathcal{E}$, and $X \notin (\mathrm{free}(A) \cup \mathrm{free}(\mathcal{E}))$. We call an unification problem $\mathcal{E}$ a solved form, iff all its pairs are solved.
- **Lemma 1.17.** *Solved forms are of the form $X^1{=}^{?}B^1 \wedge \ldots \wedge X^n{=}^{?}B^n$ where the X^i are distinct and $X^i \notin \mathrm{free}(B^j)$.*
- **Definition 1.18.** Any substitution $\sigma = [B^1/X^1], \ldots, [B^n/X^n]$ induces a solved unification problem $\mathcal{E}_\sigma := (X^1{=}^{?}B^1 \wedge \ldots \wedge X^n{=}^{?}B^n)$.
- **Lemma 1.19.** *If $\mathcal{E} = X^1{=}^{?}B^1 \wedge \ldots \wedge X^n{=}^{?}B^n$ is a solved form, then $\mathcal{E}$ has the unique most general unifier $\sigma_\mathcal{E} := [B^1/X^1], \ldots, [B^n/X^n]$.*
- *Proof:* Let $\theta \in \mathrm{U}(\mathcal{E})$
  1. then $\theta(X^i) = \theta(B^i) = \theta \circ \sigma_\mathcal{E}(X^i)$
  2. and thus $\theta = (\theta \circ \sigma_\mathcal{E})[\mathrm{supp}(\sigma)]$.
- **Note:** We can rename the introduced variables in most general unifiers!

Unification Algorithm

- **Definition 1.20.** The inference system $\mathcal{U}$ consists of the following rules:

$$\frac{\mathcal{E} \wedge f(\mathsf{A}^1, \ldots, \mathsf{A}^n) =^? f(\mathsf{B}^1, \ldots, \mathsf{B}^n)}{\mathcal{E} \wedge \mathsf{A}^1 =^? \mathsf{B}^1 \wedge \ldots \wedge \mathsf{A}^n =^? \mathsf{B}^n}\ \mathcal{U}\mathrm{dec} \qquad \frac{\mathcal{E} \wedge \mathsf{A} =^? \mathsf{A}}{\mathcal{E}}\ \mathcal{U}\mathrm{triv}$$

$$\frac{\mathcal{E} \wedge X =^? \mathsf{A} \quad X \notin \mathrm{free}(\mathsf{A}) \quad X \in \mathrm{free}(\mathcal{E})}{[\mathsf{A}/X](\mathcal{E}) \wedge X =^? \mathsf{A}}\ \mathcal{U}\mathrm{elim}$$

- **Lemma 1.21.** *$\mathcal{U}$ is correct: $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ implies $\mathrm{U}(\mathcal{F}) \subseteq \mathrm{U}(\mathcal{E})$.*
- **Lemma 1.22.** *$\mathcal{U}$ is complete: $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ implies $\mathrm{U}(\mathcal{E}) \subseteq \mathrm{U}(\mathcal{F})$.*
- **Lemma 1.23.** *$\mathcal{U}$ is confluent: the order of derivations does not matter.*
- **Corollary 1.24.** *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.

Unification Examples

- **Example 1.25.** Two similar unification problems:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f(g(X,X),h(a)) =^? f(g(a,Z),h(Z))}{g(X,X) =^? g(a,Z) \wedge h(a) =^? h(Z)}\,\mathcal{U}\mathrm{dec}}{X =^? a \wedge X =^? Z \wedge h(a) =^? h(Z)}\,\mathcal{U}\mathrm{dec}}{X =^? a \wedge X =^? Z \wedge a =^? Z}\,\mathcal{U}\mathrm{dec}}{X =^? a \wedge a =^? Z \wedge a =^? Z}\,\mathcal{U}\mathrm{elim}}{X =^? a \wedge Z =^? a \wedge a =^? a}\,\mathcal{U}\mathrm{elim}}{X =^? a \wedge Z =^? a}\,\mathcal{U}\mathrm{triv}$$	$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f(g(X,X),h(a)) =^? f(g(b,Z),h(Z))}{g(X,X) =^? g(b,Z) \wedge h(a) =^? h(Z)}\,\mathcal{U}\mathrm{dec}}{X =^? b \wedge X =^? Z \wedge h(a) =^? h(Z)}\,\mathcal{U}\mathrm{dec}}{X =^? b \wedge X =^? Z \wedge a =^? Z}\,\mathcal{U}\mathrm{dec}}{X =^? b \wedge b =^? Z \wedge a =^? Z}\,\mathcal{U}\mathrm{elim}}{X =^? b \wedge Z =^? b \wedge a =^? b}\,\mathcal{U}\mathrm{elim}$$
MGU: $[a/X], [a/Z]$	$a =^? b$ not unifiable

Unification (Termination)

- **Definition 1.26.** 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 1.27.** We call a variable X solved in an unification problem $\mathcal{E}$, iff $\mathcal{E}$ contains a solved pair $X =^? A$.
- **Lemma 1.28.** *If $\prec$ is linear/terminating on S, then $\prec^m$ is linear/terminating on $\mathcal{P}(S)$.*
- **Lemma 1.29.** *$\mathcal{U}$ 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.

First-Order Unification is Decidable

- **Definition 1.30.** 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 1.31.** *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
  3. $\mathrm{A}=X\in\mathrm{free}(\mathcal{E})$
     3.1. then $\mathcal{U}\mathrm{elim}$ (if $\mathrm{B}\neq X$) or $\mathcal{U}\mathrm{triv}$ (if $\mathrm{B}=X$) is applicable.
- **Corollary 1.32.** *First-order unification is decidable in PL^1.*

  Proof:
- 1. $\mathcal{U}$-irreducible unification problems can be reached in finite time by 1.29.
  2. They are either solved or unsolvable by 1.31, so they provide the answer.

15.1.3 Efficient Unification

Complexity of Unification

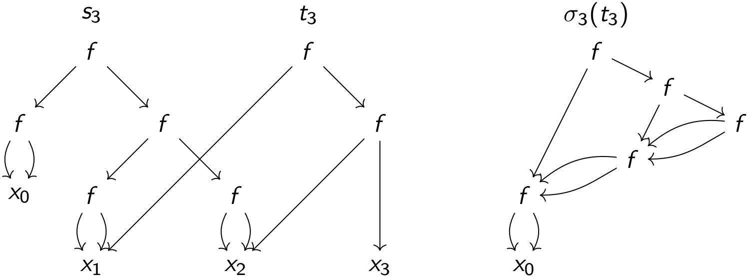
- **Observation:** Naive implementations of unification are exponential in time and space.
- **Example 1.33.** Consider the terms

$$\begin{aligned} s_n &= f(f(x_0,x_0),f(f(x_1,x_1),f(\ldots,f(x_{n-1},x_{n-1}))\ldots)) \\ t_n &= f(x_1,f(x_2,f(x_3,f(\cdots,x_n)\cdots))) \end{aligned}$$

  - The most general unifier of s_n and t_n is
    $\sigma_n := [f(x_0,x_0)/x_1],[f(f(x_0,x_0),f(x_0,x_0))/x_2],[f(f(f(x_0,x_0),f(x_0,x_0)),f(f(x_0,x_0),f(x_0,x_0$
  - 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 1.34.** *Terms can be transformed into DAGs in linear time.*
- **Example 1.35.** Continuing from 1.33 … 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 $\rightsquigarrow$ subterm duplication.
- **Idea:** Replace $\mathcal{U}\mathrm{elim}$ the notion of solved forms by something better.
- **Definition 1.36.** We say that $X^1 =^? \mathrm{B}^1 \wedge \ldots \wedge X^n =^? \mathrm{B}^n$ is a DAG solved form, iff the X^i are distinct and $X^i \notin \mathrm{free}(\mathrm{B}^j)$ for $i \leq j$.
- **Definition 1.37.** The inference system $\mathcal{DU}$ contains rules $\mathcal{U}\mathrm{dec}$ and $\mathcal{U}\mathrm{triv}$ from $\mathcal{U}$ plus the following:

$$\frac{\mathcal{E} \wedge X =^? \mathrm{A} \wedge X =^? \mathrm{B} \quad \mathrm{A}, \mathrm{B} \notin \mathcal{V}_\iota \quad |\mathrm{A}| \leq |\mathrm{B}|}{\mathcal{E} \wedge X =^? \mathrm{A} \wedge \mathrm{A} =^? \mathrm{B}} \ \mathcal{DU}\mathrm{merge}$$

$$\frac{\mathcal{E} \wedge X =^? Y \quad X \neq Y \quad X, Y \in \mathrm{free}(\mathcal{E})}{[Y/X](\mathcal{E}) \wedge X =^? Y} \ \mathcal{DU}\mathrm{evar}$$

  where $|\mathrm{A}|$ is the number of symbols in A.
- The analysis for $\mathcal{U}$ applies mutatis mutandis.

Unification by DAG-chase

- **Idea:** Extend the Input-DAGs by edges that represent unifiers.
- **Definition 1.38.** 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.

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 1.39.** 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 1.40.** 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)

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.

15.1.4 Implementing First-Order Tableaux

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 1.41.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*

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 1.46.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.47.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.

Start, close left branch	use $\mathcal{T}_1^f\forall$ again (right branch)
$\begin{array}{c} ((p(a) \vee p(b)) \Rightarrow (\exists.p()))^{\mathsf{F}} \\ (p(a) \vee p(b))^{\mathsf{T}} \\ (\exists x.p(x))^{\mathsf{F}} \\ (\forall x.\neg p(x))^{\mathsf{T}} \\ \neg p(y)^{\mathsf{T}} \\ p(y)^{\mathsf{F}} \\ \begin{array}{c\|c} p(a)^{\mathsf{T}} & p(b)^{\mathsf{T}} \\ \bot : [a/y] & \end{array} \end{array}$	$\begin{array}{c} ((p(a) \vee p(b)) \Rightarrow (\exists.p()))^{\mathsf{F}} \\ (p(a) \vee p(b))^{\mathsf{T}} \\ (\exists x.p(x))^{\mathsf{F}} \\ (\forall x.\neg p(x))^{\mathsf{T}} \\ \neg p(a)^{\mathsf{T}} \\ p(a)^{\mathsf{F}} \\ \begin{array}{c\|c} p(a)^{\mathsf{T}} & p(b)^{\mathsf{T}} \\ \bot : [a/y] & \neg p(z)^{\mathsf{T}} \\ & p(z)^{\mathsf{F}} \\ & \bot : [b/z] \end{array} \end{array}$

After we have used up $p(y)^{\mathsf{F}}$ by applying $[a/y]$ in $\mathcal{T}_1^f\bot$, we have to get a new instance $p(z)^{\mathsf{F}}$ via $\mathcal{T}_1^f\forall$.

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 1.51.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.52.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.
- **Definition 1.53.** 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 1.54.** *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 $\forall$.

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 1.56.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.57.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.
- **Definition 1.58.** 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 1.59.** *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 1.60.** *$\mathcal{T}_1^f$ 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 1.42.

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 1.61.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.62.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.
- **Definition 1.63.** 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 1.64.** *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 1.65.** *$\mathcal{T}_1^f$ 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 1.42.
- **Remark:** Otherwise validity in PL^1 would be decidable.
- **Implementation:** We need an iterative multiplicity deepening process.

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 1.66.** Choosing which matters – this tableau does not close!

$$
\begin{array}{c}
(\exists x.(p(a) \land p(b) \Rightarrow p()) \land (q(b) \Rightarrow q(x)))^{\mathsf{F}} \\
((p(a) \land p(b) \Rightarrow p()) \land (q(b) \Rightarrow q(y)))^{\mathsf{F}} \\
\begin{array}{c|c}
(p(a) \Rightarrow p(b) \Rightarrow p())^{\mathsf{F}} & (q(b) \Rightarrow q(y))^{\mathsf{F}} \\
p(a)^{\mathsf{T}} & q(b)^{\mathsf{T}} \\
p(b)^{\mathsf{T}} & q(y)^{\mathsf{F}} \\
p(y)^{\mathsf{F}} & \\
\bot : [a/y] &
\end{array}
\end{array}
$$

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 1.67.** *$\mathcal{T}_1^f$ 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 1.68.**
  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 1.69.** *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.

15.2 First-Order Resolution

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:

$$\frac{(\forall X.\mathrm{A})^{\mathsf{T}} \vee \mathrm{C} \quad Z \notin (\mathrm{free}(\mathrm{A}) \cup \mathrm{free}(\mathrm{C}))}{([Z/X](\mathrm{A}))^{\mathsf{T}} \vee \mathrm{C}}$$

$$\frac{(\forall X.\mathrm{A})^{\mathsf{F}} \vee \mathrm{C} \quad \{X_1, \ldots, X_k\} = \mathrm{free}(\forall X.\mathrm{A}) \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X_1, \ldots, X_k)/X](\mathrm{A}))^{\mathsf{F}} \vee \mathrm{C}}$$

  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:

$$\frac{\mathrm{A}^{\mathsf{T}} \vee \mathrm{C} \quad \mathrm{B}^{\mathsf{F}} \vee \mathrm{D} \quad \sigma = \mathrm{mgu}(\mathrm{A}, \mathrm{B})}{(\sigma(\mathrm{C})) \vee (\sigma(\mathrm{D}))} \qquad \frac{\mathrm{A}^{\alpha} \vee \mathrm{B}^{\alpha} \vee \mathrm{C} \quad \sigma = \mathrm{mgu}(\mathrm{A}, \mathrm{B})}{(\sigma(\mathrm{A})) \vee (\sigma(\mathrm{C}))}$$

First-Order CNF – Derived Rules

- **Definition 2.3.** The following inference rules are derivable from the ones above via $(\exists X.A) = \neg(\forall X.\neg A)$:

$$\frac{(\exists X.A)^{\mathsf{T}} \vee C \quad \{X_1, \ldots, X_k\} = \mathrm{free}(\forall X.A) \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X_1, \ldots, X_k)/X](A))^{\mathsf{T}} \vee C}$$

$$\frac{(\exists X.A)^{\mathsf{F}} \vee C \quad Z \notin (\mathrm{free}(A) \cup \mathrm{free}(C))}{([Z/X](A))^{\mathsf{F}} \vee C}$$

15.2.1 Resolution Examples

Col. West, *a Criminal?*

- **Example 2.4.** From [RN09]

  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 $50\mathrm{ms}$.
- **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 $P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{T}}$.

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)

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: $\mathrm{mle}(X_2) \wedge \mathrm{own}(\mathrm{NN}, X_2) \Rightarrow \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NN})$

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: $\mathrm{mle}(X_3) \Rightarrow \mathrm{weap}(X_3)$

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: $\mathrm{enmy}(X_4, \mathrm{USA}) \Rightarrow \mathrm{host}(X_4)$

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: $\mathrm{enmy}(X_4, \mathrm{USA}) \Rightarrow \mathrm{host}(X_4)$
- *West is an American:*
  Clause: $\mathrm{ami}(\mathrm{West})$

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*:
  $\mathrm{enmy}(\mathrm{NN}, \mathrm{USA})$

Col. West, *a Criminal!* PL1 Resolution Proof

$\mathrm{ami}(X_1)^F \vee \mathrm{weapon}(Y_1)^F \vee \mathrm{sell}(X_1, Y_1, Z_1)^F \vee \mathrm{hostile}(Z_1)^F \vee \mathrm{crook}(X_1)^T \quad \mathrm{crook}(\mathrm{West})^F$

$[\mathrm{West}/X_1]$

$\mathrm{ami}(\mathrm{West})^T \quad \mathrm{ami}(\mathrm{West})^F \vee \mathrm{weapon}(Y_1)^F \vee \mathrm{sell}(\mathrm{West}, Y_1, Z_1)^F \vee \mathrm{hostile}(Z_1)^F$

$\mathrm{missile}(X_3)^F \vee \mathrm{weapon}(X_3)^T \quad \mathrm{weapon}(Y_1)^F \vee \mathrm{sell}(\mathrm{West}, Y_1, Z_1)^F \vee \mathrm{hostile}(Z_1)^F$

$[Y_1/X_3]$

$\mathrm{missile}(c)^T \quad \mathrm{missile}(Y_1)^F \vee \mathrm{sell}(\mathrm{West}, Y_1, Z_1)^F \vee \mathrm{hostile}(Z_1)^F$

$[c/Y_1]$

$\mathrm{missile}(X_2)^F \vee \mathrm{own}(\mathrm{NoNo}, X_2)^F \vee \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NoNo})^T$

$\mathrm{sell}(\mathrm{West}, c, Z_1)^F \vee \mathrm{hostile}(Z_1)^F$

$[c/X_2]$

$[\mathrm{NoNo}/Z_1]$

$\mathrm{missile}(c)^T \quad \mathrm{missile}(c)^F \vee \mathrm{own}(\mathrm{NoNo}, c)^F \vee \mathrm{hostile}(\mathrm{NoNo})^F$

$\mathrm{own}(\mathrm{NoNo}, c)^T \quad \mathrm{own}(\mathrm{NoNo}, c)^F \vee \mathrm{hostile}(\mathrm{NoNo})^F$

$\mathrm{enemy}(X_4, USA)^F \vee \mathrm{hostile}(X_4)^T \quad \mathrm{hostile}(\mathrm{NoNo})^F$

$[\mathrm{NoNo}/X_4]$

$\mathrm{enemy}(\mathrm{NoNo}, USA)^T \quad \mathrm{enemy}(\mathrm{NoNo}, USA)^F$

$\square$

Curiosity Killed the Cat?

- **Example 2.5.** From [RN09]

  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.

Curiosity Killed the Cat? Clauses

- *Everyone who loves all animals is loved by someone*:
  $\forall X.(\forall Y.\mathrm{animal}(Y) \Rightarrow \mathrm{love}(X,Y)) \Rightarrow (\exists.\mathrm{love}(Z,X))$
  Clauses: $\mathrm{animal}(g(X_1))^{\mathsf{T}} \vee \mathrm{love}(g(X_1),X_1)^{\mathsf{T}}$ and $\mathrm{love}(X_2,f(X_2))^{\mathsf{F}} \vee \mathrm{love}(g(X_2),X_2)^{\mathsf{T}}$
- *Anyone who kills an animal is loved by noone*:
  $\forall X.\exists Y.\mathrm{animal}(Y) \wedge \mathrm{kill}(X,Y) \Rightarrow (\forall.\neg\mathrm{love}(Z,X))$
  Clause: $\mathrm{animal}(Y_3)^{\mathsf{F}} \vee \mathrm{kill}(X_3,Y_3)^{\mathsf{F}} \vee \mathrm{love}(Z_3,X_3)^{\mathsf{F}}$
- *Jack loves all animals*:
  Clause: $\mathrm{animal}(X_4)^{\mathsf{F}} \vee \mathrm{love}(\mathrm{jack},X_4)^{\mathsf{T}}$
- *Cats are animals*:
  Clause: $\mathrm{cat}(X_5)^{\mathsf{F}} \vee \mathrm{animal}(X_5)^{\mathsf{T}}$
- *Either Jack or curiosity killed the cat (whose name is "Garfield")*:
  Clauses: $\mathrm{kill}(\mathrm{jack},\mathrm{garf})^{\mathsf{T}} \vee \mathrm{kill}(\mathrm{curiosity},\mathrm{garf})^{\mathsf{T}}$ and $\mathrm{cat}(\mathrm{garf})^{\mathsf{T}}$

Curiosity Killed the Cat! PL1 Resolution Proof

$\mathrm{cat}(\mathrm{garf})^{\mathsf{T}}$ $\quad$ $\mathrm{cat}(X_5)^{\mathsf{F}} \vee \mathrm{anl}(X_5)^{\mathsf{T}}$

$[\mathrm{garf}/X_5]$

$\mathrm{anl}(\mathrm{garf})^{\mathsf{T}}$ $\quad$ $\mathrm{anl}(Y_3)^{\mathsf{F}} \vee \mathrm{kill}(X_3, Y_3)^{\mathsf{F}} \vee \mathrm{love}(Z_3, X_3)^{\mathsf{F}}$

$[\mathrm{garf}/Y_3]$

$\mathrm{kill}(X_3, \mathrm{garf})^{\mathsf{F}} \vee \mathrm{love}(Z_3, X_3)^{\mathsf{F}}$ $\quad$ $\mathrm{kill}(\mathrm{jack}, \mathrm{garf})^{\mathsf{T}} \vee \mathrm{kill}(\mathrm{curty}, \mathrm{garf})^{\mathsf{T}}$ $\quad$ $\mathrm{kill}(\mathrm{curty}, \mathrm{garf})^{\mathsf{F}}$

$[\mathrm{jack}/X_3]$ $\quad$ $\mathrm{kill}(\mathrm{jack}, \mathrm{garf})^{\mathsf{T}}$

$\mathrm{love}(Z_3, \mathrm{jack})^{\mathsf{F}}$ $\quad$ $\mathrm{love}(X_2, f(X_2))^{\mathsf{F}} \vee \mathrm{love}(g(X_2), X_2)^{\mathsf{T}}$ $\quad$ $\mathrm{anl}(X_4)^{\mathsf{F}} \vee \mathrm{love}(\mathrm{jack}, X_4)^{\mathsf{T}}$

$[\mathrm{jack}/X_2], [f(\mathrm{jack})/X_4]$

$\mathrm{love}(g(\mathrm{jack}), \mathrm{jack})^{\mathsf{T}} \vee \mathrm{anl}(f(\mathrm{jack}))^{\mathsf{F}}$ $\quad$ $\mathrm{anl}(f(X_1))^{\mathsf{T}} \vee \mathrm{love}(g(X_1), X_1)^{\mathsf{T}}$

$[g(\mathrm{jack})/Z_3]$ $\quad$ $[\mathrm{jack}/X_1]$

$\mathrm{love}(g(\mathrm{jack}), \mathrm{jack})^{\mathsf{T}}$

$\Box$

15.3 Logic Programming as Resolution Theorem Proving

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 $H^{\mathsf{T}} \vee B_1{}^{\mathsf{F}} \vee \ldots \vee B_n{}^{\mathsf{F}}$ (head the only positive literal)*
- **Observation 3.2 (Goal set).** *?− G_1,...,G_n. corresponds to $G_1{}^{\mathsf{F}} \vee \ldots \vee G_n{}^{\mathsf{F}}$*
- **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

$$\frac{P^{\mathsf{T}} \vee \mathrm{A} \quad P^{\mathsf{F}} \vee \mathrm{B}}{\mathrm{A} \vee \mathrm{B}}$$

positive, unit-resulting hyperresolution (PURR)

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.

