Artificial Intelligence 1
Winter Semester 2024/25

– Lecture Notes –

Prof. Dr. Michael Kohlhase

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

2025-02-06

0.1 Preface

0.1.1 Course Concept

Objective: The course aims at giving students a solid (and often somewhat theoretically oriented) foundation of the basic concepts and practices of artificial intelligence. The course will predominantly cover symbolic AI – also sometimes called "good old-fashioned AI (GofAI)" – in the first semester and offers the very foundations of statistical approaches in the second. Indeed, a full account sub symbolic, machine learning based AI deserves its own specialization courses and needs much more mathematical prerequisites than we can assume in this course.

Context: The course "Artificial Intelligence" (AI 1 & 2) at FAU Erlangen is a two-semester course in the "Wahlpflichtbereich" (specialization phase) in semester 5/6 of the bachelor program "Computer Science" at FAU Erlangen. It is also available as a (somewhat remedial) course in the "Vertiefungsmodul Künstliche Intelligenz" in the Computer Science Master's program.

Prerequisites: AI-1 & 2 builds on the mandatory courses in the FAU bachelor's program, in particular the course "Grundlagen der Logik in der Informatik" [Glo], which already covers a lot of the materials usually presented in the "knowledge and reasoning" part of an introductory AI course. The AI 1& 2 course also minimizes overlap with the course.

The course is relatively elementary, we expect that any student who attended the mandatory CS course at FAU Erlangen can follow it.

Open to external students: Other bachelor programs are increasingly co-opting the course as specialization option. There is no inherent restriction to computer science students in this course. Students with other study biographies – e.g. students from other bachelor programs our external Master's students should be able to pick up the prerequisites when needed.

0.1.2 Course Contents

Goal: To give students a solid foundation of the basic concepts and practices of the field of Artificial Intelligence. The course will be based on Russell/Norvig's book "*Artificial Intelligence; A modern Approach*" [RN09]

Artificial Intelligence I (the first semester): introduces AI as an area of study, discusses "rational agents" as a unifying conceptual paradigm for AI and covers problem solving, search, constraint propagation, logic, knowledge representation, and planning.

Artificial Intelligence II (the second semester): is more oriented towards exposing students to the basics of statistically based AI: We start out with reasoning under uncertainty, setting the foundation with Bayesian Networks and extending this to rational decision theory. Building on this we cover the basics of machine learning.

0.1.3 This Document

Format: The document mixes the slides presented in class with comments of the instructor to give students a more complete background reference.

Caveat: This document is made available for the students of this course only. It is still very much a draft and will develop over the course of the current course and in coming academic years. **Licensing:** This document is licensed under a Creative Commons license that requires attribution, allows commercial use, and allows derivative works as long as these are licensed under the same license. **Knowledge Representation Experiment:** This document is also an experiment in knowledge representation. Under the hood, it uses the sTeX package [Koh08; sTeX], a TeX/LaTeX extension for semantic markup, which allows to export the contents into active documents that adapt to the reader and can be instrumented with services based on the explicitly represented meaning of the documents.

0.1.4 Acknowledgments

Materials: Most of the materials in this course is based on Russel/Norvik's book "Artificial Intelligence — A Modern Approach" (AIMA [RN95]). Even the slides are based on a LaTeX-based slide set, but heavily edited. The section on search algorithms is based on materials obtained from Bernhard Beckert (then Uni Koblenz), which is in turn based on AIMA. Some extensions have been inspired by an AI course by Jörg Hoffmann and Wolfgang Wahlster at Saarland University in 2016. Finally Dennis Müller suggested and supplied some extensions on AGI. Florian Rabe, Max Rapp and Katja Berčič have carefully re-read the text and pointed out problems.

All course materials have been restructured and semantically annotated in the STEX format, so that we can base additional semantic services on them.

AI Students: The following students have submitted corrections and suggestions to this and earlier versions of the notes: Rares Ambrus, Ioan Sucan, Yashodan Nevatia, Dennis Müller, Simon Rainer, Demian Vöhringer, Lorenz Gorse, Philipp Reger, Benedikt Lorch, Maximilian Lösch, Luca Reeb, Marius Frinken, Peter Eichinger, Oskar Herrmann, Daniel Höfer, Stephan Mattejat, Matthias Sonntag, Jan Urfei, Tanja Würsching, Adrian Kretschmer, Tobias Schmidt, Maxim Onciul, Armin Roth, Liam Corona, Tobias Völk, Lena Voigt, Yinan Shao, Michael Girstl, Matthias Vietz, Anatoliy Cherepantsev, Stefan Musevski, Matthias Lobenhofer, Philipp Kaludercic, Diwarkara Reddy, Martin Helmke, Stefan Müller, Dominik Mehlich, Paul Martini, Vishwang Dave, Arthur Miehlich, Christian Schabesberger, Vishaal Saravanan, Simon Heilig, Michelle Fribrance, Wenwen Wang, Xinyuan Tu, Lobna Eldeeb.