PROLOG: Our Example

- **Program:**

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

- **Example 3.8 (Query).** ?− fallible(X),greek(X).
- **Answer substitution:** $[\mathrm{sokrates}/X]$

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).
```

$$\dfrac{\dfrac{m(c) \quad w(c,4)}{m(c) \wedge w(c,4)}\wedge I \qquad \dfrac{m(x) \wedge w(x,4) \Rightarrow car()}{m(c) \wedge w(c,4) \Rightarrow car()}\,\mathrm{Subst}}{car(c)}\,\mathrm{MP}$$

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?????

Three Principal Modes of Inference

- **Definition 3.10.** Deduction $\widehat{=}$ knowledge extension
- **Example 3.11.** $\dfrac{rains \Rightarrow wet\_street \quad rains}{wet\_street}\ D$

Three Principal Modes of Inference

- **Definition 3.16.** Deduction $\widehat{=}$ knowledge extension
- **Example 3.17.** $\dfrac{rains \Rightarrow wet\_street \quad rains}{wet\_street} D$
- **Definition 3.18.** Abduction $\widehat{=}$ explanation
- **Example 3.19.** $\dfrac{rains \Rightarrow wet\_street \quad wet\_street}{rains} A$

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.** $\frac{wet\_street \quad rains}{rains \Rightarrow wet\_street} I$

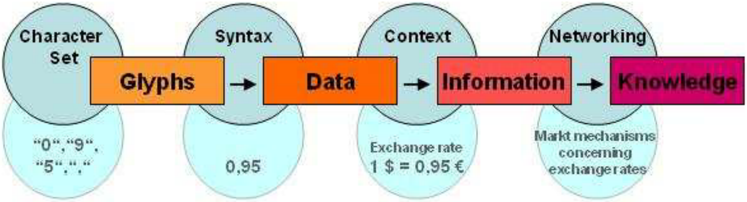
Chapter 16
Knowledge Representation and the Semantic Web

16.1 Introduction to Knowledge Representation

16.1.1 Knowledge & Representation

What is knowledge? Why Representation?

- Lots/all of (academic) disciplines deal with knowledge!
- According to Probst/Raub/Romhardt [PRR97]



- **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

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 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.

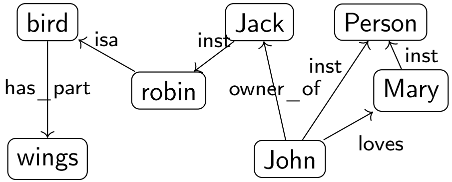
KR Approaches/Evaluation Criteria

- **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.

16.1.2 Semantic Networks

Semantic Networks [CQ69]

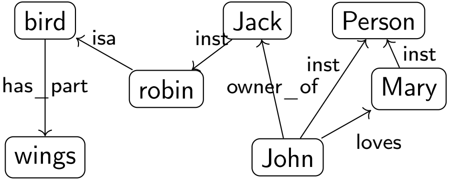
- **Definition 1.2.** A semantic network is a directed graph for representing knowledge:
  - nodes represent objects and concepts (classes of objects) (e.g. John (object) and bird (concept))
  - edges (called links) represent relations between these (isa, father_of, belongs_to)
- **Example 1.3.** A semantic 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.

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*.



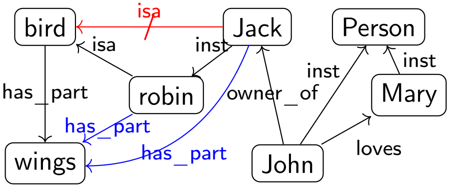
- **Idea:** Links labeled with "isa" and "inst" are special: they propagate properties encoded by other links.
- **Definition 1.6.** We call links labeled by
  - "isa" an inclusion or isa link (inclusion of concepts)
  - "inst" instance or inst link (concept membership)

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.

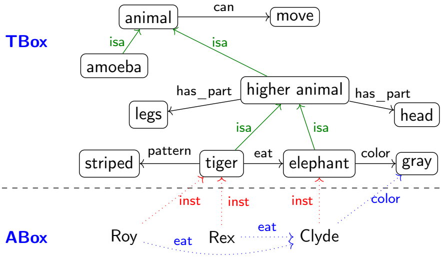
  The process of deriving new concepts and relations from existing ones is called inference and concepts/relations that are only available via inference implicit (in a semantic network).
- **Intuition:** Derived relations represent knowledge that is implicit in the network; they could be added, but usually 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.

Terminologies and Assertions

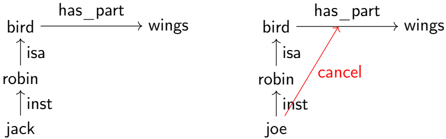
- *Remark 1.9.* We should distinguish concepts from objects.
- **Definition 1.10.** We call the subgraph of a semantic network N spanned by the isa links and relations between concepts the terminology (or TBox, or the famous Isa Hierarchy) and the subgraph spanned by the inst links and relations between objects, the assertions (or 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

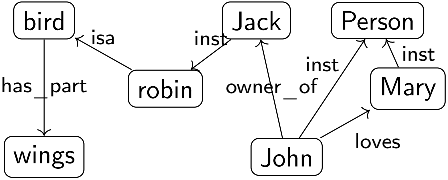
- What is the meaning of a link?
  - link labels are very suggestive (misleading for humans)
  - meaning of link types defined in the process model (no denotational semantics)
- **Problem:** No distinction of optional and defining traits!
- **Example 1.12.** Consider a robin that has lost its wings in 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.**

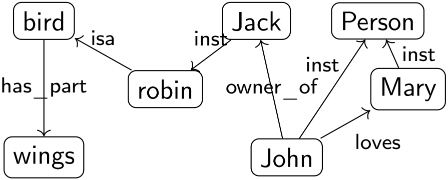


isa(robin,bird)
haspart(bird,wings)
inst(Jack,robin)
owner_of(John, robin)
loves(John,Mary)

- **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)

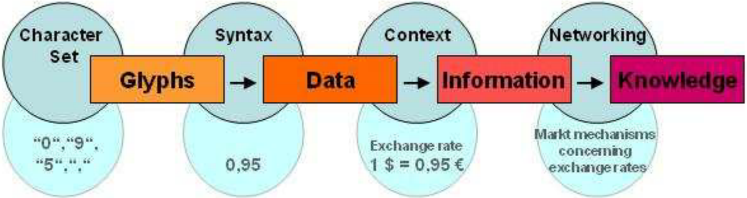
it looks like first-order logic, if we take

  - $a \in S$ 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)

16.1.3 The Semantic Web

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 $\leadsto$ semantic web technology.

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.

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:

  $\Re\sqcup\rangle\sqcup\updownarrow\rceil\top\mathcal{WWW}\in\prime\prime\in$

  $\mathcal{T}\langle\rceil\rceil\updownarrow\rceil\sqsubseteq\rceil\backslash\sqcup\langle\mathcal{I}\backslash\sqcup\rceil\nabla\backslash\dashv\sqcup\rangle\wr\backslash\dashv\updownarrow\mathcal{W}\wr\nabla\updownarrow\lceil\mathcal{W}\rangle\lceil\rceil\mathcal{W}\rceil\lfloor\mathcal{C}\wr\backslash\{\rceil\nabla\rceil\backslash\rfloor\rceil\Re\propto\sqcup\rangle\sqcup\updownarrow\rceil\top$

  $\Re\surd\updownarrow\dashv\rfloor\rceil\top\mathcal{S}\langle\rceil\nabla\dashv\sqcup\wr\backslash\mathcal{W}\dashv\rangle\|\rangle\|\rangle\mathcal{H}\wr\sqcup\rceil\updownarrow\mathcal{H}\wr\backslash\wr\updownarrow\sqcap\updownarrow\sqcap\Leftrightarrow\mathcal{H}\dashv\sqsupseteq\dashv\rangle\rangle\Leftrightarrow\mathcal{USA}\Re\propto\surd\updownarrow\dashv\rfloor\rceil\top$

  $\Re\lceil\dashv\sqcup\rceil\top\nwarrow\infty\infty\mathcal{M}\dashv\dagger\in\prime\prime\in\Re\propto\lceil\dashv\sqcup\rceil\top$

  Given the markup above, a machine agent can

  - parse $\infty\infty\mathcal{M}\dashv\dagger\in\prime\prime\in$ as the date May 7 11 2002 and add this to the user's calendar,
  - parse $\mathcal{S}\langle\rceil\nabla\dashv\sqcup\wr\backslash\mathcal{W}\dashv\rangle\|\rangle\|\rangle\mathcal{H}\wr\sqcup\rceil\updownarrow\mathcal{H}\wr\backslash\wr\updownarrow\sqcap\updownarrow\sqcap\Leftrightarrow\mathcal{H}\dashv\sqsupseteq\dashv\rangle\rangle\Leftrightarrow\mathcal{USA}$ as a destination and find flights.

- **But:** do not be deceived by your ability to understand English!

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⊒⟩⊔‡]∇↕⊣\[⇔⊔⟨]U\⟩⊔][K⟩\}[≀↕⇔⊔⟨]U\⟩⊔][S⊔⊣⊔]∫⇔V⟩]⊔\⊣↕⇔Z⊣⟩∇]

</√⊣∇⊔⟩⌋⟩√⊣\⊔∫>

<⟩\⊔∇≀[⊓⌋⊔⟩≀\>O\⊔⟨]⊔⟨M⊣†H≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈]⊔⟨]⌊⊣]∥[∇≀√≀{⊔⟨]]↕]⊑]\⊔⟨I\⊔]∇↖

\⊣⊔⟩≀\⊣↕W≀∇↕[W⟩⌈]W]⌊C≀\{]∇]\⌋]↙</⟩\⊔∇≀[⊓⌋⊔⟩≀\>

<√∇≀}∇⊣↕>S√]⊣∥]∇∫⌋≀\{⟩∇↕][

<∫√]⊣∥]∇>T⟩↕B]∇\]∇∫↖L]]¬T⟩↕⟩∫⊔⟨]⊒]↕↕∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]W]⌊</∫√]⊣∥]∇>

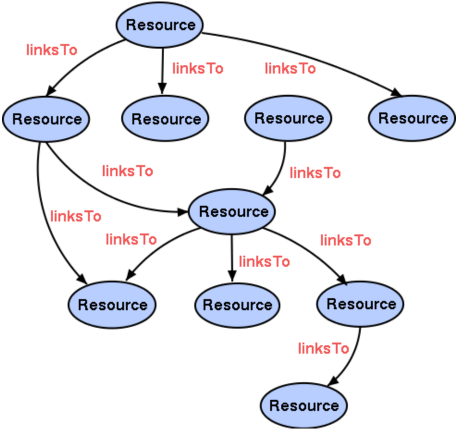
<∫√]⊣∥]∇>I⊣\F≀∫⊔]∇¬I⊣\⟩∫⊔⟨]√⟩≀\]]∇≀{⊔⟨]G∇⟩[⇔⊔⟨]\]§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇↖

\]⊔<∫√]⊣∥]∇>

</√∇≀}∇⊣↕>

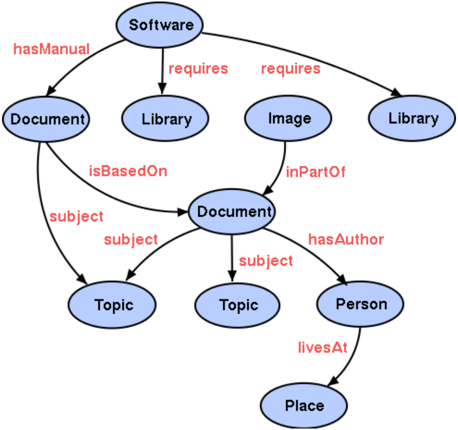
The Current Web

- **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.



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.



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

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

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 [KC04] or RDFa [Her+13] and ontologies on OWL [OWL09]
  - 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)

16.1.4 Other Knowledge Representation Approaches

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

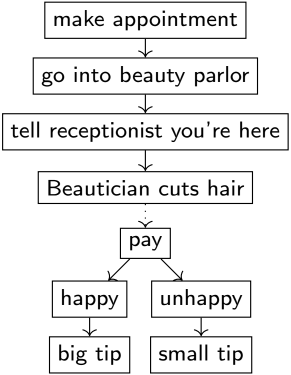
- **Definition 1.22.** Frames (group everything around the object)

```
(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)

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



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

16.2 Logic-Based Knowledge Representation

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)

16.2.1 Propositional Logic as a Set Description Language

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)

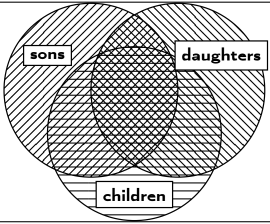
$$\mathcal{L} ::= C \mid \top \mid \bot \mid \overline{\mathcal{L}} \mid \mathcal{L} \sqcap \mathcal{L} \mid \mathcal{L} \sqcup \mathcal{L} \mid \mathcal{L} \sqsubseteq \mathcal{L} \mid \mathcal{L} \equiv \mathcal{L}$$

  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) and $\varphi\colon \mathcal{V}_0 \to \mathcal{P}(\mathcal{D})$, then we define
  - $[\![P]\!] := \varphi(P)$, (remember $\varphi(P) \subseteq \mathcal{D}$).
  - $[\![\mathrm{A} \sqcap \mathrm{B}]\!] := [\![\mathrm{A}]\!] \cap [\![\mathrm{B}]\!]$ and $[\![\overline{\mathrm{A}}]\!] := \mathcal{D} \backslash [\![\mathrm{A}]\!]$ ...
- **Note:** $\langle \mathrm{PL}^0_{\mathrm{DL}}, \mathcal{S}, [\![\cdot]\!] \rangle$, where $\mathcal{S}$ is the class of possible domains forms a logical system.

Concept Axioms

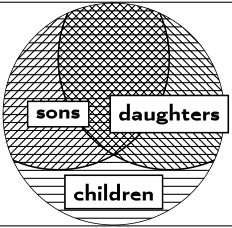
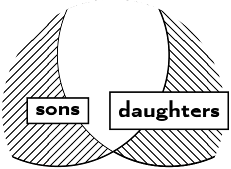
- **Observation:** Set-theoretic semantics of 'true' and 'false' ($\top := \varphi \sqcup \overline{\varphi}$ $\bot := \varphi \sqcap \overline{\varphi}$)

  $[\![\top]\!] = [\![p]\!] \cup [\![\overline{p}]\!] = [\![p]\!] \cup \mathcal{D} \backslash [\![p]\!] = \mathcal{D}$ 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 $\mathcal{D}$ 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
$\mathrm{son} \sqsubseteq \mathrm{child}$ iff $[\![\overline{\mathrm{son}}]\!] \cup [\![\mathrm{child}]\!] = \mathcal{D}$ iff $[\![\mathrm{son}]\!] \subseteq [\![\mathrm{child}]\!]$ $\mathrm{daughter} \sqsubseteq \mathrm{child}$ iff $\left[\!\left[\overline{\mathrm{daughter}}\right]\!\right] \cup [\![\mathrm{child}]\!] = \mathcal{D}$ iff $[\![\mathrm{daughter}]\!] \subseteq [\![\mathrm{child}]\!]$	
$\overline{\mathrm{son} \sqcap \mathrm{daughter}}$ $\mathrm{child} \sqsubseteq \mathrm{son} \sqcup \mathrm{daughter}$	

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	$\overline{\overline{\varphi}} = \varphi$	

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
$\overline{p}^{fo(x)} := p(x)$	
$\overline{\overline{A}}^{fo(x)} := \neg \overline{A}^{fo(x)}$	
$\overline{A \sqcap B}^{fo(x)} := \overline{A}^{fo(x)} \wedge \overline{B}^{fo(x)}$	$\wedge$ vs. $\sqcap$
$\overline{A \sqcup B}^{fo(x)} := \overline{A}^{fo(x)} \vee \overline{B}^{fo(x)}$	$\vee$ vs. $\sqcup$
$\overline{A \sqsubseteq B}^{fo(x)} := \overline{A}^{fo(x)} \Rightarrow \overline{B}^{fo(x)}$	$\Rightarrow$ vs. $\sqsubseteq$
$\overline{A = B}^{fo(x)} := \overline{A}^{fo(x)} \Leftrightarrow \overline{B}^{fo(x)}$	$\Leftrightarrow$ vs. $=$
$\overline{A}^{fo} := (\forall x.\overline{A}^{fo(x)})$	for formulae

Translation Examples

- **Example 2.8.** We translate the concept axioms from 2.6 to fortify our intuition:

$$\overline{\mathrm{son} \sqsubseteq \mathrm{child}}^{fo} = \forall x.\mathrm{son}(x) \Rightarrow \mathrm{child}(x)$$

$$\overline{\mathrm{daughter} \sqsubseteq \mathrm{child}}^{fo} = \forall x.\mathrm{daughter}(x) \Rightarrow \mathrm{child}(x)$$

$$\overline{\overline{\mathrm{son} \sqcap \mathrm{daughter}}}^{fo} = \forall x.\overline{\mathrm{son}(x) \wedge \mathrm{daughter}(x)}$$

$$\overline{\mathrm{child} \sqsubseteq \mathrm{son} \sqcup \mathrm{daughter}}^{fo} = \forall x.\mathrm{child}(x) \Rightarrow (\mathrm{son}(x) \vee \mathrm{daughter}(x))$$

- 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

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 $\langle \mathcal{L}, \mathcal{K}, \models, \mathcal{C} \rangle$ 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)

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 $\sqcap$, $\bar{\cdot}$, $\sqcup$, $\sqsubseteq$, 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.

PL^1 undecideable

decideable

$$\mathrm{PL}^0 \xrightarrow{\varphi} \mathrm{DL} \xrightarrow{\psi} \mathrm{PL}^1$$

$$\psi := \left\{ \begin{array}{l} C \mapsto p \in \Sigma_1^p \\ \sqcap \mapsto \cap \\ \bar{\cdot} \mapsto \mathcal{D} \backslash \cdot \end{array} \right\}$$

$$\varphi := \left\{ \begin{array}{l} X \in \mathcal{V}_0 \mapsto C \\ \wedge \mapsto \sqcap \\ \neg \mapsto \bar{\cdot} \end{array} \right\}$$

- **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.

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.
- **Definition 2.17.** A concept definition $c{=}C$ is called recursive, iff c occurs in C.
- **Example 2.18.** We can define mother$=$woman $\sqcap$ has_child.
- **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.*

16.2.3 Description Logics and Inference

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

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).**

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:** Propositional satisfiability test (NP complete)
  we know how to do this, e.g. tableau, resolution.

Subsumption Test

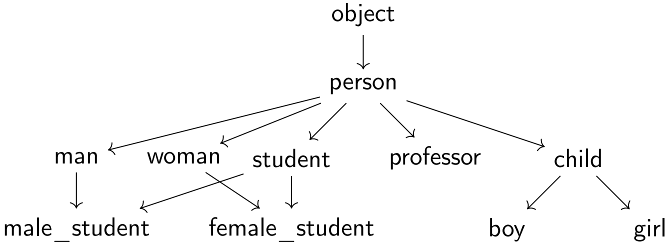
- **Example 2.27.** In this case trivial

axiom	entailed subsumption relation
$\text{man} = \text{person} \sqcap \text{has\_Y}$	$\text{man} \sqsubseteq \text{person}$
$\text{woman} = \text{person} \sqcap \overline{\text{has\_Y}}$	$\text{woman} \sqsubseteq \text{person}$

- **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 $\mathcal{A}$.
- **Reduction to consistency test:** (need to implement only one)
  $\mathcal{A} \Rightarrow (A \Rightarrow B)$ is valid iff $\mathcal{A} \wedge A \wedge \neg B$ is consistentin.
- **Observation:** Or equivalently, iff $\mathcal{A} \Rightarrow B \Rightarrow A$ is valid in PL^0.
- **In our example:** person subsumes woman and man

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)



16.3 A simple Description Logic: ALC

16.3.1 Basic ALC: Concepts, Roles, and Quantification

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 ($\exists$) and universal ($\forall$))
- **Idea:** Use first-order predicate logic (PL^1)

$$\forall x.\mathit{mother}(x) \Leftrightarrow (\mathit{woman}(x) \wedge (\exists y.\mathit{has\_child}(x,y)))$$

- **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*.

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: $F_{\mathcal{ALC}} ::= C \mid \top \mid \bot \mid \overline{F_{\mathcal{ALC}}} \mid F_{\mathcal{ALC}} \sqcap F_{\mathcal{ALC}} \mid F_{\mathcal{ALC}} \sqcup F_{\mathcal{ALC}} \mid \exists R.F_{\mathcal{ALC}} \mid \forall R.F_{\mathcal{ALC}}$

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 $\sqcap$ $\forall$haschild.$\exists$has_child.student
  $\widehat{=}$ grandparents, whose children all have at least one child that is a student

More $\mathcal{ALC}$ Examples

- **Example 3.13.** $\mathrm{car} \sqcap \exists \mathrm{has\_part}.\exists \mathrm{made\_in}.\overline{\mathrm{EU}}$
  $\widehat{=}$ cars that have at least one part that has not been made in the EU
- **Example 3.14.** $\mathrm{student} \sqcap \forall \mathrm{audits\_course}.\mathrm{graduatelevelcourse}$
  $\widehat{=}$ students, that only audit graduate level courses
- **Example 3.15.** $\mathrm{house} \sqcap \forall \mathrm{has\_parking}.\mathrm{off\_street}$ $\widehat{=}$ houses with off-street parking
- **Note:** $p \sqsubseteq q$ can still be used as an abbreviation for $\overline{p} \sqcup q$.
- **Example 3.16.** $\mathrm{student} \sqcap \forall \mathrm{audits\_course}.(\exists \mathrm{hastutorial}.\top \sqsubseteq \forall \mathrm{has\_TA}.\mathrm{woman})$
  $\widehat{=}$ students that only audit courses that either have no tutorial or tutorials that are TAed by women

$\mathcal{ALC}$ 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 3.7 by the production

$$CD_{\mathcal{ALC}} ::= C = F_{\mathcal{ALC}}$$

- **Example 3.18.**

Definition	rec?
$\text{man} = \text{person} \sqcap \exists \text{has\_chrom}.\text{Y\_chrom}$	-
$\text{woman} = \text{person} \sqcap \forall \text{has\_chrom}.\overline{\text{Y\_chrom}}$	-
$\text{mother} = \text{woman} \sqcap \exists \text{has\_child}.\text{person}$	-
$\text{father} = \text{man} \sqcap \exists \text{has\_child}.\text{person}$	-
$\text{grandparent} = \text{person} \sqcap \exists \text{has\_child}.(\text{mother} \sqcup \text{father})$	-
$\text{german} = \text{person} \sqcap \exists \text{has\_parents}.\text{german}$	+
$\text{number\_list} = \text{empty\_list} \sqcup \exists \text{is\_first}.\text{number} \sqcap \exists \text{is\_rest}.\text{number\_list}$	+

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).**

$$
\begin{array}{ll}
 & \textbf{grandparent} \\
\mapsto & \textbf{person} \sqcap \exists\textbf{has\_child}.(\textbf{mother} \sqcup \textbf{father}) \\
\mapsto & \textbf{person} \sqcap \exists\textbf{has\_child}.(\textbf{woman} \sqcap \exists\textbf{has\_child}.\textbf{person} \sqcap \textbf{man} \sqcap \exists\textbf{has\_child}.\textbf{person}) \\
\mapsto & \textbf{person} \sqcap \exists\textbf{has\_child}.(\textbf{person} \sqcap \exists\textbf{has\_chrom}.\textbf{Y\_chrom} \sqcap \exists\textbf{has\_child}.\textbf{person} \sqcap \textbf{person} \sqcap \exists\textbf{has\_chrom}.\textbf{Y\_chrom} \sqcap \exists\textbf{has}
\end{array}
$$

- **Observation 3.22.** *Normalization results can be exponential.* *(contain redundancies)*
- **Observation 3.23.** *Normalization need not terminate on cyclic TBoxes.*
- **Example 3.24.**

$$
\begin{array}{lll}
\text{german} & \mapsto & \text{person} \sqcap \exists\text{has\_parents}.\text{german} \\
 & \mapsto & \text{person} \sqcap \exists\text{has\_parents}.(\text{person} \sqcap \exists\text{has\_parents}.\text{german}) \\
 & \mapsto & \ldots
\end{array}
$$

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 $\langle\mathcal{D},[\![\cdot]\!]\rangle$, where $\mathcal{D}$ is a non-empty set called the domain and $[\![\cdot]\!]$ a mapping called the interpretation, such that

Op.	formula semantics
	$[\![c]\!]\subseteq\mathcal{D}=[\![\top]\!]\quad [\![\bot]\!]=\emptyset\quad [\![r]\!]\subseteq\mathcal{D}\times\mathcal{D}$
$\overline{\cdot}$	$[\![\overline{\varphi}]\!]=\overline{[\![\varphi]\!]}=\mathcal{D}\backslash[\![\varphi]\!]$
$\sqcap$	$[\![\varphi\sqcap\psi]\!]=[\![\varphi]\!]\cap[\![\psi]\!]$
$\sqcup$	$[\![\varphi\sqcup\psi]\!]=[\![\varphi]\!]\cup[\![\psi]\!]$
$\exists \mathrm{R}.$	$[\![\exists\mathrm{R}.\varphi]\!]=\{x\in\mathcal{D}\mid\exists y.\langle x,y\rangle\in[\![\mathrm{R}]\!]\text{ and }y\in[\![\varphi]\!]\}$
$\forall \mathrm{R}.$	$[\![\forall\mathrm{R}.\varphi]\!]=\{x\in\mathcal{D}\mid\forall y.\text{if }\langle x,y\rangle\in[\![\mathrm{R}]\!]\text{ then }y\in[\![\varphi]\!]\}$

- 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 2.7 by the following quantifier rules:

$$\overline{\forall\mathrm{R}.\varphi}^{fo(x)}:=(\forall y.\mathrm{R}(x,y)\Rightarrow\overline{\varphi}^{fo(y)})\qquad\overline{\exists\mathrm{R}.\varphi}^{fo(x)}:=(\exists y.\mathrm{R}(x,y)\wedge\overline{\varphi}^{fo(y)})$$

- **Observation 3.27.** *The set-theoretic semantics from 3.25 and the "semantics-by-translation" from 3.26 induce the same notion of satisfiability.*

$\mathcal{ALC}$ Identities

1	$\overline{\exists R.\varphi} = \forall R.\overline{\varphi}$	3	$\overline{\forall R.\varphi} = \exists R.\overline{\varphi}$
2	$\forall R.(\varphi \sqcap \psi) = \forall R.\varphi \sqcap \forall R.\psi$	4	$\exists R.(\varphi \sqcup \psi) = \exists R.\varphi \sqcup \exists R.\psi$

- Proof of 1

$$
\begin{aligned}
[[\overline{\exists R.\varphi}]] = \mathcal{D} \backslash [\![\exists R.\varphi]\!] &= \mathcal{D} \backslash \{x \in \mathcal{D} \mid \exists y.(\langle x,y\rangle \in [\![R]\!]) \text{ and } (y \in [\![\varphi]\!])\} \\
&= \{x \in \mathcal{D} \mid \text{not } \exists y.(\langle x,y\rangle \in [\![R]\!]) \text{ and } (y \in [\![\varphi]\!])\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x,y\rangle \in [\![R]\!]) \text{ then } (y \notin [\![\varphi]\!])\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x,y\rangle \in [\![R]\!]) \text{ then } (y \in (\mathcal{D} \backslash [\![\varphi]\!]))\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x,y\rangle \in [\![R]\!]) \text{ then } (y \in [\![\overline{\varphi}]\!])\} \\
&= [\![\forall R.\overline{\varphi}]\!]
\end{aligned}
$$

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
$\overline{\exists R.(\forall S.e \sqcap \overline{\forall S.d})}$	
$\mapsto \forall R.\overline{\forall S.e \sqcap \overline{\forall S.d}}$	$\overline{\exists R.\varphi} \mapsto \forall R.\overline{\varphi}$
$\mapsto \forall R.(\overline{\forall S.e} \sqcup \overline{\overline{\forall S.d}})$	$\overline{\varphi \sqcap \psi} \mapsto \overline{\varphi} \sqcup \overline{\psi}$
$\mapsto \forall R.(\exists S.\overline{e} \sqcup \overline{\overline{\forall S.d}})$	$\overline{\forall R.\varphi} \mapsto \exists R.\overline{\varphi}$
$\mapsto \forall R.(\exists S.\overline{e} \sqcup \forall S.d)$	$\overline{\overline{\varphi}} \mapsto \varphi$

$\mathcal{ALC}$ with Assertions about Individuals

- **Definition 3.30.** We define the assertions for $\mathcal{ALC}$
  - Role assertions$a{:}\varphi$ (a is a φ)
  - $a \; \mathrm{R} \; b$ (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 $\perp$).
- **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:

$$\frac{\begin{array}{c}x{:}c\\ x{:}\overline{c}\end{array}}{\perp}\,\mathcal{T}_{\perp} \qquad \frac{x{:}\varphi \sqcap \psi}{\begin{array}{c}x{:}\varphi\\ x{:}\psi\end{array}}\,\mathcal{T}_{\sqcap} \qquad \frac{x{:}\varphi \sqcup \psi}{x{:}\varphi \,\big|\, x{:}\psi}\,\mathcal{T}_{\sqcup} \qquad \frac{\begin{array}{c}x{:}\forall \mathrm{R}.\varphi\\ x \; \mathrm{R} \; y\end{array}}{y{:}\varphi}\,\mathcal{T}_{\forall} \qquad \frac{x{:}\exists \mathrm{R}.\varphi}{\begin{array}{c}x \; \mathrm{R} \; y\\ y{:}\varphi\end{array}}\,\mathcal{T}_{\exists}$$

- To test consistency of a concept φ, normalize φ to ψ, initialize the tableau with $x{:}\psi$, saturate. Open branches $\rightsquigarrow$ consistent. (x arbitrary)

$\mathcal{T}_{\mathcal{ALC}}$ 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
$x{:}\forall \text{has\_child}.\text{man}$	$\mathcal{T}_{\sqcap}$
$x{:}\exists \text{has\_child}.\overline{\text{man}}$	$\mathcal{T}_{\sqcap}$
$x\ \text{has\_child}\ y$	$\mathcal{T}_{\exists}$
$y{:}\overline{\text{man}}$	$\mathcal{T}_{\exists}$
$\perp$	$\mathcal{T}_{\perp}$
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
$x{:}\forall \text{has\_child}.\text{man}$	$\mathcal{T}_{\sqcap}$
$x{:}\exists \text{has\_child}.\text{man}$	$\mathcal{T}_{\sqcap}$
$x\ \text{has\_child}\ y$	$\mathcal{T}_{\exists}$
$y{:}\text{man}$	$\mathcal{T}_{\exists}$
open	

  This tableau shows a model: there are two persons, x and y. y is the only child of x, y is a man.

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

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

1	$x{:}\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has\_child}.\overline{\text{ugrad}}$		initial
2	$x{:}\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad})$		$\mathcal{T}_{\sqcap}$
3	$x{:}\exists \text{has\_child}.\overline{\text{ugrad}}$		$\mathcal{T}_{\sqcap}$
4	$x \text{ has\_child } y$		$\mathcal{T}_{\exists}$
5	$y{:}\overline{\text{ugrad}}$		$\mathcal{T}_{\exists}$
6	$y{:}\text{ugrad} \sqcup \text{grad}$		$\mathcal{T}_{\forall}$
7	$y{:}\text{ugrad}$	$y{:}\text{grad}$	$\mathcal{T}_{\sqcup}$
8	$\perp$	open	

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.

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)

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 $\begin{array}{lll} \mathcal{M}{\models}(x{:}\psi) & \text{iff} & [\![x]\!]\in[\![\psi]\!] \\ \mathcal{M}{\models}x \mathrel{\mathrm{R}} y & \text{iff} & \langle x,y\rangle{\in}[\![\mathrm{R}]\!] \\ \mathcal{M}{\models}S & \text{iff} & \mathcal{I}{\models}c \text{ for all } c{\in}S \end{array}$
  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 $\mathcal{M}\models(y{:}\varphi)$

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

$$\begin{array}{rcl} \mathcal{D} & := & \{x | x{:}\psi \in \mathcal{B} \text{ or } z \mathrel{R} x \in \mathcal{B}\} \\ [\![c]\!] & := & \{x | x{:}c \in \mathcal{B}\} \\ [\![R]\!] & := & \{\langle x, y\rangle | x \mathrel{R} y \in \mathcal{B}\} \end{array}$$

     - well-defined since never $x{:}c, x{:}\overline{c} \in \mathcal{B}$ (otherwise $\mathcal{T}_\perp$ applies)
     - $\mathcal{M}$ satisfies all assertions $x{:}c$, $x{:}\overline{c}$ and $x \mathrel{R} y$, (by construction)
  2. $\mathcal{M} \models (y{:}\psi)$, for all $y{:}\psi \in \mathcal{B}$ (on $k = size(\psi)$ next slide)
  3. $\mathcal{M} \models (x{:}\varphi)$.

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 $\{y \mathrel{\mathrm{R}} z, z{:}\theta\} \subseteq \mathrm{B}$ (z new variable) ($\mathcal{T}_{\exists}$-rules, saturation)
so $\mathcal{M} \models (z{:}\theta)$ and $\mathcal{M} \models y \mathrel{\mathrm{R}} z$, 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 $y \mathrel{\mathrm{R}} z \in \mathrm{B}$ (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 $\mathcal{M} \models (y{:}\forall \mathrm{R}.\theta)$.

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

$$\frac{x{:}c \quad x{:}\overline{c}}{\bot}\,\mathcal{T}_\bot \qquad \frac{x{:}\varphi \sqcap \psi}{x{:}\varphi \quad x{:}\psi}\,\mathcal{T}_\sqcap \qquad \frac{x{:}\varphi \sqcup \psi}{x{:}\varphi \mid x{:}\psi}\,\mathcal{T}_\sqcup \qquad \frac{x{:}\forall \mathsf{R}.\varphi \quad x\;\mathsf{R}\;y}{y{:}\varphi}\,\mathcal{T}_\forall \qquad \frac{x{:}\exists \mathsf{R}.\varphi}{x\;\mathsf{R}\;y \quad y{:}\varphi}\,\mathcal{T}_\exists$$

- *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 $x{:}\forall \mathsf{R}.\psi$ at most as the number of R-successors of x. (those y with x R y above)
  3. The R-successors are generated by $x{:}\exists \mathsf{R}.\theta$ 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.

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 4., the second one from 3. and 2.
- **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)

The functional Algorithm for $\mathcal{ALC}$

- **Observation:** (leads to a better treatment for $\exists$)
  - the $\mathcal{T}_\exists$-rule generates the constraints x 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** ('φ'$\notin S$ or 'ψ'$\notin S$)
    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)

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 $\exists$: $\dfrac{x{:}\exists R.\varphi \quad \mathcal{CA} = \{\alpha_1, \ldots, \alpha_n\}}{\begin{array}{c} y{:}\varphi \\ x\ R\ y \\ y{:}\alpha_1 \\ \vdots \\ y{:}\alpha_n \end{array}}\ \mathcal{T}^{\exists}_{\mathcal{CA}}$ (instead of $\mathcal{T}_{\exists}$)
- **Problem:** $\mathcal{CA}{:}{=}\{\exists R.c\}$ and start tableau with $x{:}d$ (non-termination)

Non-Termination of $\mathcal{T}_{\mathcal{ALC}}$ with Concept Axioms

- **Problem:** $\mathcal{CA} := \{\exists R.c\}$ and start tableau with $x:d$. (non-termination)

$x:d$	start
$x:\exists R.c$	in $\mathcal{CA}$
$x\ R\ y_1$	$\mathcal{T}_\exists$
$y_1:c$	$\mathcal{T}_\exists$
$y_1:\exists R.c$	$\mathcal{T}_{\mathcal{CA}}^\exists$
$y_1\ R\ y_2$	$\mathcal{T}_\exists$
$y_2:c$	$\mathcal{T}_\exists$
$y_2:\exists R.c$	$\mathcal{T}_{\mathcal{CA}}^\exists$
...	

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

Instance Test: Concept Membership

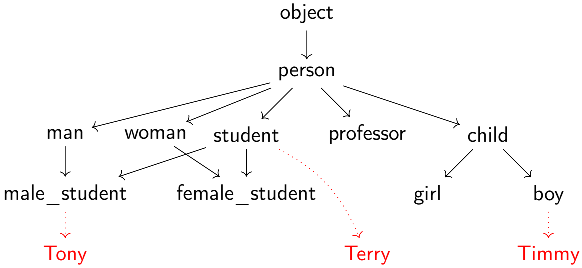
- **Definition 3.46.** An instance test computes whether given an $\mathcal{ALC}$ ontology an individual is a member of a given concept.
- **Example 3.47 (An Ontology).**

TBox (terminological Box)	ABox (assertional Box, data base)	
$\mathrm{woman} = \mathrm{person} \sqcap \overline{\mathrm{has\_Y}}$	tony:person	Tony is a person
$\mathrm{man} = \mathrm{person} \sqcap \mathrm{has\_Y}$	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: $\text{tony}:\forall\text{has\_grad}.\text{genius}$ $\text{tony has\_grad mary}$ $\models \text{mary}:\text{genius}$
information that can be combined with TBox info	TBox: $\text{happy\_prof} = \text{prof} \sqcap \forall\text{has\_grad}.\text{genius}$ ABox: $\text{tony}:\text{happy\_prof}$, $\text{tony has\_grad mary}$ $\models \text{mary}:\text{genius}$

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	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 — $\mathcal{T}_{\forall}$ $\bot$ — $\mathcal{T}_{\bot}$
ABox	tony:happy_prof tony has_grad mary	

- **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)

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

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}*

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.

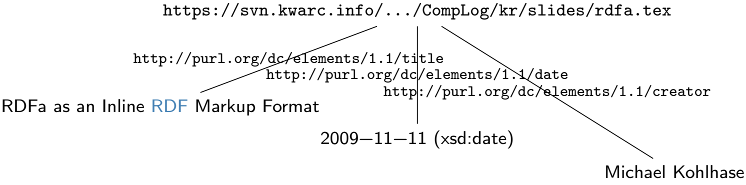
RDFa as an Inline RDF Markup Format

- **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>
```



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 $\mathcal{ALC}$)

OWL as an Ontology Language for the Semantic Web

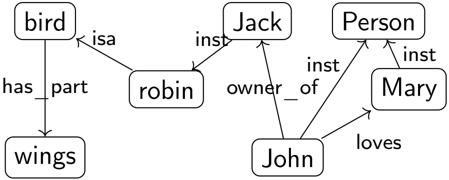
- **Task:** Complement RDF (ABox) with a TBox language.
- **Idea:** Make use of resources that are values in rdf:type. (called Classes)
- **Definition 4.11.** OWL (the ontology web language) is a language for encoding TBox information about RDF classes.
- **Example 4.12 (A concept definition for "Mother").**
  Mother$=$Woman $\sqcap$ Parent is represented as

XML Syntax	Functional Syntax

`<EquivalentClasses>`
`  <Class IRI="Mother"/>`
`  <ObjectIntersectionOf>`
`    <Class IRI="Woman"/>`
`    <Class IRI="Parent"/>`
`  </ObjectIntersectionOf>`
`</EquivalentClasses>`
	EquivalentClasses(
	  :Mother
	  ObjectIntersectionOf(
	    :Woman
	    :Parent
	  )
	)

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*.

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*.

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 [PS08].
- 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.

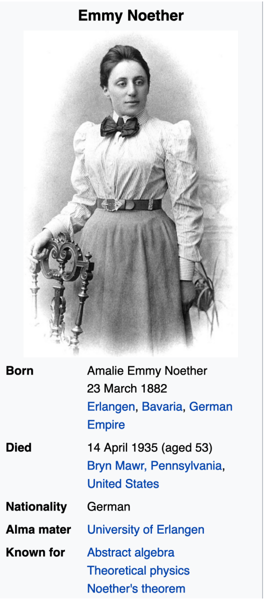
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

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.



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
```

Results: Browse | Go! | Reset

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

Triple Stores: the Semantic Web Databases

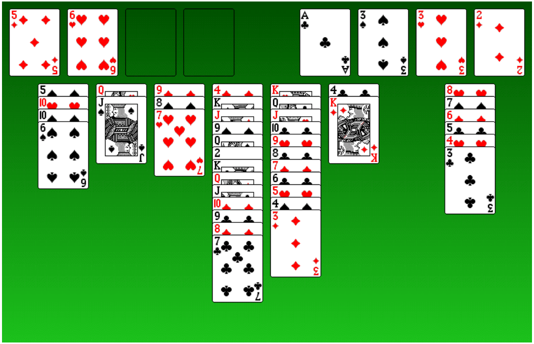
- **Definition 4.19.** A triplestore or RDF store is a purpose-built database for the storage RDF graphs and retrieval of RDF triples usually through variants of SPARQL.
- Common triplestores include
  - Virtuoso: `https://virtuoso.openlinksw.com/` (used in DBpedia)
  - GraphDB: `http://graphdb.ontotext.com/` (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.

Part 4
Planning & Acting

Chapter 17
Planning I: Framework

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.

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?)

17.1 Logic-Based Planning

Fluents: Time-Dependent Knowledge in Planning

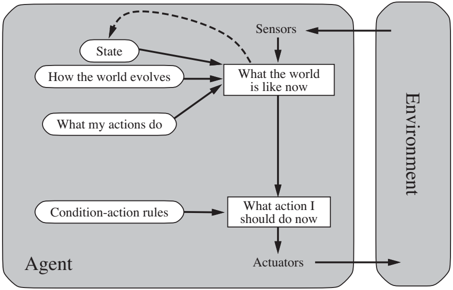
- **Recall from :** We can represent the Wumpus rules in logical systems. (propositional/first-order/ALC)
  - Use inference systems to deduce new world knowledge from percepts and actions.
- **Problem:** Representing (changing) percepts immediately leads to contradictions!
- **Example 1.1.** If the agent moves and a cell with a draft (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 (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 an aspect of the world that changes, all others we call atemporal.

Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percepts, and actions.



Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percepts, 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*

Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percepts, 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)

Fluents: Modeling the Agent's Sensors

- **Idea:** Relate percept fluents to atemporal cell attributes.
- **Example 1.7.** E.g., if the agent perceives a draft 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.
- **Definition 1.8.** We call fluents that describe the agent's sensors sensor axioms.
- **Problem:** Where do fluents like $\mathrm{Ag@}(t, x, y)$ come from?

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 1.7 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, ($\leadsto \mathrm{PL}^{\mathrm{nq}}$)
  2. translate $\mathrm{PL}^{\mathrm{nq}}$ atoms to propositional variables. ($\leadsto \mathrm{PL}^0$)
- **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 1.9.** The action of "going forward" at time t is captured by the fluent $\mathrm{forw}(t)$.
- **Definition 1.10.** Effect axioms describe how the environment changes under an agent's actions.
- **Example 1.11.** 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 $[1,1]$:

$$\mathrm{Ag@}(0,1,1) \wedge \mathrm{faceeast}(0) \wedge \mathrm{forw}(0) \Rightarrow \mathrm{Ag@}(1,2,1) \wedge \neg\mathrm{Ag@}(1,1,1)$$

Generally: (barring exceptions for domain border cells)

$$\forall t,x,y.\mathrm{Ag@}(t,x,y)\wedge\mathrm{faceeast}(t)\wedge\mathrm{forw}(t)\Rightarrow\mathrm{Ag@}(t+1,x+1,y)\wedge\neg\mathrm{Ag@}(t+1,x,y)$$

This compiles down to $16 \cdot t_{\mathrm{end}}$ $\mathrm{PL}^{\mathrm{nq}}/\mathrm{PL}^0$ axioms.

Frames and Frame Axioms

- **Problem:** Effect axioms are not enough.
- **Example 1.12.** Say that the agent has an arrow at time 0, and then moves forward 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 1.13.** 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.

A Hybrid Agent for the Wumpus World

- **Example 1.14 (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.

A Hybrid Agent: Custom Action Selection

- **Example 1.15 (A Hybrid Agent (continued)).** So that we can plan the best strategy:

```
TELL(KB, the temporal "physics" sentences for time t)
safe := {[x,y]|ASK(KB,OK(t,x,y))=T}
if ASK(KB,glitter(t)) = T then
    plan := [grab] + PLAN−ROUTE(current,{[1,1]},safe) + [exit]
if plan is empty then
    unvisited := {[x,y]|ASK(KB,Ag@(t',x,y))=F} for all t'≤t
    plan := PLAN−ROUTE(current,unvisited ∪ safe,safe)
if plan is empty and ASK(KB,havarrow(t)) = T then
    possible_wumpus := {x,y|[x,y]}ASK(KB,¬wumpus(t,x,y)) = F
    plan := PLAN−SHOT(current,possible_wumpus,safe)
if plan is empty then // no choice but to take a risk
    not_unsafe := {[x,y]|ASK(KB,¬OK(t,x,y)) = F}
    plan := PLAN−ROUTE(current,unvisited ∪ not_unsafe,safe)
if plan is empty then
    plan := PLAN−ROUTE(current,{[1,1]},safe) + [exit]
```

Note that OK wumpus, and glitter are fluents, since the Wumpus might have died or the gold might have been grabbed.

A Hybrid Agent: Custom Action Selection

- **Example 1.16 (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)

17.2 Planning: Introduction

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.: $Eq(x,1)$.)

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.: $Eq(x,2)$.)

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 $Eq(x,1)$, eff $Eq(x \wedge 2) \wedge \neg Eq(x,1)$".)

  A logical description of all of these is called a planning task.

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 $Eq(x,1)$, eff $Eq(x \wedge 2) \wedge \neg Eq(x,1)$".)

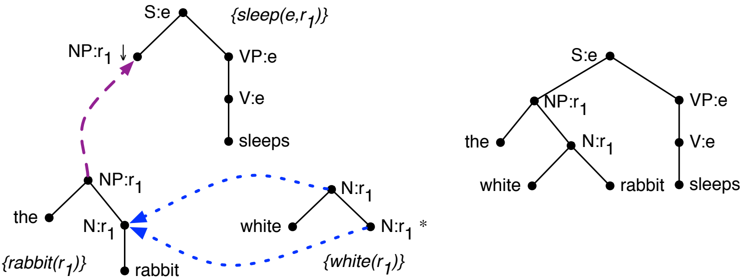
  A logical description of all of these is called a planning task.
- **Definition 2.10.** Solution (plan) $\widehat{=}$ sequence of actions from $\mathcal{A}$, transforming $\mathcal{I}$ into a state that satisfies $\mathcal{G}$. (E.g.: "increment x".)

  The process of finding a plan given a planning task is called planning.

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 [GNT04].

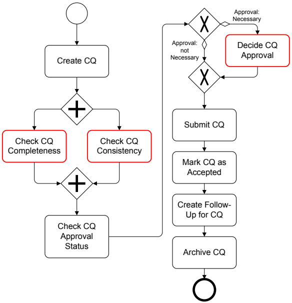
Application: Natural Language Generation



- **Input**: Tree-adjoining grammar, intended meaning.
- **Output**: Sentence expressing that meaning.

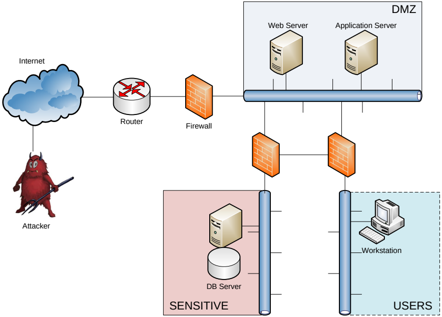
Application: Business Process Templates at SAP

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



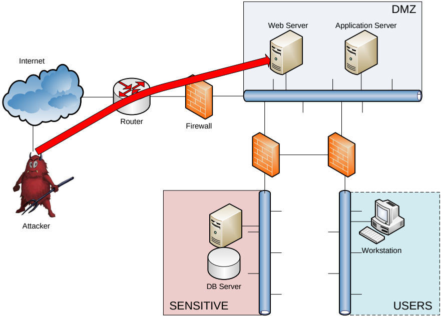
- **Input**: model of behavior of activities on business objects, process endpoint.
- **Output**: Process template leading to this point.

Application: Automatic Hacking



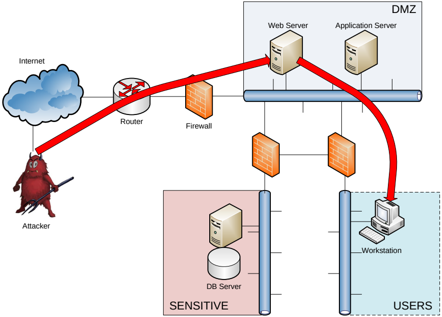
- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

Application: Automatic Hacking



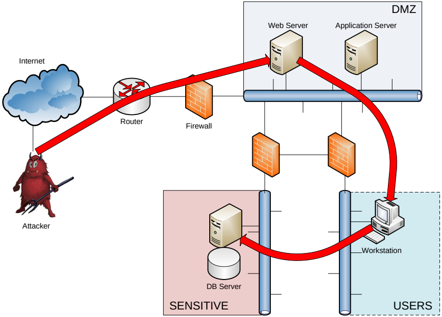
- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

Application: Automatic Hacking



- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

Application: Automatic Hacking



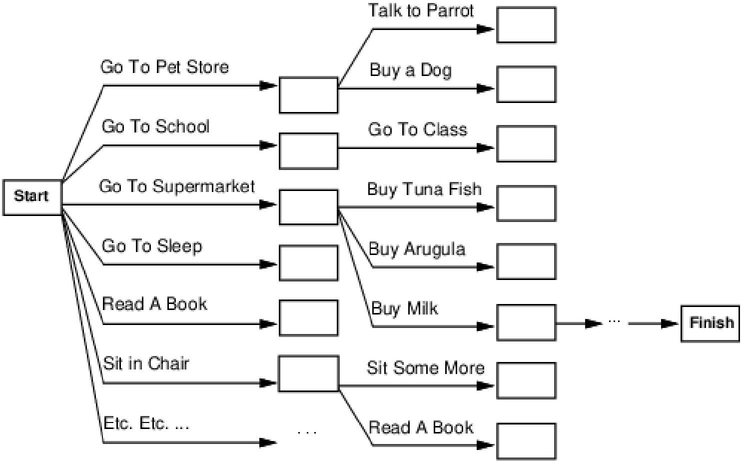
- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

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?

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

- Planning systems do the following:
  1. open up action and goal representation to allow selection
  2. divide-and-conquer by subgoaling

Reminder: Greedy Best-First Search and A^*

- **Recall:** Our heuristic search algorithms(duplicate pruning omitted for simplicity)

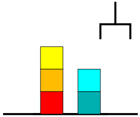
function Greedy_Best−First_Search (problem)
  returns a solution, or failure
  n := node with n.state=problem.InitialState
  frontier := priority queue ordered by ascending h, initially $[n]$
  loop do
    if Empty?(*frontier*) **then return** failure
    n := Pop(*frontier*)
    if problem.GoalTest(n.state) **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 $\rightsquigarrow$ satisficing planning.
- Is A^* optimal? Yes, but only if h is admissible $\rightsquigarrow$ optimal planning, **with such** h.

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
1	1
2	3
3	13
4	73
5	501
6	4051
7	37633
8	394353

blocks	states
9	4596553
10	58941091
11	824073141
12	12470162233
13	202976401213
14	3535017524403
15	65573803186921
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!

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.

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.

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.

17.3 The History of Planning

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:**
  - [NS63] 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.

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 $move(A, B)$; $\leadsto$ $open = \{at(A)\}$ ...
  - **Popular when**: 1990 – 1995.
  - **Approach**: *Starting at goal, extend partially ordered set of actions by inserting achievers for open sub-goals, or by adding ordering constraints to avoid conflicts.*
  - **Keywords/cites**: **UCPOP** [PW92], causal links, flaw selection strategies, ...

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**: [BF95; BF97; Koe+97], action/fact mutexes, step-optimal plans, ...
- **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. ${at(B)_1}^{\mathsf{F}} \vee {move(A, B)_0}^{\mathsf{T}}$; unit clauses to encode initial state ${at(A)_0}^{\mathsf{T}}$, ${at(B)_0}^{\mathsf{T}}$; unit clauses to encode goal ${at(B)_1}^{\mathsf{T}}$.
  - **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**: [KS92; KS98; RHN06; Rin10], SAT encoding schemes, BlackBox, ...

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**: [BG99; HG00; BG01; HN01; Ede01; GSS03; Hel06; HHH07; HG08; KD09; HD09; RW10; NHH11; KHH12a; KHH12b; KHD13; DHK15], critical path heuristics, ignoring delete lists, relaxed plans, landmark heuristics, abstractions, partial delete relaxation, ...

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 [McD+98; FL03; HE05; Ger+09]
  - Problem Corpus: ≈ 50 **domains**, $\gg 1000$ **instances**, 74 (!!) planners in 2011

International Planning Competition

- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?

International Planning Competition

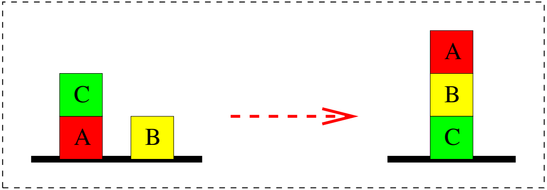
- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?
- **Answer:** Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a "winner"! On other domains and/or according to other criteria, you may well be better off with the "looser".

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)

Planning History, p.s.: Planning is Non-Trivial!

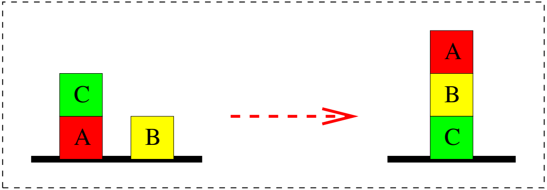
- **Example 3.2.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

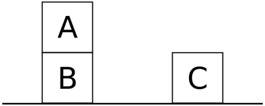
Planning History, p.s.: Planning is Non-Trivial!

- **Example 3.3.** The Sussman anomaly is a simple blocksworld planning problem:



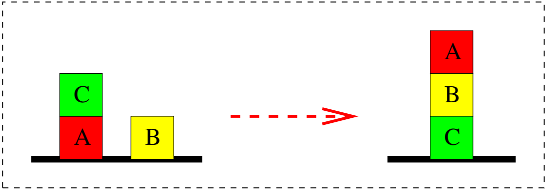
Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

- 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.



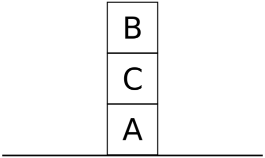
Planning History, p.s.: Planning is Non-Trivial!

- **Example 3.4.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ 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.