0.1.5 Recorded Syllabus

The recorded syllabus – a record the progress of the course in the academic year 2024/25– is in the course page in the ALeA system at `https://courses.voll-ki.fau.de/course-home/ai-1`. The table of contents in the AI-1 notes at `https://courses.voll-ki.fau.de` indicates the material covered to date in yellow.

The recorded syllabus of AI-2 can be found at `https://courses.voll-ki.fau.de/course-home/ai-2`. For the topics planned for this course, see **??**.

Contents

0.1 Preface . . . i
0.1.1 Course Concept . . . i
0.1.2 Course Contents . . . i
0.1.3 This Document . . . i
0.1.4 Acknowledgments . . . ii
0.1.5 Recorded Syllabus . . . ii

1 Preliminaries 1
1.1 Administrative Ground Rules . . . 1
1.2 Getting Most out of AI-1 . . . 4
1.3 Learning Resources for AI-1 . . . 6
1.4 AI-Supported Learning . . . 9

2 AI – Who?, What?, When?, Where?, and Why? 19
2.1 What is Artificial Intelligence? . . . 19
2.2 Artificial Intelligence is here today! . . . 21
2.3 Ways to Attack the AI Problem . . . 25
2.4 Strong vs. Weak AI . . . 27
2.5 AI Topics Covered . . . 29
2.6 AI in the KWARC Group . . . 30

I Getting Started with AI: A Conceptual Framework 33

3 Logic Programming 37
3.1 Introduction to Logic Programming and ProLog . . . 37
3.2 Programming as Search . . . 41
3.2.1 Running Prolog . . . 41
3.2.2 Knowledge Bases and Backtracking . . . 42
3.2.3 Programming Features . . . 44
3.2.4 Advanced Relational Programming . . . 46

4 Recap of Prerequisites from Math & Theoretical Computer Science 49
4.1 Recap: Complexity Analysis in AI? . . . 49
4.2 Recap: Formal Languages and Grammars . . . 55
4.3 Mathematical Language Recap . . . 61

5 Rational Agents: An AI Framework 65
5.1 Introduction: Rationality in Artificial Intelligence . . . 65
5.2 Agent/Env. as a Framework . . . 69
5.3 Good Behavior $\rightsquigarrow$ Rationality . . . 72
5.4 Classifying Environments . . . 74
5.5 Types of Agents . . . 75

5.6 Representing the Environment in Agents 82
5.7 Rational Agents: Summary 83

II General Problem Solving 85

6 Problem Solving and Search 89
6.1 Problem Solving 89
6.2 Problem Types 92
6.3 Search 96
6.4 Uninformed Search Strategies 99
6.4.1 Breadth-First Search Strategies 100
6.4.2 Depth-First Search Strategies 104
6.4.3 Further Topics 111
6.5 Informed Search Strategies 112
6.5.1 Greedy Search 113
6.5.2 Heuristics and their Properties 117
6.5.3 A-Star Search 119
6.5.4 Finding Good Heuristics 125
6.6 Local Search 127

7 Adversarial Search for Game Playing 133
7.1 Introduction 133
7.2 Minimax Search 137
7.3 Evaluation Functions 145
7.4 Alpha-Beta Search 148
7.5 Monte-Carlo Tree Search (MCTS) 160
7.6 State of the Art 165
7.7 Conclusion 166

8 Constraint Satisfaction Problems 169
8.1 Constraint Satisfaction Problems: Motivation 169
8.2 The Waltz Algorithm 174
8.3 CSP: Towards a Formal Definition 177
8.4 Constraint Networks: Formalizing Binary CSPs 180
8.5 CSP as Search 182
8.6 Conclusion & Preview 187

9 Constraint Propagation 189
9.1 Introduction 189
9.2 Constraint Propagation/Inference 190
9.3 Forward Checking 194
9.4 Arc Consistency 196
9.5 Decomposition: Constraint Graphs, and Three Simple Cases 204
9.6 Cutset Conditioning 209
9.7 Constraint Propagation with Local Search 211
9.8 Conclusion & Summary 212

III Knowledge and Inference 215

10 Propositional Logic & Reasoning, Part I: Principles 219
10.1 Introduction: Inference with Structured State Representations 219
10.1.1 A Running Example: The Wumpus World 219
10.1.2 Propositional Logic: Preview 222

10.1.3 Propositional Logic: Agenda . . . 224
10.2 Propositional Logic (Syntax/Semantics) . . . 224
10.3 Inference in Propositional Logics . . . 230
10.4 Propositional Natural Deduction Calculus . . . 233
10.5 Predicate Logic Without Quantifiers . . . 238
10.6 Conclusion . . . 241