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/goals are as canonical as imaginable:
  - Preconditions, goals: 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 [FN71] was originally a planner (cf. Shakey), whose language actually wasn't quite that simple.

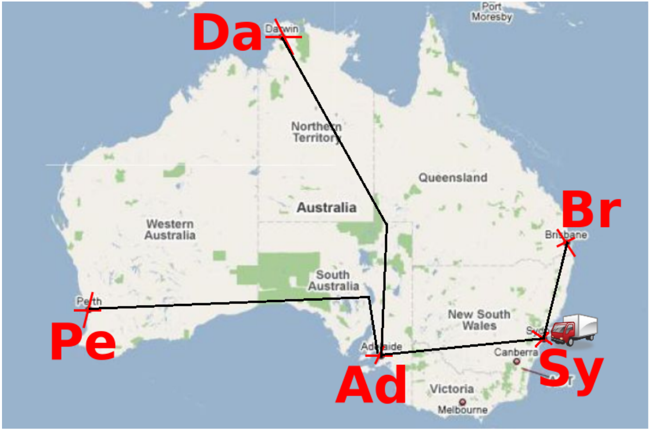
STRIPS Planning: Syntax

- **Definition 4.2.** A STRIPS task is a quadruple $\langle P, A, I, G\rangle$ where:
  - P is a finite set of facts: atomic proposition in PL^0 or $\mathrm{PL}^{\mathrm{nq}}$.
  - A is a finite set of actions; each $a \in A$ is a triple $a = \langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a\rangle$ of subsets of P referred to as the action's preconditions, add list, and delete list respectively; we require that $\mathrm{add}_a \cap \mathrm{del}_a = \emptyset$.
  - $I \subseteq P$ is the initial state.
  - $G \subseteq P$ is the goal state.

  We will often give each action $a \in A$ a name (a string), and identify a with that name.
- **Note:** We assume, for simplicity, that every action has cost 1. (Unit costs, cf.)

"TSP" in Australia

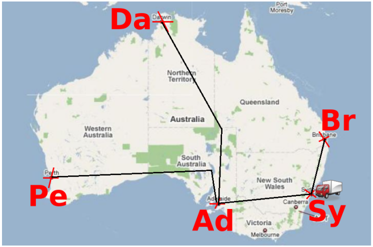
- **Example 4.3 (Salesman Travelling in Australia).**



Strictly speaking, this is not actually a **TSP** problem instance; simplified/adapted for illustration.

STRIPS Encoding of “TSP”

- **Example 4.4 (continuing).**



  - Facts P: $\{\mathrm{at}(x), \mathrm{vis}(x) | x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}, \mathrm{Pe}, \mathrm{Da}\}\}$.
  - Initial state I: $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
  - Goal state G:$\{\mathrm{at}(\mathrm{Sy})\} \cup \{\mathrm{vis}(x) | x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}, \mathrm{Pe}, \mathrm{Da}\}\}$.
  - 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: $\langle \mathrm{drv}(\mathrm{Sy}, \mathrm{Br}), \mathrm{drv}(\mathrm{Br}, \mathrm{Sy}), \mathrm{drv}(\mathrm{Sy}, \mathrm{Ad}), \mathrm{drv}(\mathrm{Ad}, \mathrm{Pe}), \mathrm{drv}(\mathrm{Pe}, \mathrm{Ad}), \ldots$
    $\ldots, \mathrm{drv}(\mathrm{Ad}, \mathrm{Da}), \mathrm{drv}(\mathrm{Da}, \mathrm{Ad}), \mathrm{drv}(\mathrm{Ad}, \mathrm{Sy}) \rangle$

STRIPS Planning: Semantics

- **Idea:** We define a plan for a STRIPS task Π as a solution to an induced search problem Θ_Π. (save work by reduction)
- **Definition 4.5.** Let $\Pi := \langle P, A, I, G\rangle$ be a STRIPS task. The search problem induced by Π is $\Theta_\Pi = \langle S_P, A, T, I, S_G\rangle$ where:
  - The states (also world state) $S_P := \mathcal{P}(P)$ are the subsets of P.
  - A is just Π's action. (so we can define plans easily)
  - The transition model T_A is $\{s \xrightarrow{a} \mathrm{apply}(s,a) | \mathrm{pre}_a \subseteq s\}$.
    If $\mathrm{pre}_a \subseteq s$, then $a \in A$ is applicable in s and $\mathrm{apply}(s,a) := (s \cup \mathrm{add}_a) \backslash \mathrm{del}_a$. If $\mathrm{pre}_a \not\subseteq s$, then $\mathrm{apply}(s,a)$ is undefined.
  - I is Π's initial state.
  - The goal states $S_G = \{s \in S_P | G \subseteq s\}$ are those that satisfy Π's goal state.

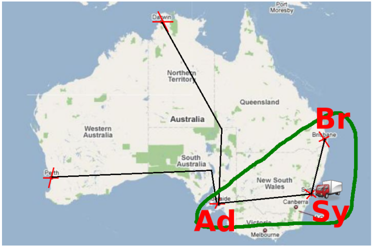
  An (optimal) plan for Π is an (optimal) solution for Θ_Π, i.e., a path from s to some $s' \in S_G$. Π is solvable if a plan for Π exists.
- **Definition 4.6.** For a plan $a = \langle a_1, \ldots, a_n\rangle$, we define

$$\mathrm{apply}(s,a) := \mathrm{apply}(\ldots \mathrm{apply}(\mathrm{apply}(s, a_1), a_2) \ldots, a_n)$$

  if each a_i is applicable in the respective state; else, $\mathrm{apply}(s,a)$ is undefined.

STRIPS Encoding of Simplified TSP

- **Example 4.7 (Simplified traveling salesman problem in Australia).**

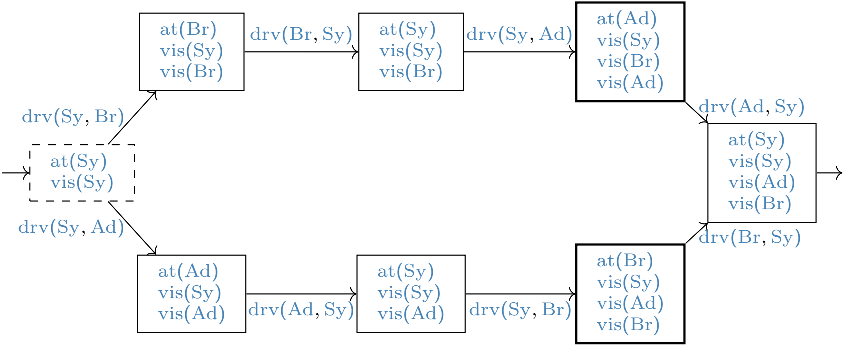


Let TSP_{-} be the STRIPS task, $\langle P, A, I, G\rangle$, where

- Facts P: $\{\mathrm{at}(x), \mathrm{vis}(x) | x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}\}\}$.
- Initial state state I: $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
- Goal state G: $\{\mathrm{vis}(x) | x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}\}\}$ (note: $\mathrm{noat}(\mathrm{Sy})$)
- Actions 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_

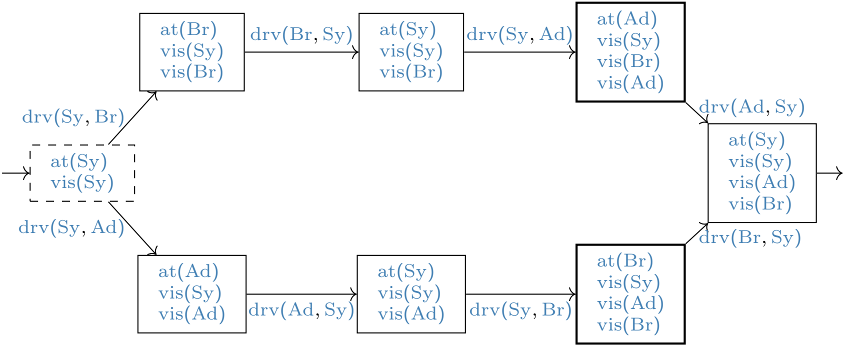
- The state space of the search problem $\Theta_{\mathrm{TSP\_}}$ induced by TSP_ from 4.7 is



- **Question:** Are there any plans for TSP_ in this graph?

Questionaire: State Space of $\mathrm{TSP}\_$

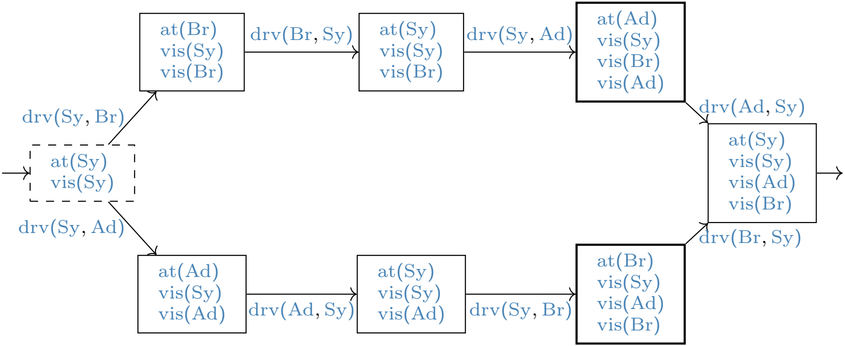
- The state space of the search problem $\Theta_{\mathrm{TSP}\_}$ induced by $\mathrm{TSP}\_$ from 4.7 is



- **Question:** Are there any plans for $\mathrm{TSP}\_$ in this graph?
- **Answer:** Yes, two – plans for $\mathrm{TSP}\_$ are solutions for $\Theta_{\mathrm{TSP}\_}$, dashed node $\widehat{=}$ I, thick nodes $\widehat{=}$ 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)

Questionaire: State Space of TSP_

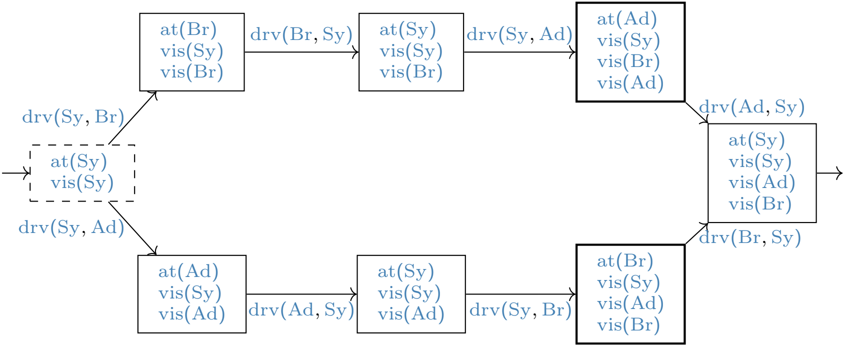
- The state space of the search problem $\Theta_{\mathrm{TSP\_}}$ induced by TSP_ from 4.7 is



- **Question:** Are there any plans for TSP_ in this graph?
- **Answer:** Yes, two – plans for TSP_ are solutions for $\Theta_{\mathrm{TSP\_}}$, dashed node $\widehat{=}$ I, thick nodes $\widehat{=}$ 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 ?

Questionaire: State Space of $\mathrm{TSP}\_$

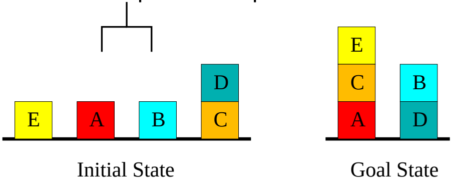
- The state space of the search problem $\Theta_{\mathrm{TSP}\_}$ induced by $\mathrm{TSP}\_$ from 4.7 is



- **Question:** Are there any plans for $\mathrm{TSP}\_$ in this graph?
- **Answer:** Yes, two – plans for $\mathrm{TSP}\_$ are solutions for $\Theta_{\mathrm{TSP}\_}$, dashed node $\widehat{=}$ I, thick nodes $\widehat{=}$ 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 ?
- **Answer:** No, only the part reachable from I. The state space of $\Theta_{\mathrm{TSP}\_}$ also includes e.g. the states $\{\mathrm{vis}(\mathrm{Sy})\}$ and $\{\mathrm{at}(\mathrm{Sy}),\mathrm{at}(\mathrm{Br})\}$.

The Blocksworld

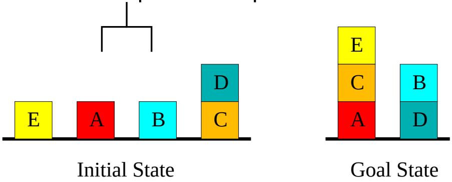
- **Definition 4.8.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.9.**



  - Facts: $\mathrm{on}(x, y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.

The Blocksworld

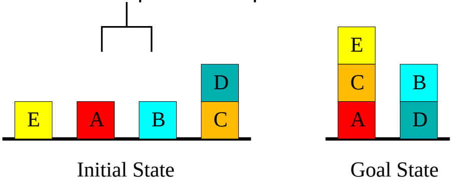
- **Definition 4.10.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.11.**



  - Facts: $\mathrm{on}(x, y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - initial state:
    $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D, C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.

The Blocksworld

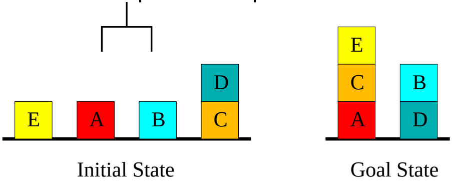
- **Definition 4.12.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.13.**



  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - initial state:
    $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D,C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.
  - Goal state: $\{\mathrm{on}(E,C), \mathrm{on}(C,A), \mathrm{on}(B,D)\}$.

The Blocksworld

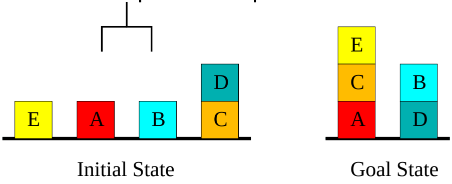
- **Definition 4.14.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.15.**



  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - initial state:
    $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D,C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.
  - Goal state: $\{\mathrm{on}(E,C), \mathrm{on}(C,A), \mathrm{on}(B,D)\}$.
  - Actions: $\mathrm{stack}(x,y)$, $\mathrm{unstack}(x,y)$, $\mathrm{putdown}(x)$, $\mathrm{pickup}(x)$.

The Blocksworld

- **Definition 4.16.** 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 4.17.**



  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - initial state:
    $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D,C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.
  - Goal state: $\{\mathrm{on}(E,C), \mathrm{on}(C,A), \mathrm{on}(B,D)\}$.
  - Actions: $\mathrm{stack}(x,y)$, $\mathrm{unstack}(x,y)$, $\mathrm{putdown}(x)$, $\mathrm{pickup}(x)$.
  - $\mathrm{stack}(x,y)$?
    $\mathrm{pre} : \{\mathrm{holding}(x), \mathrm{clear}(y)\}$
    $\mathrm{add} : \{\mathrm{on}(x,y), \mathrm{armEmpty}, \mathit{clear}x\}$
    $\mathrm{del} : \{\mathrm{holding}(x), \mathrm{clear}(y)\}$.

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

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 $\mathrm{armEmpty}$

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:**

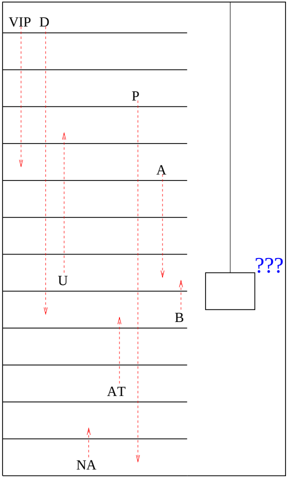
(A) No, must delete $\mathrm{armEmpty}$

(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.

Miconic-10: A Real-World Example

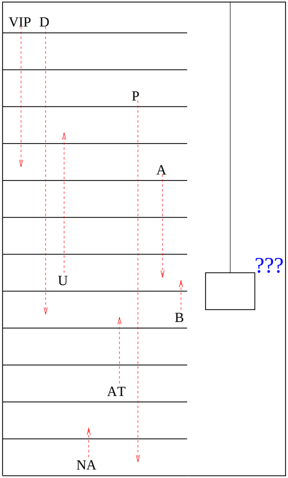
- **Example 4.18.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP:
- D:
- NA:
- AT: .
- A, B:
- P:

Miconic-10: A Real-World Example

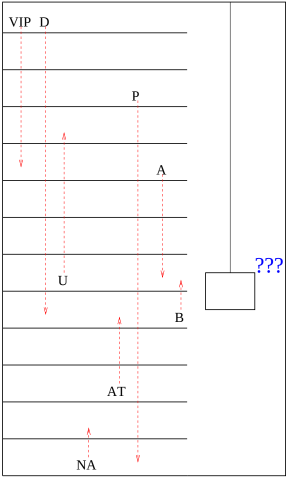
- **Example 4.19.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D:
- NA:
- AT: .
- A, B:
- P:

Miconic-10: A Real-World Example

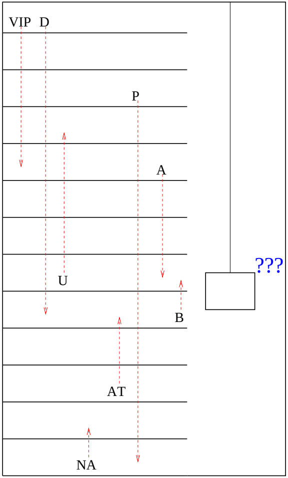
- **Example 4.20.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA:
- AT: .
- A, B:
- P:

Miconic-10: A Real-World Example

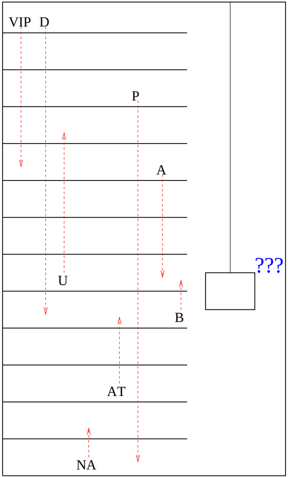
- **Example 4.21.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: .
- A, B:
- P:

Miconic-10: A Real-World Example

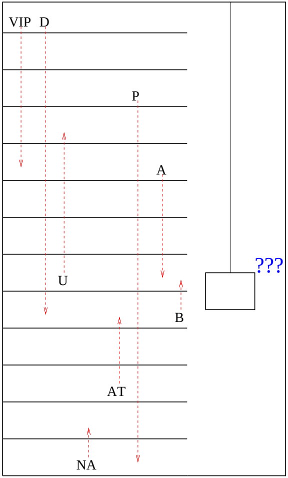
- **Example 4.22.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: Attendant.
- A, B:
- P:

Miconic-10: A Real-World Example

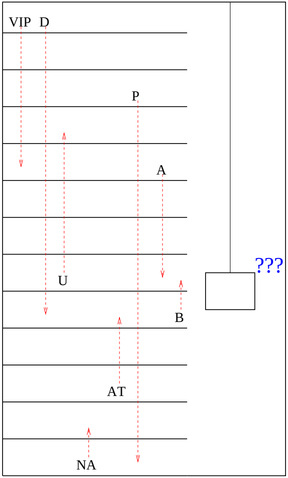
- **Example 4.23.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



  - 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:

Miconic-10: A Real-World Example

- **Example 4.24.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips

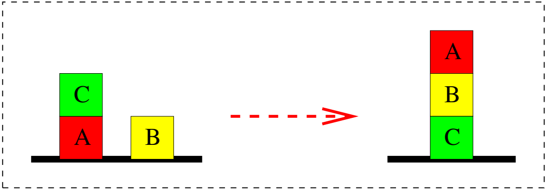


- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: Attendant.
- A, B: Never together in the same elevator
- P: Normal passenger

17.5 Partial Order Planning

Planning History, p.s.: Planning is Non-Trivial!

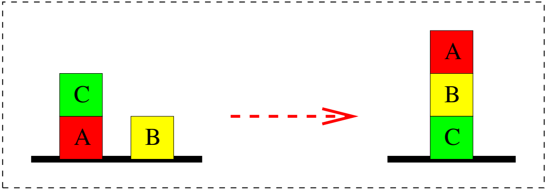
- **Example 5.1.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

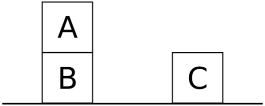
Planning History, p.s.: Planning is Non-Trivial!

- **Example 5.2.** The Sussman anomaly is a simple blocksworld planning problem:



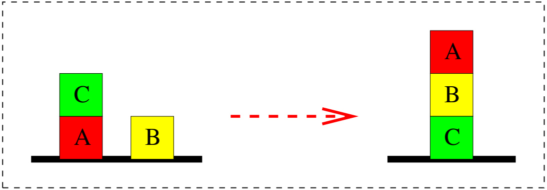
Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

- 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.



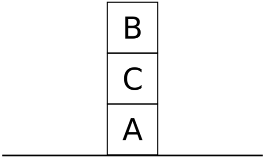
Planning History, p.s.: Planning is Non-Trivial!

- **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 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 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 actions (actions can occur multiply)
    - edges with propositions added by source and presupposed by target

    acyclicity of the graph induces a partial ordering on steps. q
  - additional temporal constraints resolve subgoal interactions and induce a linear order.

Partial Order Planning

- **Definition 5.6.** Any algorithm that can place two 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 actions (actions can occur multiply)
    - edges with propositions added by source and presupposed by target

    acyclicity of the graph induces a partial ordering on steps. q
  - 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.

Partially Ordered Plans

- **Definition 5.7.** Let $\langle P, A, I, G\rangle$ 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 actions from A, or are a
  - start step, which has label "effect" I, or a
  - finish step, which has label "precondition" G.

  Every edge $(S, T) \in E$ is either labeled by:
  - A non-empty set $p \subseteq P$ of facts that are effects of the 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.

A Notation for STRIPS Actions

- **Definition 5.12 (Notation).** In diagrams, we often write STRIPS actions into boxes with preconditions above and effects below.
- **Example 5.13.**
  - Actions: $Buy(x)$
  - Preconditions: $At(p)$, $Sells(p,x)$
  - Effects: $Have(x)$

$$At(p)\ Sells(p,x)$$
$$\boxed{\quad Buy(x)\quad}$$
$$Have(x)$$

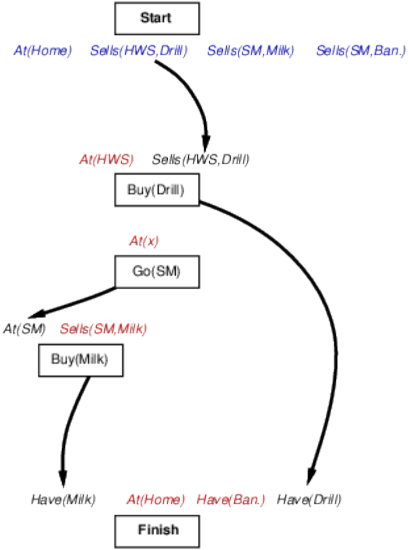
- **Notation:** A causal link $S \xrightarrow{p} T$ can also be denoted by a direct arrow between the effects p of S and the preconditions p of T in the STRIPS action notation above.
  Show temporal constraints as dashed arrows.

Planning Process

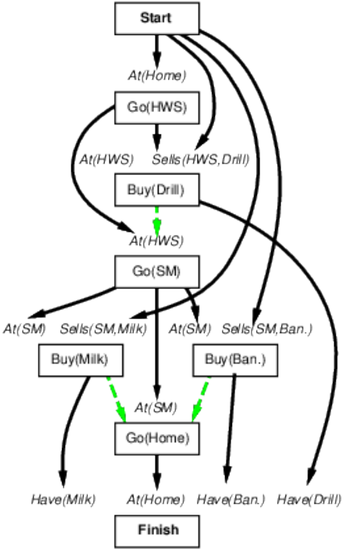
- **Definition 5.14.** Partial order planning is search in the space of partial plans via the following operations:
  - add link from an existing action to an open precondition,
  - add step (an action with links to other steps) to fulfil an open condition,
  - order one step wrt. another to remove possible 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.

Example: Shopping for Bananas, Milk, and a Cordless Drill

Example: Shopping for Bananas, Milk, and a Cordless Drill

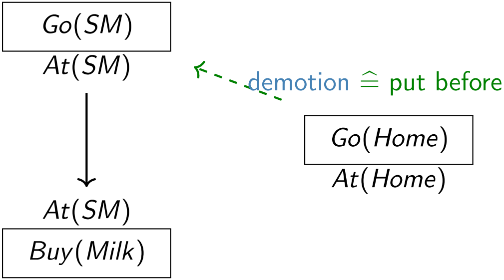


Example: Shopping for Bananas, Milk, and a Cordless Drill



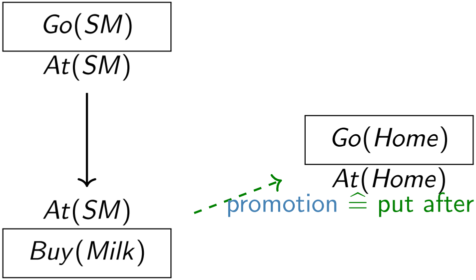
Clobbering and Promotion/Demotion

- **Definition 5.15.** 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 5.16.** 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 5.17.** $Go(Home)$ clobbers $At(Supermarket)$:



Clobbering and Promotion/Demotion

- **Definition 5.18.** 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 5.19.** 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 5.20.** $Go(Home)$ clobbers $At(Supermarket)$:



POP algorithm sketch

- **Definition 5.21.** 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
```

POP algorithm contd.

- **Definition 5.22.** The missing parts for the POP algorithm.

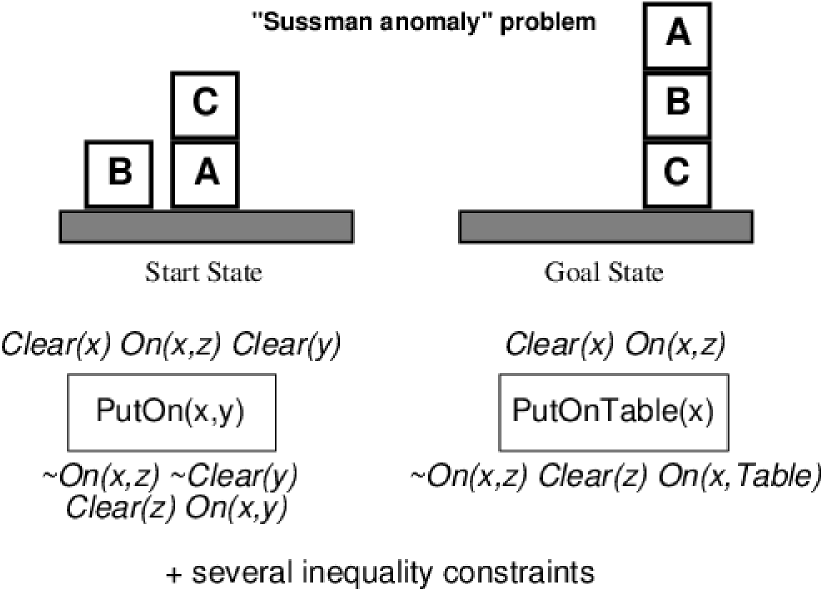
function Choose—Operator (plan, operators, S_{need}, c)
  choose a step S_{add} from operators or Steps(plan) that has c as an effect
  if there is no such step **then fail**
  add the ausal—link $S_{add} \xrightarrow{c} S_{need}$ **to** Links(plan)
  add the temporal—constraint $S_{add} \prec S_{need}$ **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**

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 5.23.** *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



Example: Solving the Sussman Anomaly (contd.)

- **Example 5.24.** Solving the Sussman anomaly

Start

$On(C,A)$ $On(A,T)$ $Cl(B)$ $On(B,T)$ $Cl(C)$

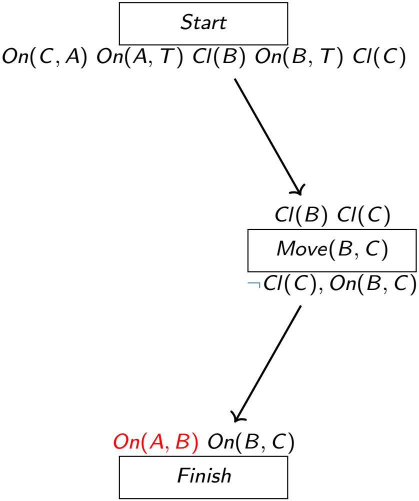
Initializing the partial order plan with with Start and Finish.

$On(A,B)$ $On(B,C)$

Finish

Example: Solving the Sussman Anomaly (contd.)

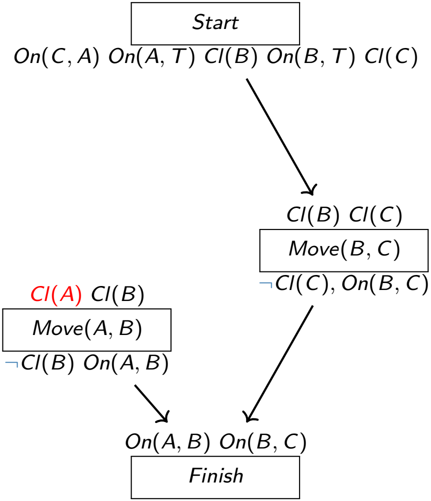
- **Example 5.25.** Solving the Sussman anomaly



Refining for the subgoal $On(B, C)$.

Example: Solving the Sussman Anomaly (contd.)

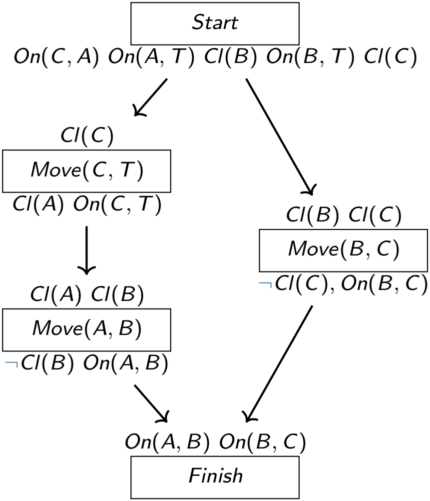
- **Example 5.26.** Solving the Sussman anomaly



Refining for the subgoal $ON(A, C)$.

Example: Solving the Sussman Anomaly (contd.)

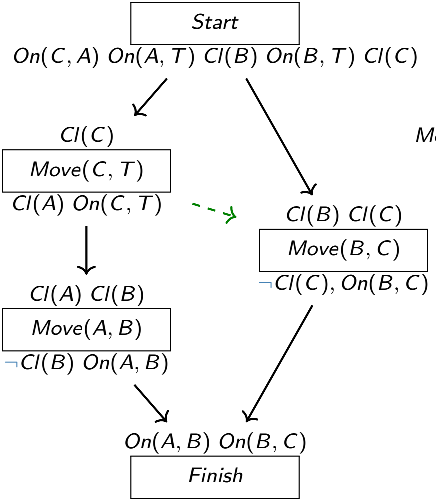
- **Example 5.27.** Solving the Sussman anomaly



Refining for the subgoal $Cl(A)$.

Example: Solving the Sussman Anomaly (contd.)

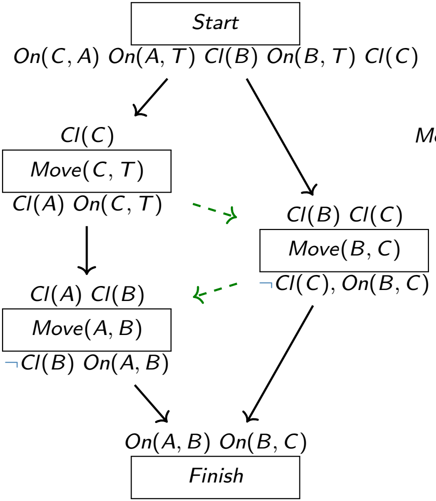
- **Example 5.28.** Solving the Sussman anomaly



$Move(A,B)$ clobbers $Cl(B)$ $\rightsquigarrow$ demote.

Example: Solving the Sussman Anomaly (contd.)

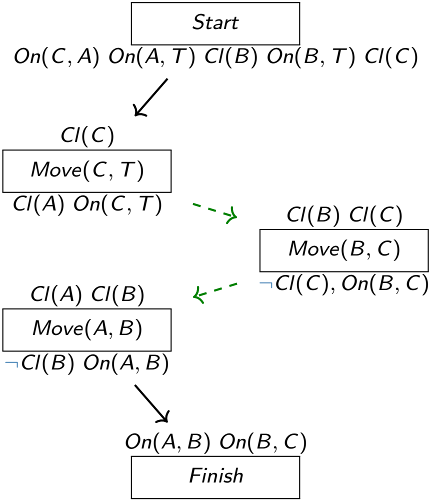
- **Example 5.29.** Solving the Sussman anomaly



$Move(B, C)$ clobbers $Cl(C) \leadsto$ demote.

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.30.** Solving the Sussman anomaly



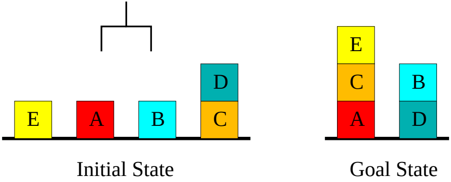
A totally ordered plan.

17.6 The PDDL Language

PDDL: Planning Domain Description Language

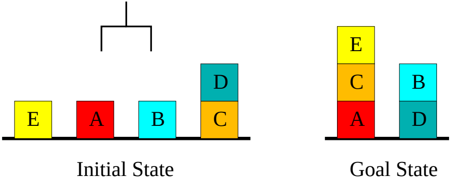
- **Definition 6.1.** The Planning Domain Description Language (PDDL) is a standardized representation language for planning benchmarks in various extensions of the STRIPS formalism.
- **Definition 6.2.** PDDL is not a propositional language
  - Representation is lifted, using object variables to be 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 actions.

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))))
  ...)
```

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)))))
)
```

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.

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).

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".)

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 . . .)

17.7 Conclusion

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 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.

Chapter 18
Planning II: Algorithms

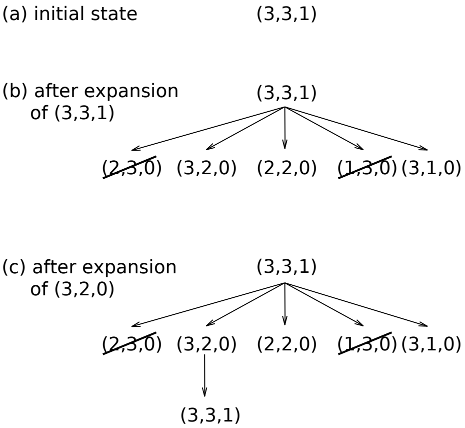
18.1 Introduction

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.

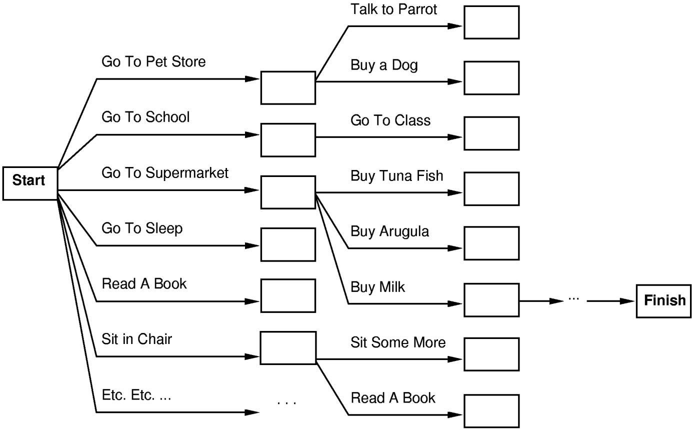
Reminder: Search

- Starting at initial state, produce all successor states step by step:



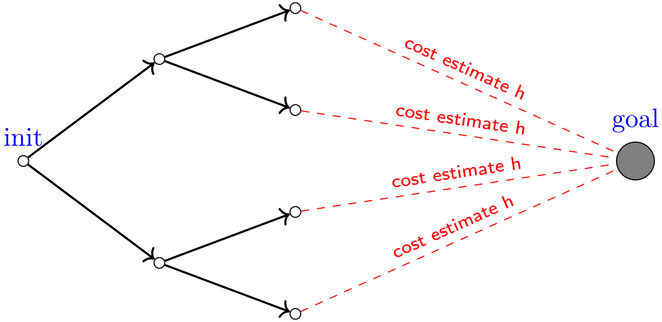
In planning, this is referred to as forward search, or forward state-space search.

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/`

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 function 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 $h^*(s)$.

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^+.

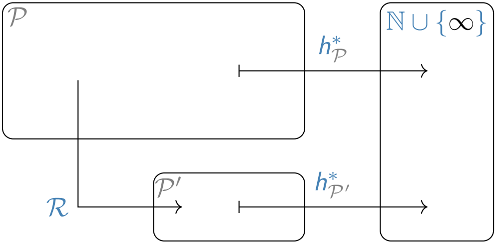
18.2 How to Relax in Planning

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).**



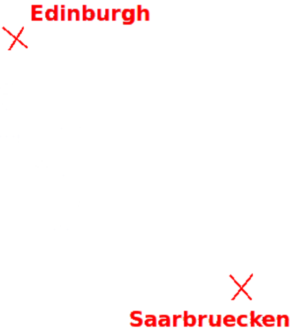
1. You have a class $\mathcal{P}$ of problems, whose perfect heuristic $h^*_{\mathcal{P}}$ you wish to estimate.
2. You define a class $\mathcal{P}'$ of *simpler problems*, whose perfect heuristic $h^*_{\mathcal{P}'}$ can be used to estimate $h^*_{\mathcal{P}}$.
3. You define a transformation – the relaxation 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.

Reminder: Heuristic Functions from Relaxed Problems

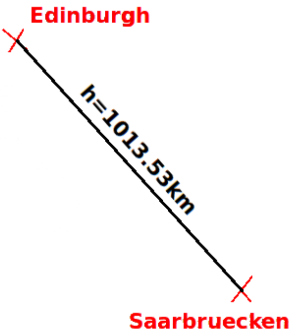
- Problem Π: Find a route from Saarbrücken to Edinburgh.

Reminder: Heuristic Functions from Relaxed Problems



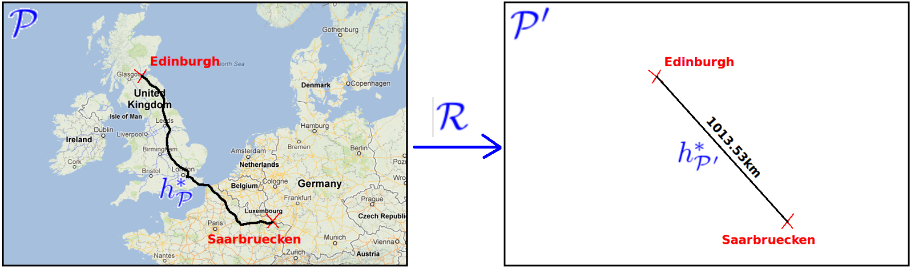
- Relaxed Problem Π': Throw away the map.

Reminder: Heuristic Functions from Relaxed Problems



- Heuristic function h: Straight line distance.

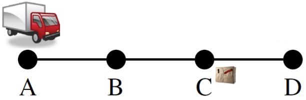
Relaxation in Route-Finding



- **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.

How to Relax in Planning? (A Reminder!)

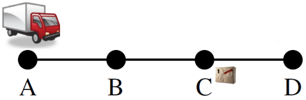
- **Example 2.5 (Logistics).**



- facts P: $\{\mathrm{truck}(x)|x{\in}\{A, B, C, D\}\} \cup \{\mathrm{pack}(x)|x{\in}\{A, B, C, D, T\}\}$.
- initial state I: $\{\mathrm{truck}(A), \mathrm{pack}(C)\}$.
- goal state G: $\{\mathrm{truck}(A), \mathrm{pack}(D)\}$.
- actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
  - $\mathrm{drive}(x, y)$, where x and y have a road: "$\mathrm{truck}(x) \Rightarrow \mathrm{truck}(y), \neg\mathrm{truck}(x)$".
  - $\mathrm{load}(x)$: "$\mathrm{truck}(x), \mathrm{pack}(x) \Rightarrow \mathrm{pack}(T), \neg\mathrm{pack}(x)$".
  - $\mathrm{unload}(x)$: "$\mathrm{truck}(x), \mathrm{pack}(T) \Rightarrow \mathrm{pack}(x), \neg\mathrm{pack}(T)$".

How to Relax in Planning? (A Reminder!)

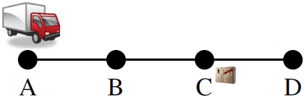
- **Example 2.7 (Logistics).**



  - facts P: $\{\mathrm{truck}(x) | x \in \{A, B, C, D\}\} \cup \{\mathrm{pack}(x) | x \in \{A, B, C, D, T\}\}$.
  - initial state I: $\{\mathrm{truck}(A), \mathrm{pack}(C)\}$.
  - goal state G: $\{\mathrm{truck}(A), \mathrm{pack}(D)\}$.
  - 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 2.8 ("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?

How to Relax in Planning? (A Reminder!)

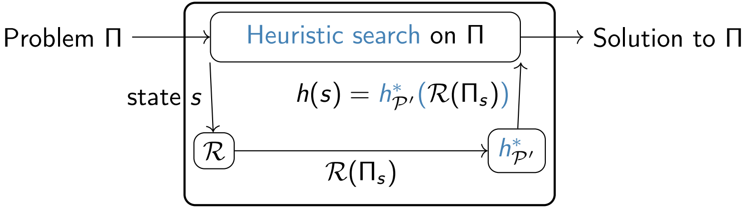
- **Example 2.9 (Logistics).**



  - facts P: $\{\mathrm{truck}(x)|x\in\{A,B,C,D\}\}\cup\{\mathrm{pack}(x)|x\in\{A,B,C,D,T\}\}$.
  - initial state I: $\{\mathrm{truck}(A),\mathrm{pack}(C)\}$.
  - goal state G: $\{\mathrm{truck}(A),\mathrm{pack}(D)\}$.
  - 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 $\langle\mathrm{unload}(D)\rangle$.

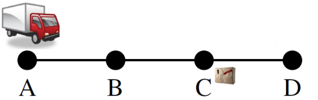
How to Relax During Search: Overview

- **Attention:** Search uses the real (un-relaxed) Π. The relaxation is applied (e.g., in Only-Adds, the simplified actions are used) **only within the call to $h(s)$!!!**



  - Here, Π_s is Π with initial state replaced by s, i.e., $\Pi{:=}\langle P, A, I, G\rangle$ changed to $\Pi^s{:=}\langle P, A, \{s\}, G\rangle$: The task of finding a plan for search state s.
  - A common student mistake 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.

How to Relax During Search: Only-Adds



Real problem:

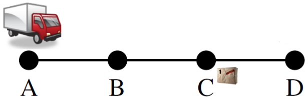
- Initial state I: AC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $drXY$, loX, ulX.

Greedy best-first search: (tie-breaking: alphabetic)

We are here

AC

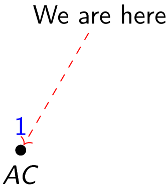
How to Relax During Search: Only-Adds

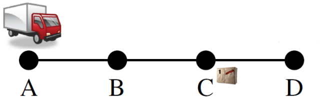


Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)





Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 1$: $\langle ulD \rangle$.

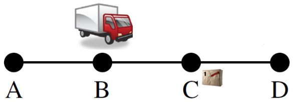
Greedy best-first search: (tie-breaking: alphabetic)

We are here

1

AC

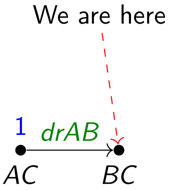
How to Relax During Search: Only-Adds



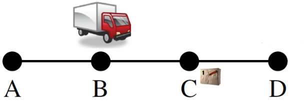
Real problem:

- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Only-Adds



Relaxed problem:

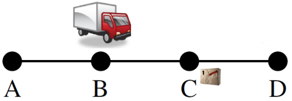
- State s: BC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)

We are here

$$\overset{1}{AC} \xrightarrow{drAB} BC$$

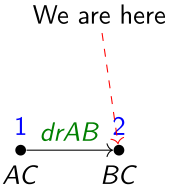
How to Relax During Search: Only-Adds



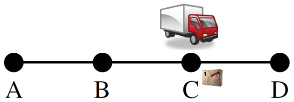
Relaxed problem:

- State s: BC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

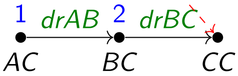


Real problem:

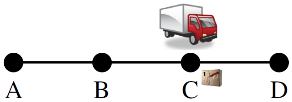
- State s: CC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here



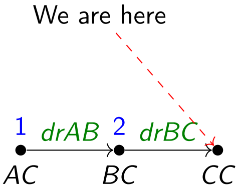
How to Relax During Search: Only-Adds



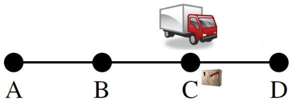
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



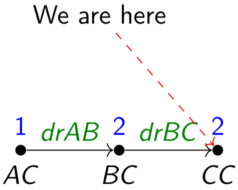
How to Relax During Search: Only-Adds



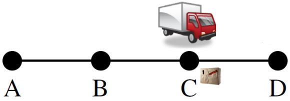
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



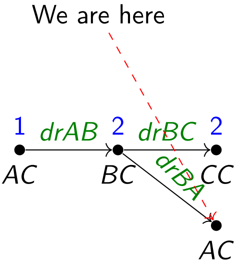
How to Relax During Search: Only-Adds



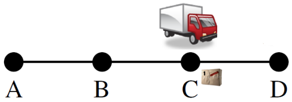
Real problem:

- State s: AC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $BC \xrightarrow{drBA} AC$.

Greedy best-first search: (tie-breaking: alphabetic)



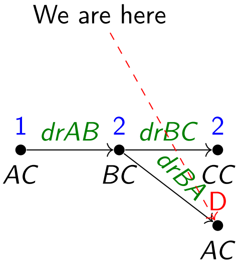
How to Relax During Search: Only-Adds



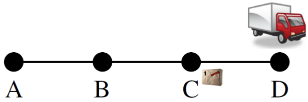
Real problem:

- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



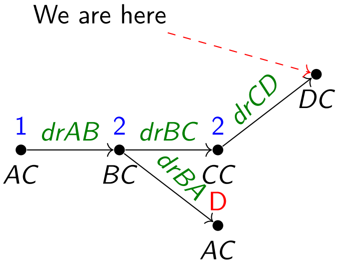
How to Relax During Search: Only-Adds



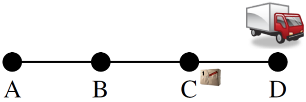
Real problem:

- State s: DC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search: (tie-breaking: alphabetic)



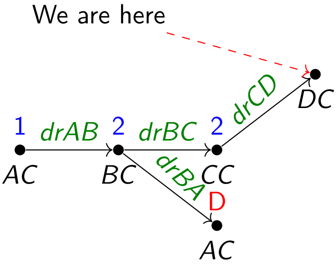
How to Relax During Search: Only-Adds



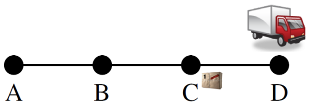
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



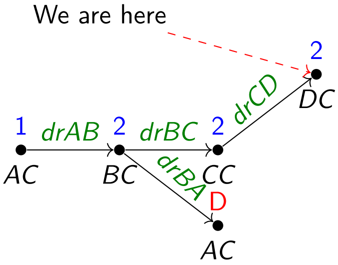
How to Relax During Search: Only-Adds



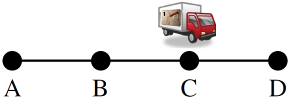
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

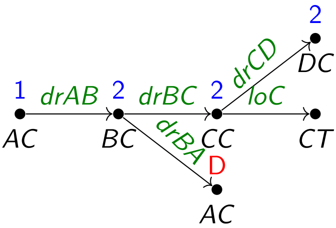


Real problem:

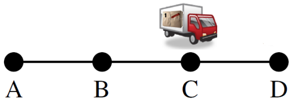
- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here



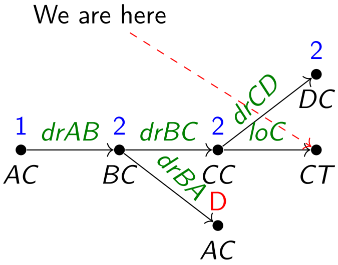
How to Relax During Search: Only-Adds



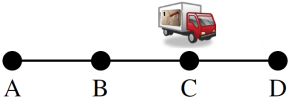
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



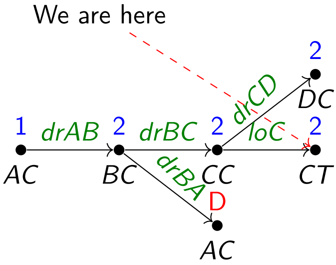
How to Relax During Search: Only-Adds



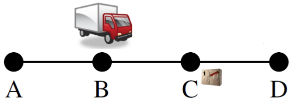
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



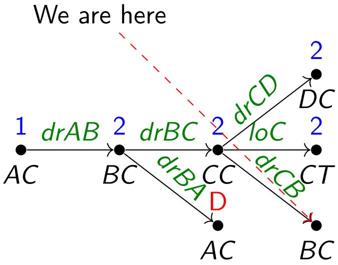
How to Relax During Search: Only-Adds



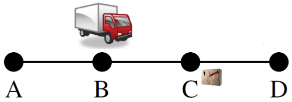
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{drCB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



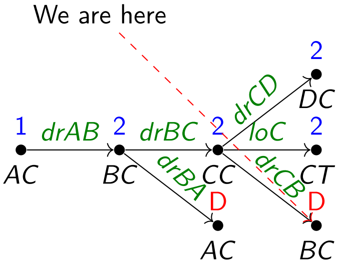
How to Relax During Search: Only-Adds



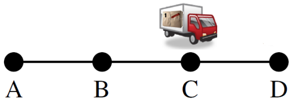
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



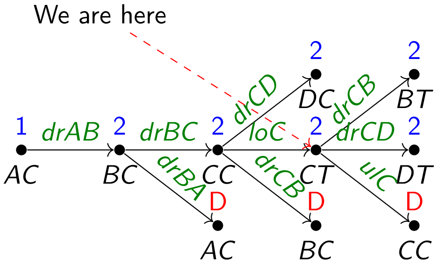
How to Relax During Search: Only-Adds



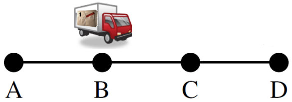
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- Successors: BT, DT, CC.

Greedy best-first search: (tie-breaking: alphabetic)



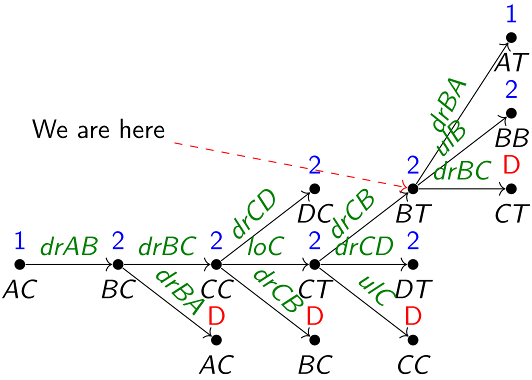
How to Relax During Search: Only-Adds



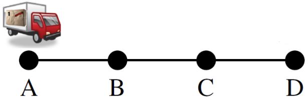
Real problem:

- State s: BT; goal G: AD.
- Actions A: $\text{pre}, \text{add}, \text{del}$.
- Successors: AT, BB, CT.

Greedy best-first search: (tie-breaking: alphabetic)



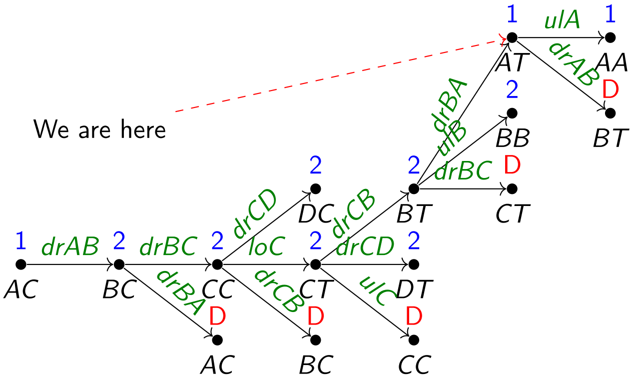
How to Relax During Search: Only-Adds



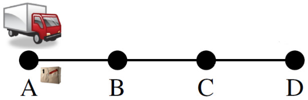
Real problem:

- State s: AT; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Successors: AA, BT.

Greedy best-first search: (tie-breaking: alphabetic)



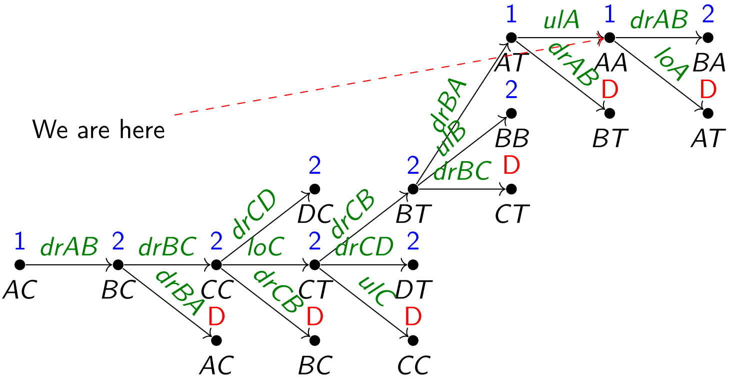
How to Relax During Search: Only-Adds



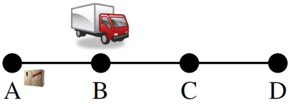
Real problem:

- State s: AA; goal G: AD.
- Actions A: pre, add, del.
- Successors: BA, AT.

Greedy best-first search: (tie-breaking: alphabetic)



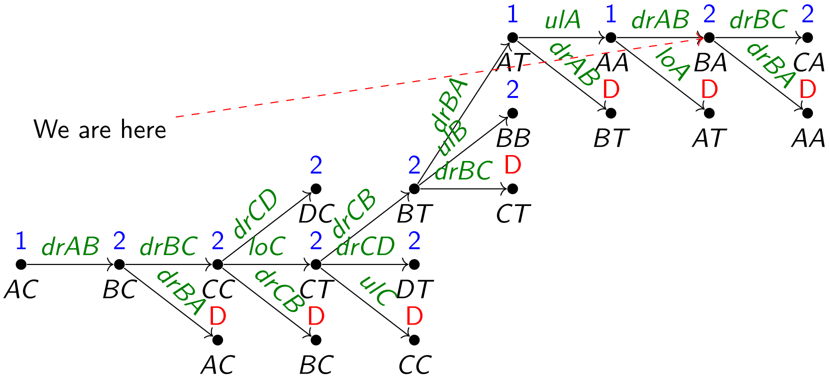
How to Relax During Search: Only-Adds



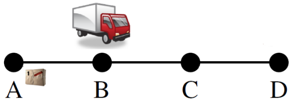
Real problem:

- State s: BA; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Successors: CA, AA.

Greedy best-first search: (tie-breaking: alphabetic)



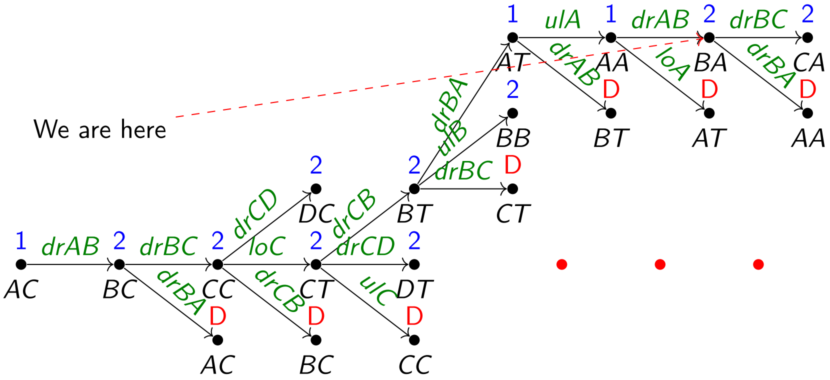
How to Relax During Search: Only-Adds



Real problem:

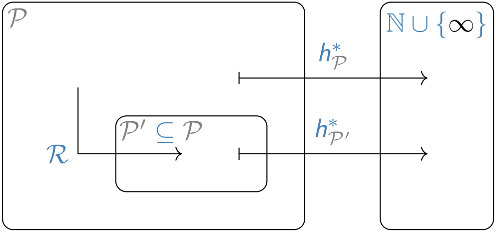
- State s: BA; goal G: AD.
- Actions A: pre, add, del.
- Successors: CA, AA.

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.

18.3 The Delete Relaxation

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)

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world: (before)

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world: (after)

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world: (before)

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world: (after)

How the Delete Relaxation Changes the World (III)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world:

How the Delete Relaxation Changes the World (III)

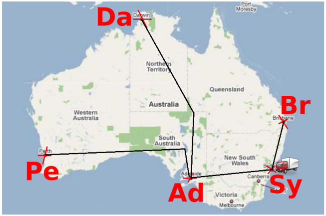
- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world:

The Delete Relaxation

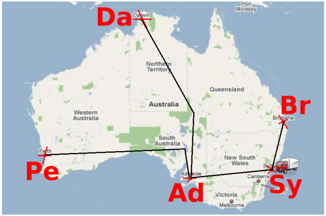
- **Definition 3.1 (Delete Relaxation).** Let $\Pi{:=}\langle P, A, I, G\rangle$ be a STRIPS task. The delete relaxation of Π is the task $\Pi^+ = \langle P, A^+, I, G\rangle$ where $A^+{:=}\{a^+ | a{\in}A\}$ with $\mathrm{pre}_{a^+}{:=}\mathrm{pre}_a$, $\mathrm{add}_{a^+}{:=}\mathrm{add}_a$, and $\mathrm{del}_{a^+}{:=}\emptyset$.
- 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 $\Pi{:=}\langle P, A, I, G\rangle$ be a STRIPS task, and let s be a state. A relaxed plan for s is a plan for $\langle P, A, s, G\rangle^+$. A relaxed plan for I is called a relaxed plan for Π.
- A relaxed plan for s is an action sequence that solves s when pretending that all delete lists are empty.
- Also called delete-relaxed plan: "relaxation" is often used to mean delete relaxation by default.

A Relaxed Plan for "TSP" in Australia



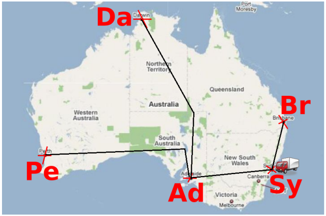
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for "TSP" in Australia



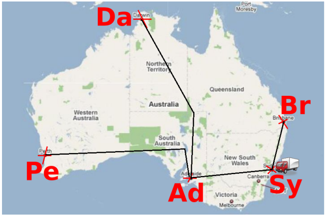
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^+$: $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for "TSP" in Australia



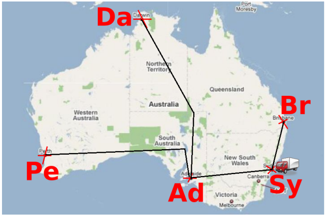
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^{+}$: $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
3. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Ad})^{+}$: $\{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for “TSP” in Australia



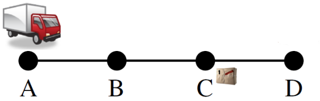
1. **Initial state:** $\{\mathrm{at(Sy)}, \mathrm{vis(Sy)}\}$.
2. $\mathrm{drv(Sy, Br)}^+$**:** $\{\mathrm{at(Br)}, \mathrm{vis(Br)}, \mathrm{at(Sy)}, \mathrm{vis(Sy)}\}$.
3. $\mathrm{drv(Sy, Ad)}^+$**:** $\{\mathrm{at(Ad)}, \mathrm{vis(Ad)}, \mathrm{at(Br)}, \mathrm{vis(Br)}, \mathrm{at(Sy)}, \mathrm{vis(Sy)}\}$.
4. $\mathrm{drv(Ad, Pe)}^+$**:**
   $\{\mathrm{at(Pe)}, \mathrm{vis(Pe)}, \mathrm{at(Ad)}, \mathrm{vis(Ad)}, \mathrm{at(Br)}, \mathrm{vis(Br)}, \mathrm{at(Sy)}, \mathrm{vis(Sy)}\}$.

A Relaxed Plan for "TSP" in Australia



1. **Initial state:** $\{at(Sy), vis(Sy)\}$.
2. $drv(Sy, Br)^+$: $\{at(Br), vis(Br), at(Sy), vis(Sy)\}$.
3. $drv(Sy, Ad)^+$: $\{at(Ad), vis(Ad), at(Br), vis(Br), at(Sy), vis(Sy)\}$.
4. $drv(Ad, Pe)^+$: $\{at(Pe), vis(Pe), at(Ad), vis(Ad), at(Br), vis(Br), at(Sy), vis(Sy)\}$.
5. $drv(Ad, Da)^+$: $\{at(Da), vis(Da), at(Pe), vis(Pe), at(Ad), vis(Ad), at(Br), vis(Br), at(Sy), vis(Sy)\}$.

A Relaxed Plan for "Logistics"



- Facts P: $\{\mathrm{truck}(x) | x \in \{A, B, C, D\}\} \cup \{\mathrm{pack}(x) | x \in \{A, B, C, D, T\}\}$.
- Initial state I: $\{\mathrm{truck}(A), \mathrm{pack}(C)\}$.
- Goal G: $\{\mathrm{truck}(A), \mathrm{pack}(D)\}$.
- Relaxed 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:

$$\langle \mathrm{drive}(A, B)^+, \mathrm{drive}(B, C)^+, \mathrm{load}(C)^+, \mathrm{drive}(C, D)^+, \mathrm{unload}(D)^+ \rangle$$

- We don't need to drive the truck back, because "it is still at A".

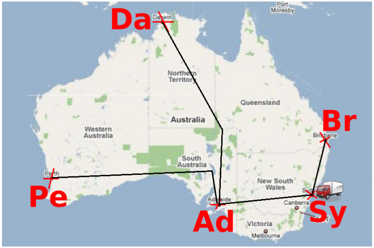
PlanEx$^+$

- **Definition 3.3 (Relaxed Plan Existence Problem).** By PlanEx$^+$, we denote the problem of deciding, given a STRIPS task $\Pi := \langle P, A, I, G \rangle$, 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 ⊈ F do
         F' := F ∪ ⋃_{a∈A: pre_a ⊆ F} add_a
         if F' = F then return "unsolvable" endif        (*)
         F := F'
     endwhile
     return "solvable"
     ```

  2. The algorithm terminates after at most $|P|$ iterations, and thus runs in polynomial time.
  3. Correctness: See slide 624

Deciding PlanEx^{+} in "TSP" in Australia

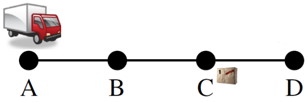


Iterations on F:

1. $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$
2. $\cup\ \{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br})\}$
3. $\cup\ \{\mathrm{at}(\mathrm{Da}), \mathrm{vis}(\mathrm{Da}), \mathrm{at}(\mathrm{Pe}), \mathrm{vis}(\mathrm{Pe})\}$

Deciding PlanEx$^+$ in "Logistics"

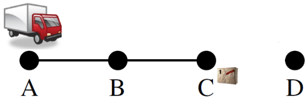
- **Example 3.4 (The solvable Case).**



Iterations on F:

1. $\{\mathrm{truck}(A), \mathrm{pack}(C)\}$
2. $\cup\{\mathrm{truck}(B)\}$
3. $\cup\{\mathrm{truck}(C)\}$
4. $\cup\{\mathrm{truck}(D), \mathrm{pack}(T)\}$
5. $\cup\{\mathrm{pack}(A), \mathrm{pack}(B), \mathrm{pack}(D)\}$

- **Example 3.5 (The unsolvable Case).**



Iterations on F:

1. $\{\mathrm{truck}(A), \mathrm{pack}(C)\}$
2. $\cup\{\mathrm{truck}(B)\}$
3. $\cup\{\mathrm{truck}(C)\}$
4. $\cup\{\mathrm{pack}(T)\}$
5. $\cup\{\mathrm{pack}(A), \mathrm{pack}(B)\}$
6. $\cup\emptyset$

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 $\mathrm{apply}(I, A_0^+)$, and so forth.
3. Thus $F_i = \mathrm{apply}(I, \langle A_0^+, \ldots, A_{i-1}^+ \rangle)$. (Within each A_j^+, we can sequence the actions in any order.)
4. Direction "$\Rightarrow$" *If "solvable" is returned after iteration n then $G \subseteq F_n = \mathrm{apply}(I, \langle A_0^+, \ldots, A_{n-1}^+ \rangle)$ so $\langle A_0^+, \ldots, A_{n-1}^+ \rangle$ can be sequenced to a relaxed plan which shows the claim.*
5. Direction "$\Leftarrow$"

5.1. Let $\langle a_0^+, \ldots, a_{n-1}^+ \rangle$ be a relaxed plan, hence $G \subseteq \mathrm{apply}(I, \langle a_0^+, \ldots, a_{n-1}^+ \rangle)$.

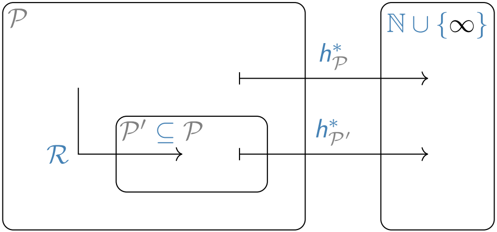
5.2. Assume, for the moment, that we drop line (*) from the algorithm. It is then easy to see that $a_i \in A_i$ and $\mathrm{apply}(I, \langle a_0^+, \ldots, a_{i-1}^+ \rangle) \subseteq F_i$, for all i.

5.3. We get $G \subseteq \mathrm{apply}(I, \langle a_0^+, \ldots, a_{n-1}^+ \rangle) \subseteq F_n$, and the algorithm returns "solvable" as desired.

5.4. 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.

18.4 The h^+Heuristic

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* $h^* \circ \mathcal{R}$.

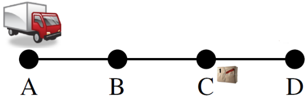
h^+: The Ideal Delete Relaxation Heuristic

- **Definition 4.1 (Optimal Relaxed Plan).** Let $\langle P, A, I, G\rangle$ be a STRIPS task, and let s be a state. A optimal relaxed plan for s is an optimal plan for $\langle P, A, \{s\}, G\rangle^+$.
- Same as slide 618, just adding the word "optimal".
- Here's what we're looking for:
- **Definition 4.2.** Let $\Pi{:=}\langle P, A, I, G\rangle$ be a STRIPS task with states S. The ideal delete relaxation heuristic h^+ for Π is the function $h^+ : S{\rightarrow}\mathbb{N} \cup \{\infty\}$ 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.

h^+ is Admissible

- **Lemma 4.3.** *Let $\Pi:=\langle P, A, I, G\rangle$ be a STRIPS task, and let s be a state. If $\langle a_1, \ldots, a_n\rangle$ is a plan for $\Pi_s:=\langle P, A, \{s\}, G\rangle$, then $\langle a_1^+, \ldots, a_n^+\rangle$ is a plan for Π^+.*
- *Proof sketch:* Show by induction over $0 \leq i \leq n$ that $\mathrm{apply}(s, \langle a_1, \ldots, a_i\rangle) \subseteq \mathrm{apply}(s, \langle a_1^+, \ldots, a_i^+\rangle)$.
- If we ignore deletes, the states along the plan can only get bigger.
- **Theorem 4.4.** *h^+ is Admissible.*
- *Proof:*
  1. Let $\Pi:=\langle P, A, I, G\rangle$ be a STRIPS task with states P, and let $s \in P$.
  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.

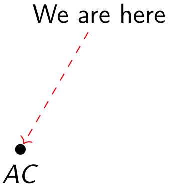
How to Relax During Search: Ignoring Deletes



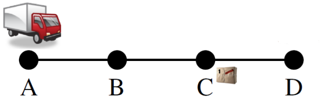
Real problem:

- Initial state I: AC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $drXY, loX, ulX$.

Greedy best-first search: (tie-breaking: alphabetic)



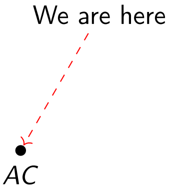
How to Relax During Search: Ignoring Deletes



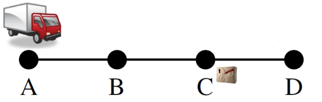
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



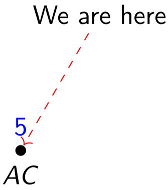
How to Relax During Search: Ignoring Deletes



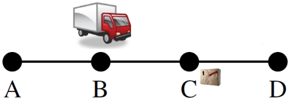
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) = 5$: e.g. $\langle drAB, drBC, drCD, loC, ulD \rangle$.
  (tie-breaking: alphabetic)

Greedy best-first search:



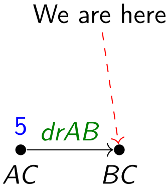
How to Relax During Search: Ignoring Deletes



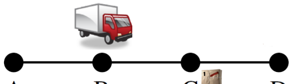
Real problem:

- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes



Relaxed problem:

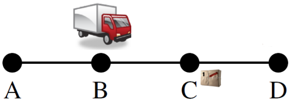
- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)

We are here

$$\overset{5}{AC} \xrightarrow{drAB} BC$$

How to Relax During Search: Ignoring Deletes



Relaxed problem:

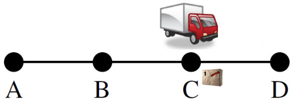
- State s: BC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drBA, drBC, drCD, loC, ulD \rangle$.
  (tie-breaking: alphabetic)

Greedy best-first search:

We are here

$$\underset{AC}{5} \xrightarrow{drAB} \underset{BC}{5}$$

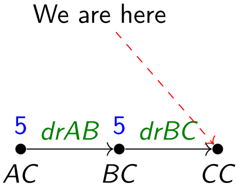
How to Relax During Search: Ignoring Deletes



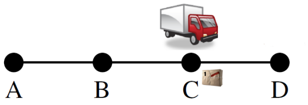
Real problem:

- State s: CC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes



Relaxed problem:

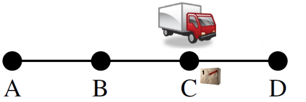
- State s: CC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)

$$\underset{AC}{\overset{5}{\bullet}} \xrightarrow{drAB} \underset{BC}{\overset{5}{\bullet}} \xrightarrow{drBC} \underset{CC}{\bullet}$$

We are here

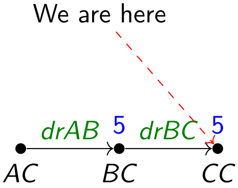
How to Relax During Search: Ignoring Deletes



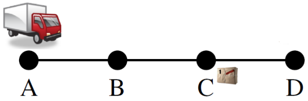
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) = 5$: e.g. $\langle drCB, drBA, drCD, loC, ulD \rangle$.
  (tie-breaking: alphabetic)

Greedy best-first search:



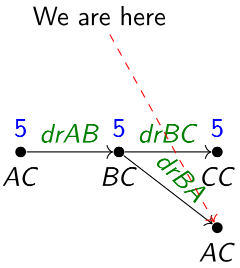
How to Relax During Search: Ignoring Deletes



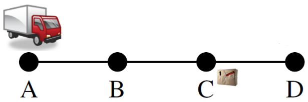
Real problem:

- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBA} AC$.

Greedy best-first search: (tie-breaking: alphabetic)



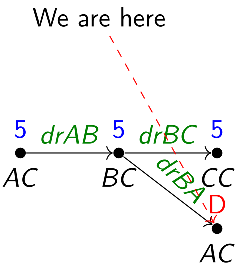
How to Relax During Search: Ignoring Deletes



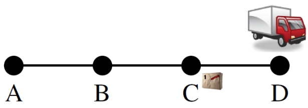
Real problem:

- State s: AC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



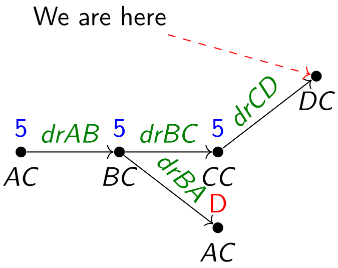
How to Relax During Search: Ignoring Deletes



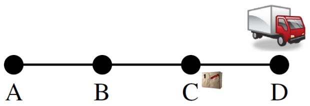
Real problem:

- State s: DC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search: (tie-breaking: alphabetic)



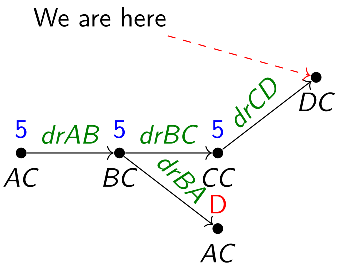
How to Relax During Search: Ignoring Deletes



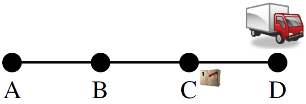
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

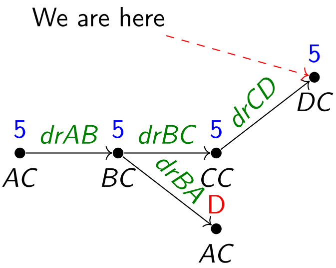


Relaxed problem:

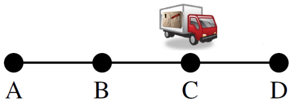
- State s: DC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s)$ =5: e.g. $\langle drDC, drCB, drBA, loC, ulD \rangle$.

(tie-breaking: alphabetic)

Greedy best-first search:



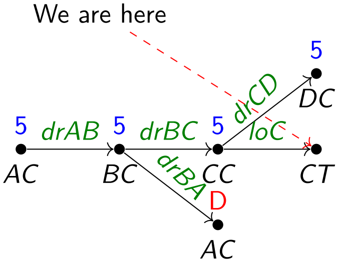
How to Relax During Search: Ignoring Deletes



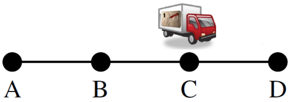
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search: (tie-breaking: alphabetic)



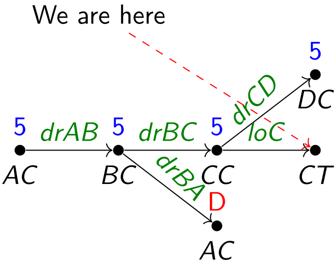
How to Relax During Search: Ignoring Deletes



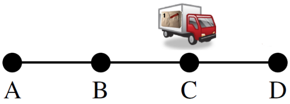
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

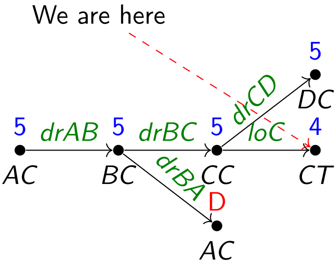


Relaxed problem:

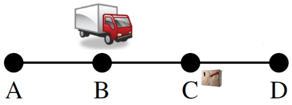
- State s: CT; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^{+}(s) = 4$: e.g. $\langle drCB, drBA, drCD, ulD \rangle$.

(tie-breaking: alphabetic)

Greedy best-first search:



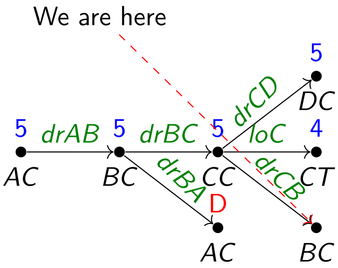
How to Relax During Search: Ignoring Deletes



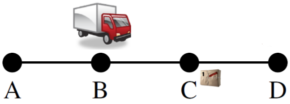
Real problem:

- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{drCB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



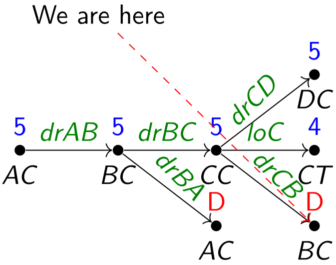
How to Relax During Search: Ignoring Deletes



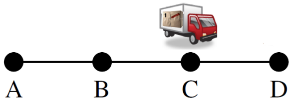
Real problem:

- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



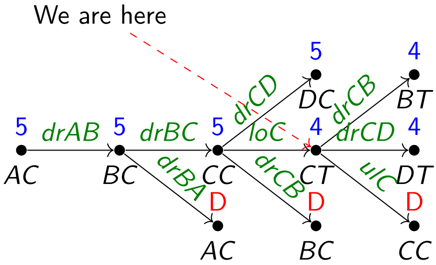
How to Relax During Search: Ignoring Deletes



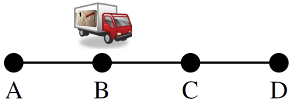
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- Successors: BT, DT, CC.

Greedy best-first search: (tie-breaking: alphabetic)



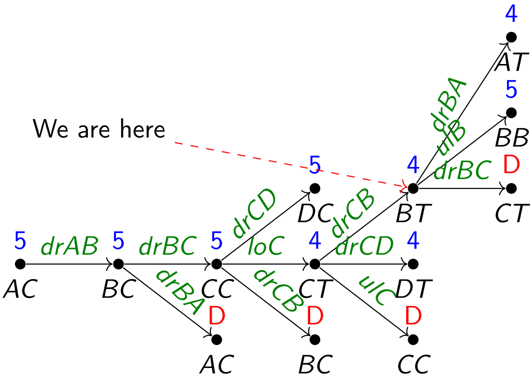
How to Relax During Search: Ignoring Deletes



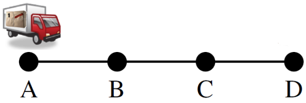
Real problem:

- State s: BT; goal G: AD.
- Actions A: pre, add, del.
- Successors: AT, BB, CT.

Greedy best-first search: (tie-breaking: alphabetic)



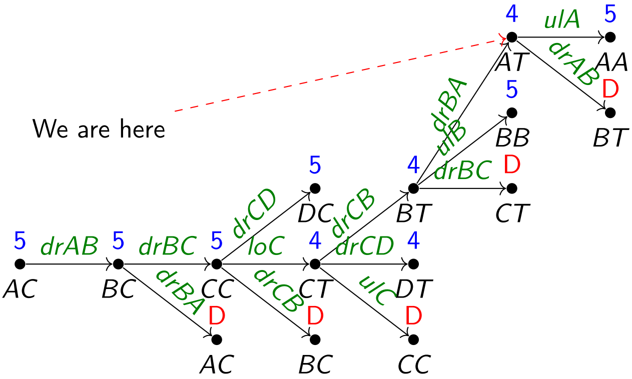
How to Relax During Search: Ignoring Deletes

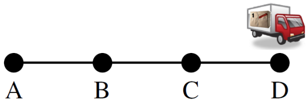


Real problem:

- State s: AT; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Successors: AA, BT.

Greedy best-first search: (tie-breaking: alphabetic)

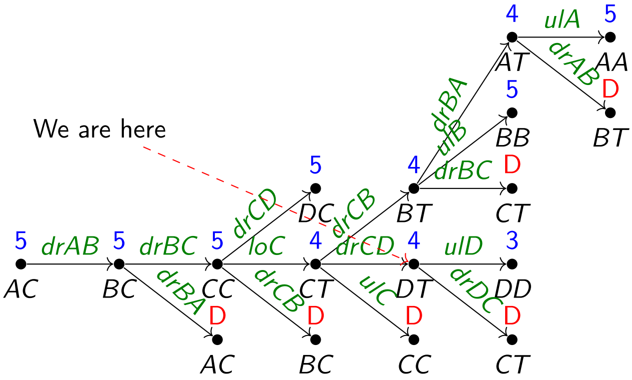




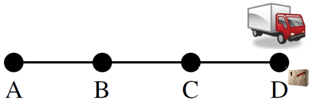
Real problem:

- State s: DT; goal G: AD.
- Actions A: pre, add, del.
- Successors: DD, CT.

Greedy best-first search: (tie-breaking: alphabetic)



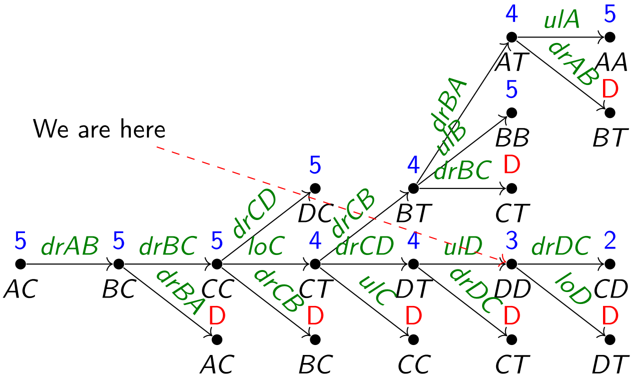
How to Relax During Search: Ignoring Deletes



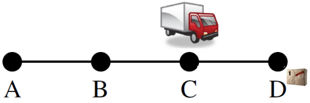
Real problem:

- State s: DD; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Successors: CD, DT.

Greedy best-first search: (tie-breaking: alphabetic)



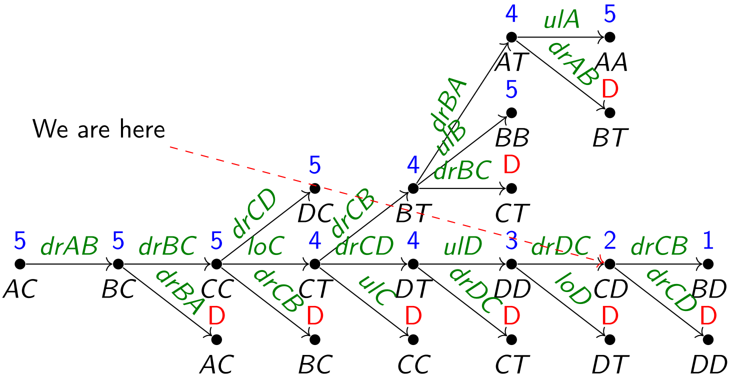
How to Relax During Search: Ignoring Deletes



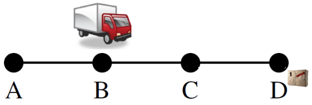
Real problem:

- State s: CD; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Successors: BD, DD.

Greedy best-first search: (tie-breaking: alphabetic)



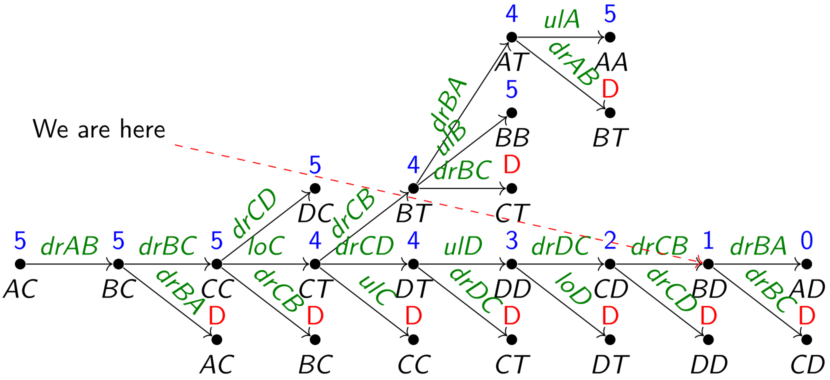
How to Relax During Search: Ignoring Deletes



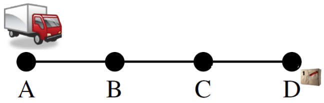
Real problem:

- State s: BD; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Successors: AD, CD.

Greedy best-first search: (tie-breaking: alphabetic)

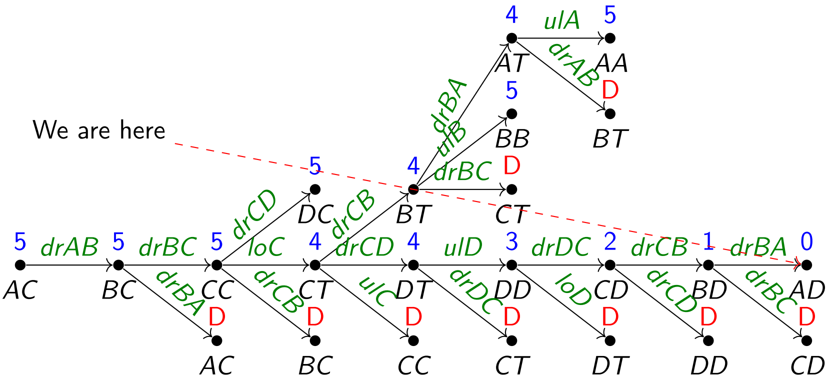


How to Relax During Search: Ignoring Deletes

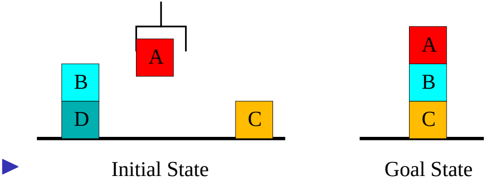


Real problem:
- State s: AD; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Goal state!

Greedy best-first search: (tie-breaking: alphabetic)



h^+ in the Blocksworld



- **Optimal plan**: $\langle \mathrm{putdown}(A), \mathrm{unstack}(B, D), \mathrm{stack}(B, C), \mathrm{pickup}(A), \mathrm{stack}(A, B) \rangle$.
- **Optimal relaxed plan**: $\langle \mathrm{stack}(A, B), \mathrm{unstack}(B, D), \mathrm{stack}(B, C) \rangle$.
- **Observation:** What can we say about the "search space surface" at the initial state here?
- The initial state lies on a local minimum under h^+, together with the successor state s where we stacked A onto B. All direct other neighbors of these two states have a strictly higher h^+ value.

18.5 Conclusion

Summary

- Heuristic search on classical search problems relies on a function h mapping states s to an estimate $h(s)$ of their 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 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*).

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** ...

Chapter 19
Searching, Planning, and Acting in the Real World

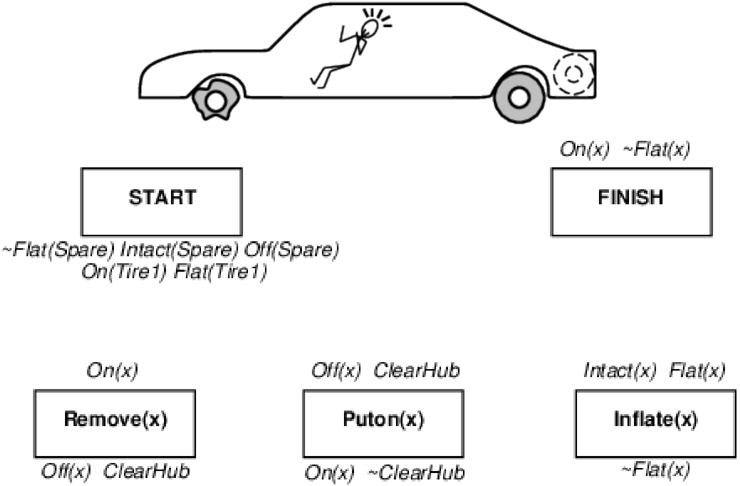
Outline

- **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

The real world

- **Example 1.1.** We have a flat tire – what to do?



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".

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)
  $[\mathit{Check}(T1), \text{if } \mathit{Intact}(T1) \text{ then } \mathit{Inflate}(T1) \text{ else } \mathit{CallAAA} \text{ fi}]$
- **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

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.
- Actions:
  - remove lid from can
  - paint object with paint from open can.

The Furniture-Coloring Example: PDDL

- **Example 2.2 (Formalization in PDDL).**
  - The PDDL domain file is as expected (actions below)

```
(define (domain furniture−coloring)
  (:predicates (object ?x) (can ?x) (inview ?x) (color ?x ?y))
  ...)
```

The Furniture-Coloring Example: PDDL

- **Example 2.3 (Formalization in PDDL).**
  - The PDDL domain file is as expected (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)))
```

The Furniture-Coloring Example: PDDL

- **Example 2.4 (Formalization in PDDL).**
  - The PDDL domain file is as expected (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.

The Furniture-Coloring Example: PDDL

- **Example 2.5 (Formalization in PDDL).**
  - The PDDL domain file is as expected (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.

The Furniture-Coloring Example: PDDL

- **Example 2.6 (Formalization in PDDL).**
  - The PDDL domain file is as expected (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))))
```

The Furniture-Coloring Example: PDDL

- **Example 2.7 (Formalization in PDDL).**
  - The PDDL domain file is as expected (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 $\leftsquigarrow$ 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 $\rightsquigarrow$ 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

19.3 Searching/Planning with Non-Deterministic Actions

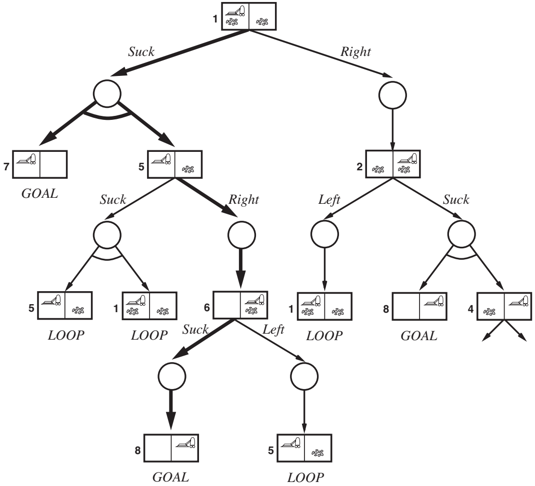
Conditional Plans

- **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)

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: $[suck, \mathrm{if}\ State = 5\ \mathrm{then}\ [right, suck]\ \mathrm{else}\ []\ \mathrm{fi}]$

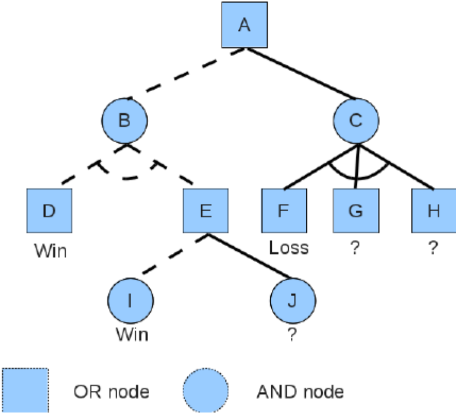
Conditional AND-OR Search (Data Structure)

- **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.

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).**



Conditional AND-OR Search (Algorithm)

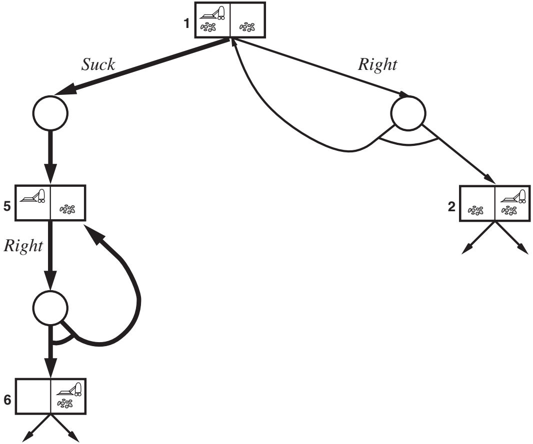
- **Definition 3.8.** AND-OR search is an algorithm for searching AND–OR graphs generated by nondeterministic environments.

```
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 ≠ 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 $\rightsquigarrow$ **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*!

The Slippery Vacuum Cleaner (try, try, try, ... try again)

- **Example 3.9 (Slippery Vacuum World).**



Moving sometimes fails
$\rightsquigarrow$ 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.

AI-1 Survey on ALeA

- Online survey evaluating ALeA from 7.02.24 to 29.02.24 24:00 (Feb last)

AI-1 Survey on ALeA

- Online survey evaluating ALeA from 7.02.24 to 29.02.24 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

AI-1 Survey on ALeA

- Online survey evaluating ALeA from 7.02.24 to 29.02.24 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

AI-1 Survey on ALeA

- Online survey evaluating ALeA from 7.02.24 to 29.02.24 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 tuesday quiz 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.

AI-1 Survey on ALeA

- Online survey evaluating ALeA from 7.02.24 to 29.02.24 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 tuesday quiz 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.

Find the Survey Here

https://ddi-survey.cs.fau.de/limesurvey/ALeA

This URL will also be posted on the forum tonight.

19.4 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 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, and
  - a sensor model that updates the belief state based on sensor information
  - a transition model that updates the belief state based on actions.

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, and
  - a sensor model that updates the belief state based on sensor information
  - a transition model that updates the belief state based on actions.
- **Idea:** The agent environment determines what the world model can be.

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, and
  - a sensor model that updates the belief state based on sensor information
  - 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 member the world state) and the transition model is a function from states and actions to states: a transition function.

World Models by Agent Type in AI-1

- **Note:** All of these considerations only give requirements to the world model. What we can do with it depends on representation and inference.

World Models by Agent Type in AI-1

- **Note:** All of these considerations only give requirements to the world model. What we can do with it depends on representation and inference.
- **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)

World Models by Agent Type in AI-1

- **Note:** All of these considerations only give requirements to the world model. What we can do with it depends on representation and inference.
- **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)

World Models by Agent Type in AI-1

- **Note:** All of these considerations only give requirements to the world model. What we can do with it depends on representation and inference.
- **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. (no decision theory covered in AI-1)

World Models by Agent Type in AI-1

- **Note:** All of these considerations only give requirements to the world model. What we can do with it depends on representation and inference.
- **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. (no decision theory covered in AI-1)
- **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)

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.
  - $\leadsto$ 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)

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.
  - $\rightsquigarrow$ 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)
- 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)

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.

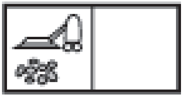
Preview: New World Models (Belief) $\rightsquigarrow$ new Agent Types

- **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 detailin the coming semester.

19.5 Searching/Planning without Observations

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).**

States	integer dirt and robot locations
Actions	*left*, *right*, *suck*, *noOp*
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: $[right, suck, left, suck]$

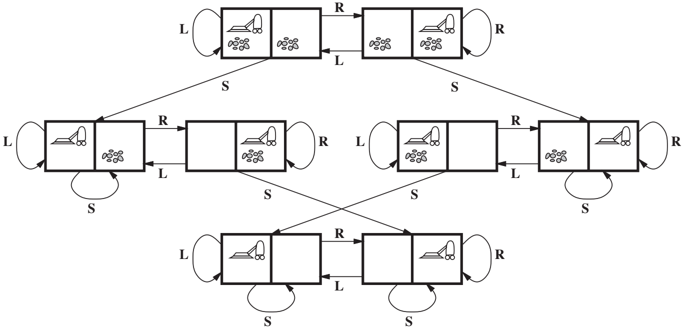
$$\begin{array}{lll} right & \rightarrow & \{2, 4, 6, 8\} \\ suck & \rightarrow & \{4, 8\} \\ left & \rightarrow & \{3, 7\} \\ suck & \rightarrow & \{7\} \end{array}$$

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 $\mathcal{T} : \mathcal{A} \times \mathcal{S} \to \mathcal{P}(\mathcal{A})$, goal states $\mathcal{G}$, and initial state $\mathcal{I}$.
- **Problem:** What is the corresponding sensorless problem?
- **Let' 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 $\mathcal{T}^b : \mathcal{A}^b \times \mathcal{S}^b \to \mathcal{P}(\mathcal{A}^b)$: 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!*

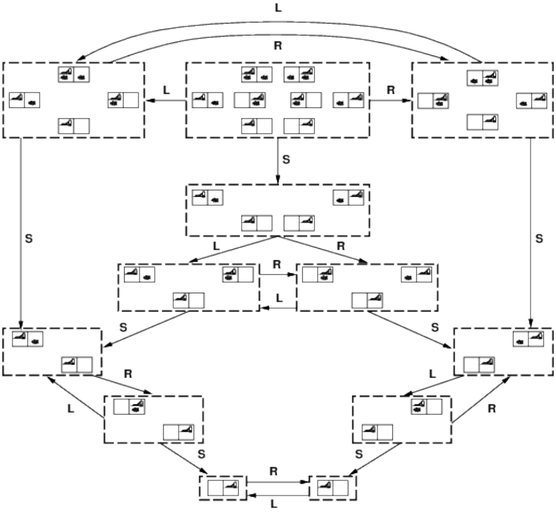
State Space vs. Belief State Space

- **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)



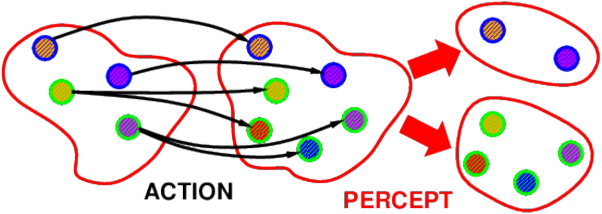
Evaluating Conformant Planning

- **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!

19.6 Searching/Planning with Observation

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.

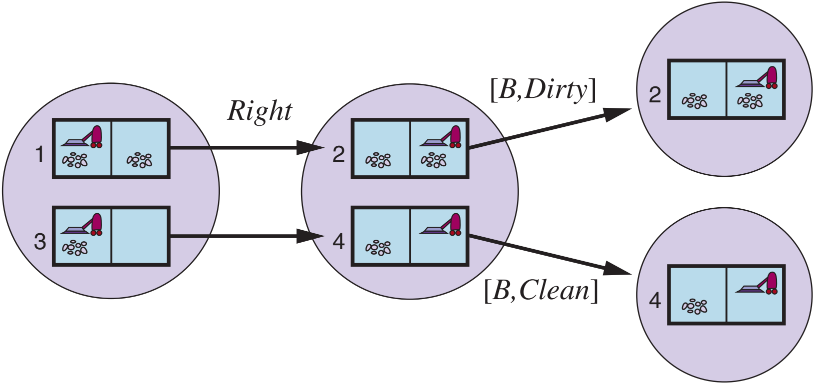
A Transition Model for Belief-State Search

- We extend the ideas from slide 651 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 $\langle \mathcal{P}(\mathcal{S}), \mathcal{A}, \mathcal{T}^b, \mathcal{S}, \{S \in \mathcal{S}^b \mid S \subseteq \mathcal{G}\} \rangle$, 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.*

Example: Local Sensing Vacuum Worlds

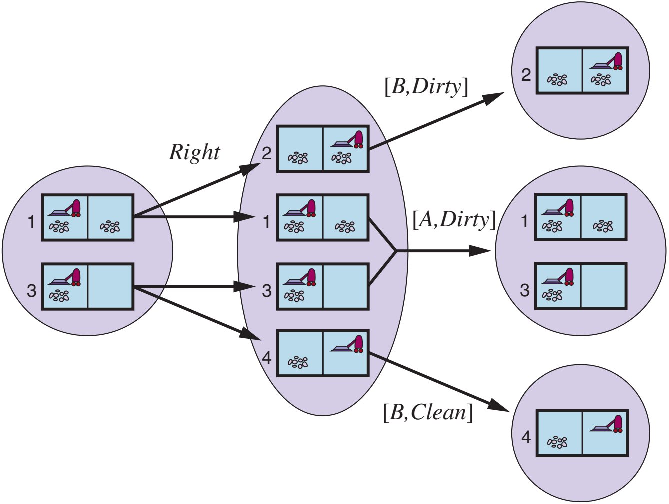
- **Example 6.4 (Transitions in the Vacuum World).** Deterministic World:



The action *Right* is deterministic, sensing disambiguates to singletons

Example: Local Sensing Vacuum Worlds

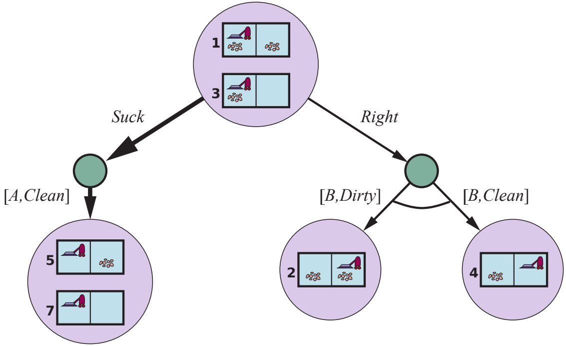
- **Example 6.5 (Transitions in the Vacuum World).** Slippery World:



The action *Right* is non-deterministic, sensing disambiguates somewhat

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: $[Suck, Right, \text{if } Bstate = \{6\} \text{ then } Suck \text{ else } [] \text{ fi}]$

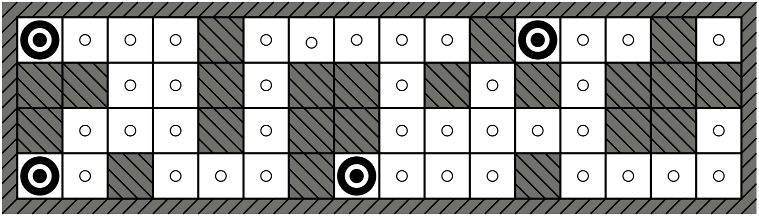
- **Note:** Belief-state-problem $\leadsto$ conditional step tests on belief-state percept (plan would not be executable in a partially observable environment otherwise)

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.

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) $\rightsquigarrow \widehat{b}_2 = \mathrm{UPDATE}(\widehat{b}_1, \mathit{NWS})$

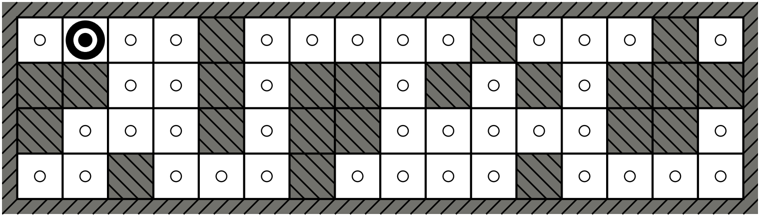


Example: Agent Localization

- **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) $\rightsquigarrow \widehat{b}_2 = \mathrm{UPDATE}(\widehat{b}_1, \mathit{NWS})$
  3. Agent executes *Move* $\rightsquigarrow \widehat{b}_3 = \mathrm{PRED}(\widehat{b}_2, \mathit{Move}) =$ *one step away from these*.

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* $\rightsquigarrow \widehat{b_4} = \mathrm{UPDATE}(\widehat{b_3}, NS)$



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) $\leadsto \widehat{b_2} = \mathrm{UPDATE}(\widehat{b_1}, NWS)$
  3. Agent executes *Move* $\leadsto \widehat{b_3} = \mathrm{PRED}(\widehat{b_2}, Move) =$ *one step away from these*.
  4. Next percept: *NS* $\leadsto \widehat{b_4} = \mathrm{UPDATE}(\widehat{b_3}, NS)$

  All in all, $\widehat{b_4} = \mathrm{UPDATE}(\mathrm{PRED}(\mathrm{UPDATE}(\widehat{b_1}, NWS), Move), NS)$ localizes the agent.
- **Observation:** PRED enlarges the belief state, while UPDATE shrinks it again.

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 $\leadsto$ done.
  - line 4/5: Condition can be satisfied by $[c_1/can]$ or $[c_2/can]$ $\leadsto$ instantiate accordingly.
- **Definition 6.14.** During plan execution the agent maintains the belief state b, chooses the branch depending on whether $b \models c$ for the condition c.
- **Note:** The planner must make sure $b \models c$ can always be decided.

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 actions: we extend the representation of effects to disjunctions.

19.7 Online Search

Online Search and Replanning

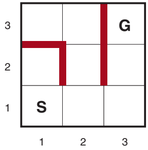
- **Note:** So far we have concentrated on offline problem solving, where the agent only acts (plan execution) after search/planning terminates.
- **Recall:** In online problem solving an agent interleaves computation and action: it computes one action at a time based on incoming perceptions.
- Online 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 $\rightsquigarrow$ exploration problem.

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.

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)



Online Search Obstacles (Dead Ends)

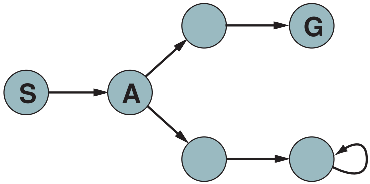
- **Definition 7.4.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.

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.*

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:

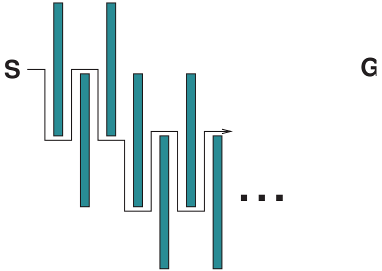


  Any agent will fail in at least one of the spaces.
- **Definition 7.19.** We call 7.6 an adversary argument.

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.

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)

Online DFS Search Agent

- **Definition 7.34.** The :

```
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 Goal Test(s′) then return stop
  if s′∉untried then untried[s′] := 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.

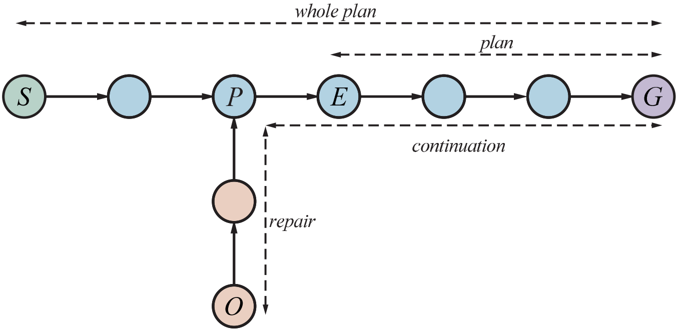
19.8 Replanning and Execution Monitoring

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).

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*

Factors in World Model Failure $\rightsquigarrow$ Monitoring

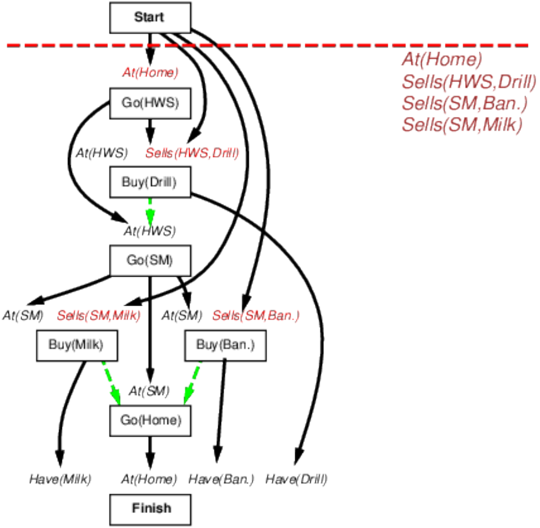
- **Generally:** The agent's world model can be incorrect, because
  - an action has a missing precondition (need a screwdriver for remove−lid)
  - an action misses an effect (painting a table gets paint on the floor)
  - it is missing a state variable (amount of paint in a can: no paint $\rightsquigarrow$ 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.

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 actions replaced by links from *Start* when done

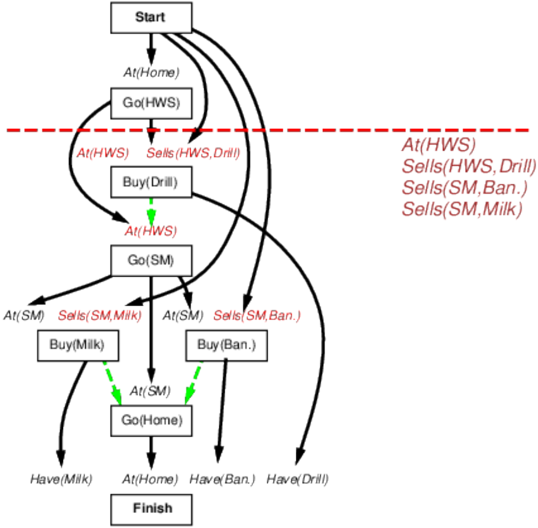
Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



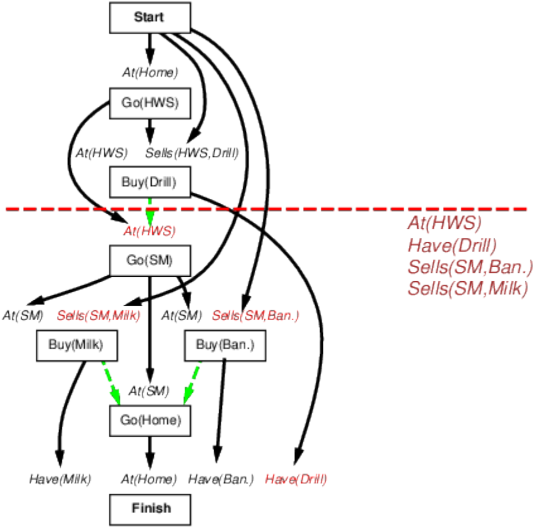
Execution Monitoring Example

- **Example 8.8 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



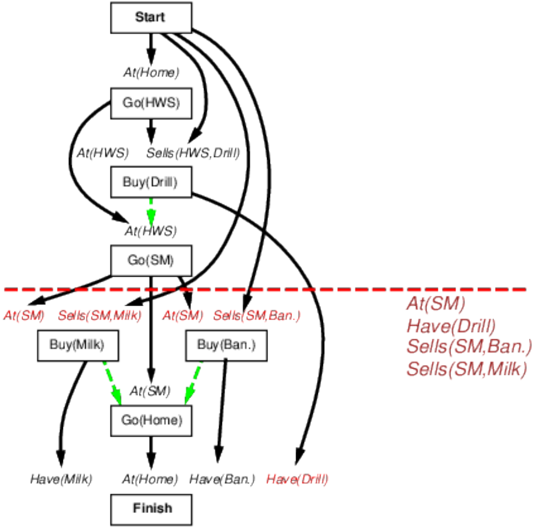
Execution Monitoring Example

- **Example 8.9 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



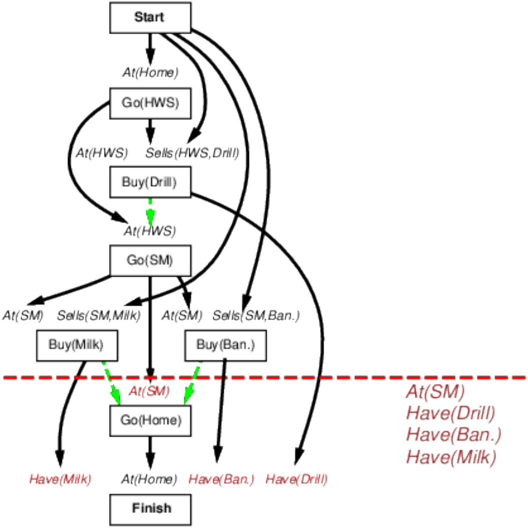
Execution Monitoring Example

- **Example 8.10 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



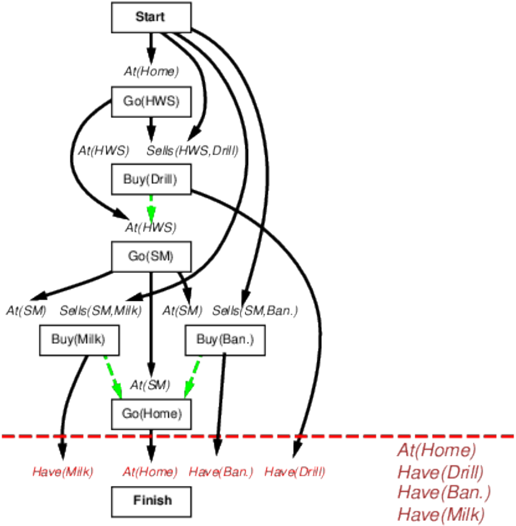
Execution Monitoring Example

- **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.



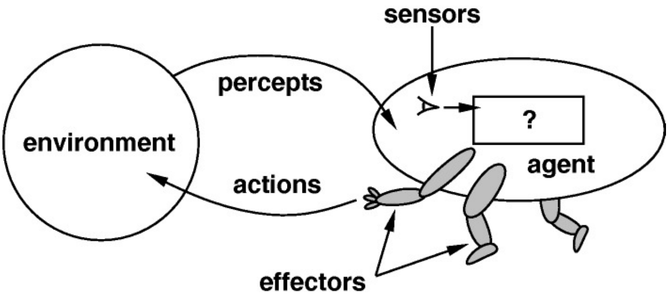
Part 5
What did we learn in AI 1?

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

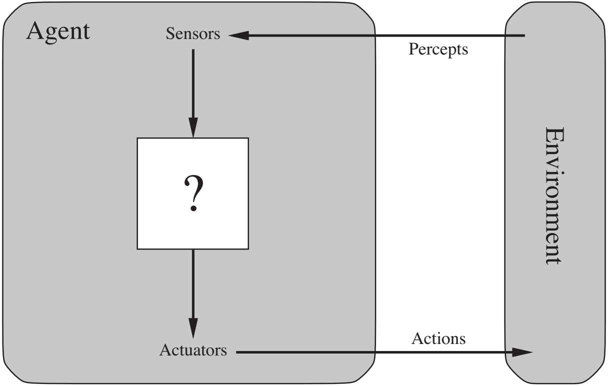
Rational Agents as an Evaluation Framework for AI

- Agents interact with the environment



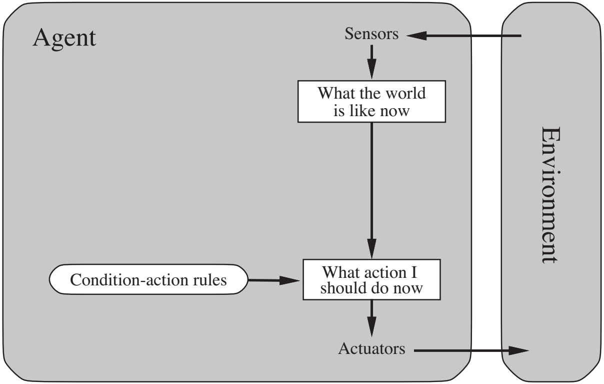
Rational Agents as an Evaluation Framework for AI

- General agent schema



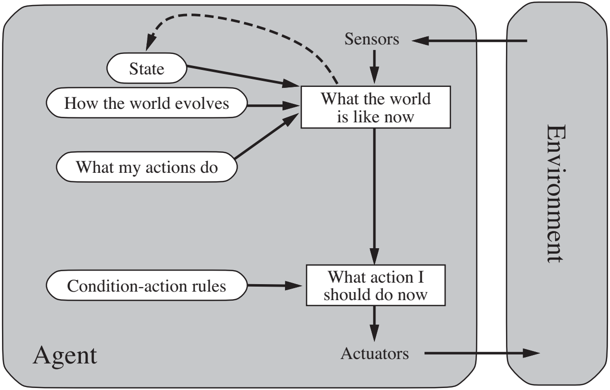
Rational Agents as an Evaluation Framework for AI

- Simple Reflex Agents



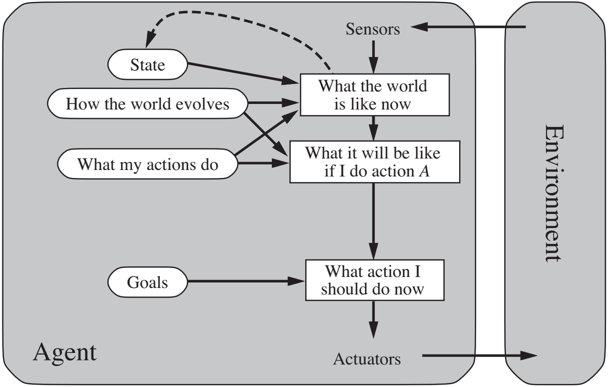
Rational Agents as an Evaluation Framework for AI

- Reflex Agents with State



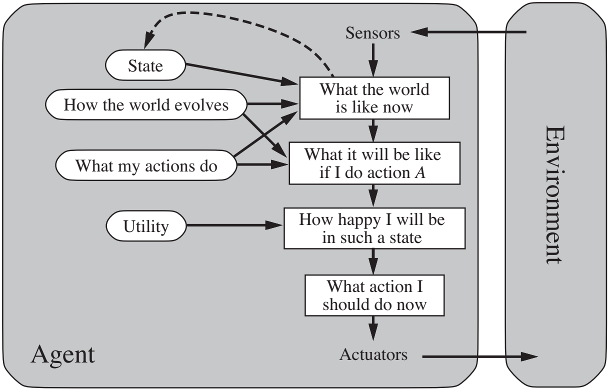
Rational Agents as an Evaluation Framework for AI

- Goal-Based Agents



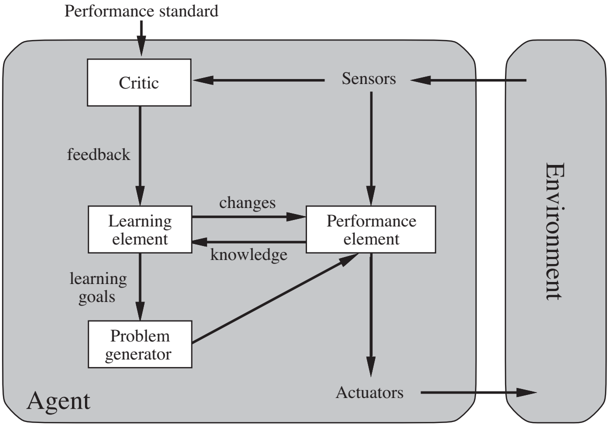
Rational Agents as an Evaluation Framework for AI

- Utility-Based Agent



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 8.13.** 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

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)

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)

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

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

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

References I

[BF95] Avrim L. Blum and Merrick L. Furst. "Fast planning through planning graph analysis". In: *Proceedings of the 14^{th} International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Chris S. Mellish. Montreal, Canada: Morgan Kaufmann, San Mateo, CA, 1995, pp. 1636–1642.

[BF97] Avrim L. Blum and Merrick L. Furst. "Fast planning through planning graph analysis". In: *Artificial Intelligence* 90.1-2 (1997), pp. 279–298.

[BG01] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search". In: *Artificial Intelligence* 129.1–2 (2001), pp. 5–33.

[BG99] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search: New Results". In: *Proceedings of the 5th European Conference on Planning (ECP'99)*. Ed. by S. Biundo and M. Fox. Springer-Verlag, 1999, pp. 60–72.

[BKS04] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. "Towards Understanding and Harnessing the Potential of Clause Learning". In: *Journal of Artificial Intelligence Research* 22 (2004), pp. 319–351.

References II

[Bon+12] Blai Bonet et al., eds. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 2012.

[Cho65] Noam Chomsky. *Syntactic structures*. Den Haag: Mouton, 1965.

[CKT91] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. "Where the *Really* Hard Problems Are". In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by John Mylopoulos and Ray Reiter. Sydney, Australia: Morgan Kaufmann, San Mateo, CA, 1991, pp. 331–337.

[CM85] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence*. Addison Wesley, 1985.

[CQ69] Allan M. Collins and M. Ross Quillian. "Retrieval time from semantic memory". In: *Journal of verbal learning and verbal behavior* 8.2 (1969), pp. 240–247. doi: 10.1016/S0022-5371(69)80069-1.

References III

[DHK15] Carmel Domshlak, Jörg Hoffmann, and Michael Katz. “Red-Black Planning: A New Systematic Approach to Partial Delete Relaxation”. In: *Artificial Intelligence* 221 (2015), pp. 73–114.

[Ede01] Stefan Edelkamp. “Planning with Pattern Databases”. In: *Proceedings of the 6th European Conference on Planning (ECP'01)*. Ed. by A. Cesta and D. Borrajo. Springer-Verlag, 2001, pp. 13–24.

[FD14] Zohar Feldman and Carmel Domshlak. “Simple Regret Optimization in Online Planning for Markov Decision Processes”. In: *Journal of Artificial Intelligence Research* 51 (2014), pp. 165–205.

[Fis] John R. Fisher. *prolog :- tutorial*. url: `https://www.cpp.edu/~jrfisher/www/prolog_tutorial/` (visited on 10/10/2019).

[FL03] Maria Fox and Derek Long. “PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains”. In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124.

References IV

[Fla94] Peter Flach. Wiley, 1994. isbn: 0471 94152 2. url: https://github.com/simply-logical/simply-logical/releases/download/v1.0/SL.pdf.

[FN71] Richard E. Fikes and Nils Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2 (1971), pp. 189–208.

[Gen34] Gerhard Gentzen. "Untersuchungen über das logische Schließen I". In: *Mathematische Zeitschrift* 39.2 (1934), pp. 176–210.

[Ger+09] Alfonso Gerevini et al. "Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners". In: *Artificial Intelligence* 173.5-6 (2009), pp. 619–668.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. BN book: Freeman, 1979.

References V

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[GS05] Carla Gomes and Bart Selman. "Can get satisfaction". In: *Nature* 435 (2005), pp. 751–752.

[GSS03] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. "Planning through Stochastic Local Search and Temporal Action Graphs". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 239–290.

[Hau85] John Haugeland. *Artificial intelligence: the very idea*. Massachusetts Institute of Technology, 1985.

[HD09] Malte Helmert and Carmel Domshlak. "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?" In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*. Ed. by Alfonso Gerevini et al. AAAI Press, 2009, pp. 162–169.

References VI

[HE05] Jörg Hoffmann and Stefan Edelkamp. "The Deterministic Part of IPC-4: An Overview". In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 519–579.

[Hel06] Malte Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246.

[Her+13] Ivan Herman et al. *RDFa 1.1 Primer – Second Edition. Rich Structured Data Markup for Web Documents*. W3C Working Goup Note. World Wide Web Consortium (W3C), Apr. 19, 2013. url: `http://www.w3.org/TR/xhtml-rdfa-primer/`.

[HG00] Patrik Haslum and Hector Geffner. "Admissible Heuristics for Optimal Planning". In: *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*. Ed. by S. Chien, R. Kambhampati, and C. Knoblock. Breckenridge, CO: AAAI Press, Menlo Park, 2000, pp. 140–149.

References VII

[HG08] Malte Helmert and Hector Geffner. “Unifying the Causal Graph and Additive Heuristics”. In: *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*. Ed. by Jussi Rintanen et al. AAAI Press, 2008, pp. 140–147.

[HHH07] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. “Flexible Abstraction Heuristics for Optimal Sequential Planning”. In: *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*. Ed. by Mark Boddy, Maria Fox, and Sylvie Thiebaux. Providence, Rhode Island, USA: Morgan Kaufmann, 2007, pp. 176–183.

[HN01] Jörg Hoffmann and Bernhard Nebel. “The FF Planning System: Fast Plan Generation Through Heuristic Search”. In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302.

[KC04] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium (W3C), Feb. 10, 2004. url: `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

References VIII

[KD09] Erez Karpas and Carmel Domshlak. "Cost-Optimal Planning with Landmarks". In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*. Ed. by C. Boutilier. Pasadena, California, USA: Morgan Kaufmann, July 2009, pp. 1728–1733.

[KHD13] Michael Katz, Jörg Hoffmann, and Carmel Domshlak. "Who Said We Need to Relax *all* Variables?" In: *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*. Ed. by Daniel Borrajo et al. Rome, Italy: AAAI Press, 2013, pp. 126–134.

[KHH12a] Michael Katz, Jörg Hoffmann, and Malte Helmert. "How to Relax a Bisimulation?" In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. Ed. by Blai Bonet et al. AAAI Press, 2012, pp. 101–109.

References IX

[KHH12b] Emil Keyder, Jörg Hoffmann, and Patrik Haslum. "Semi-Relaxed Plan Heuristics". In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. Ed. by Blai Bonet et al. AAAI Press, 2012, pp. 128–136.

[Koe+97] Jana Koehler et al. "Extending Planning Graphs to an ADL Subset". In: *Proceedings of the 4th European Conference on Planning (ECP'97)*. Ed. by S. Steel and R. Alami. Springer-Verlag, 1997, pp. 273–285. url: `ftp://ftp.informatik.uni-freiburg.de/papers/ki/koehler-etal-ecp-97.ps.gz`.

[Kow97] Robert Kowalski. "Algorithm = Logic + Control". In: *Communications of the Association for Computing Machinery* 22 (1997), pp. 424–436.

[KS00] Jana Köhler and Kilian Schuster. "Elevator Control as a Planning Problem". In: *AIPS 2000 Proceedings*. AAAI, 2000, pp. 331–338. url: `https://www.aaai.org/Papers/AIPS/2000/AIPS00-036.pdf`.

References X

[KS06] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Vol. 4212. LNCS. Springer-Verlag, 2006, pp. 282–293.

[KS92] Henry A. Kautz and Bart Selman. "Planning as Satisfiability". In: *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*. Ed. by B. Neumann. Vienna, Austria: Wiley, Aug. 1992, pp. 359–363.

[KS98] Henry A. Kautz and Bart Selman. "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI-96*. MIT Press, 1998, pp. 1194–1201.

[Kur90] Ray Kurzweil. *The Age of Intelligent Machines*. MIT Press, 1990. isbn: 0-262-11121-7.

[LPN] *Learn Prolog Now!* url: `http://lpn.swi-prolog.org/` (visited on 10/10/2019).

[LS93] George F. Luger and William A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. World Student Series. The Benjamin/Cummings, 1993. isbn: 9780805347852.

[McD+98] Drew McDermott et al. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee. 1998.

[Met+53] N. Metropolis et al. "Equations of state calculations by fast computing machines". In: *Journal of Chemical Physics* 21 (1953), pp. 1087–1091.

[Min] *Minion - Constraint Modelling*. System Web page at `http://constraintmodelling.org/minion/`. url: `http://constraintmodelling.org/minion/`.

[MSL92] David Mitchell, Bart Selman, and Hector J. Levesque. "Hard and Easy Distributions of SAT Problems". In: *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence (AAAI'92)*. San Jose, CA: MIT Press, 1992, pp. 459–465.

References XII

[NHH11] Raz Nissim, Jörg Hoffmann, and Malte Helmert. "Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*. Ed. by Toby Walsh. AAAI Press/IJCAI, 2011, pp. 1983–1990.

[Nor+18a] Emily Nordmann et al. *Lecture capture: Practical recommendations for students and lecturers*. 2018. url: https://osf.io/huydx/download.

[Nor+18b] Emily Nordmann et al. *Vorlesungsaufzeichnungen nutzen: Eine Anleitung für Studierende*. 2018. url: https://osf.io/e6r7a/download.

[NS63] Allen Newell and Herbert Simon. "GPS, a program that simulates human thought". In: *Computers and Thought*. Ed. by E. Feigenbaum and J. Feldman. McGraw-Hill, 1963, pp. 279–293.

References XIII

[OWL09] OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation. World Wide Web Consortium (W3C), Oct. 27, 2009. url: `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`.

[PD09] Knot Pipatsrisawat and Adnan Darwiche. "On the Power of Clause-Learning SAT Solvers with Restarts". In: *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP'09)*. Ed. by Ian P. Gent. Vol. 5732. Lecture Notes in Computer Science. Springer, 2009, pp. 654–668.

[Pól73] George Pólya. *How to Solve it. A New Aspect of Mathematical Method*. Princeton University Press, 1973.

[PRR97] G. Probst, St. Raub, and Kai Romhardt. *Wissen managen*. 4 (2003). Gabler Verlag, 1997.

[PS08] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Recommendation. World Wide Web Consortium (W3C), Jan. 15, 2008. url: `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.

References XIV

[PW92] J. Scott Penberthy and Daniel S. Weld. "UCPOP: A Sound, Complete, Partial Order Planner for ADL". In: *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*. Ed. by B. Nebel, W. Swartout, and C. Rich. Cambridge, MA: Morgan Kaufmann, Oct. 1992, pp. 103–114. url: ftp://ftp.cs.washington.edu/pub/ai/ucpop-kr92.ps.Z.

[RHN06] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. "Planning as satisfiability: parallel plans and algorithms for plan search". In: *Artificial Intelligence* 170.12-13 (2006), pp. 1031–1080.

[Rin10] Jussi Rintanen. "Heuristics for Planning with SAT". In: *Proceeedings of the 16th International Conference on Principles and Practice of Constraint Programming*. 2010, pp. 414–428.

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2nd ed. Pearso n Education, 2003. isbn: 0137903952.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall Press, 2009. isbn: 0136042597, 9780136042594.

References XV

[RW10] Silvia Richter and Matthias Westphal. "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 127–177.

[Sil+16] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529 (2016), pp. 484–503. url: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[SWI] *SWI Prolog Reference Manual*. url: https://www.swi-prolog.org/pldoc/refman/ (visited on 10/10/2019).

[Tur50] Alan Turing. "Computing Machinery and Intelligence". In: *Mind* 59 (1950), pp. 433–460.

[Wal75] David Waltz. "Understanding Line Drawings of Scenes with Shadows". In: *The Psychology of Computer Vision*. Ed. by P. H. Winston. McGraw-Hill, 1975, pp. 1–19.