11 Formal Systems 243

12 Machine-Oriented Calculi for Propositional Logic 247
12.1 Test Calculi . . . 247
12.1.1 Normal Forms . . . 248
12.2 Analytical Tableaux . . . 249
12.2.1 Analytical Tableaux . . . 249
12.2.2 Practical Enhancements for Tableaux . . . 253
12.2.3 Soundness and Termination of Tableaux . . . 254
12.3 Resolution for Propositional Logic . . . 256
12.3.1 Resolution for Propositional Logic . . . 256
12.3.2 Killing a Wumpus with Propositional Inference . . . 259
12.4 Conclusion . . . 261

13 Propositional Reasoning: SAT Solvers 263
13.1 Introduction . . . 263
13.2 Davis-Putnam . . . 265
13.3 DPLL $\widehat{=}$ (A Restricted Form of) Resolution . . . 267
13.4 Conclusion . . . 270

14 First-Order Predicate Logic 273
14.1 Motivation: A more Expressive Language . . . 273
14.2 First-Order Logic . . . 277
14.2.1 First-Order Logic: Syntax and Semantics . . . 277
14.2.2 First-Order Substitutions . . . 281
14.3 First-Order Natural Deduction . . . 284
14.4 Conclusion . . . 288

15 Automated Theorem Proving in First-Order Logic 291
15.1 First-Order Inference with Tableaux . . . 291
15.1.1 First-Order Tableau Calculi . . . 291
15.1.2 First-Order Unification . . . 295
15.1.3 Efficient Unification . . . 300
15.1.4 Implementing First-Order Tableaux . . . 303
15.2 First-Order Resolution . . . 305
15.2.1 Resolution Examples . . . 306
15.3 Logic Programming as Resolution Theorem Proving . . . 308
15.4 Summary: ATP in First-Order Logic . . . 311

16 Knowledge Representation and the Semantic Web 313
16.1 Introduction to Knowledge Representation . . . 313
16.1.1 Knowledge & Representation . . . 313
16.1.2 Semantic Networks . . . 315
16.1.3 The Semantic Web . . . 320
16.1.4 Other Knowledge Representation Approaches . . . 325
16.2 Logic-Based Knowledge Representation . . . 326
16.2.1 Propositional Logic as a Set Description Language . . . 327
16.2.2 Ontologies and Description Logics . . . 330

16.2.3 Description Logics and Inference . . . 332
16.3 A simple Description Logic: ALC . . . 334
16.3.1 Basic ALC: Concepts, Roles, and Quantification . . . 335
16.3.2 Inference for ALC . . . 339
16.3.3 ABoxes, Instance Testing, and ALC . . . 346
16.4 Description Logics and the Semantic Web . . . 348

IV Planning & Acting 357

17 Planning I: Framework 361
17.1 Logic-Based Planning . . . 362
17.2 Planning: Introduction . . . 366
17.3 Planning History . . . 373
17.4 STRIPS Planning . . . 375
17.5 Partial Order Planning . . . 381
17.6 PDDL Language . . . 397
17.7 Conclusion . . . 400

18 Planning II: Algorithms 401
18.1 Introduction . . . 401
18.2 How to Relax . . . 403
18.3 Delete Relaxation . . . 415
18.4 The h^+Heuristic . . . 421
18.5 Conclusion . . . 433

19 Searching, Planning, and Acting in the Real World 435
19.1 Introduction . . . 435
19.2 The Furniture Coloring Example . . . 437
19.3 Searching/Planning with Non-Deterministic Actions . . . 438
19.4 Agent Architectures based on Belief States . . . 441
19.5 Searching/Planning without Observations . . . 443
19.6 Searching/Planning with Observation . . . 446
19.7 Online Search . . . 451
19.8 Replanning and Execution Monitoring . . . 454

V What did we learn in AI 1? 459

VI Excursions 473

A Completeness of Calculi for Propositional Logic 477
A.1 Abstract Consistency and Model Existence . . . 477
A.2 A Completeness Proof for Propositional Tableaux . . . 483

B Conflict Driven Clause Learning 485
B.1 UP Conflict Analysis . . . 485
B.2 Clause Learning . . . 490
B.3 Phase Transitions . . . 494

C Completeness of Calculi for First-Order Logic **499**
C.1 Abstract Consistency and Model Existence 499
C.2 A Completeness Proof for First-Order ND 505
C.3 Soundness and Completeness of First-Order Tableaux 507
C.4 Soundness and Completeness of First-Order Resolution 508

Chapter 1

Preliminaries

In this chapter, we want to get all the organizational matters out of the way, so that we can get into the discussion of artificial intelligence content unencumbered. We will talk about the necessary administrative details, go into how students can get most out of the course, talk about where the various resources provided with the course can be found, and finally introduce the ALeA system, an experimental – using AI methods – learning support system for the AI course.

1.1 Administrative Ground Rules

We will now go through the ground rules for the course. This is a kind of a social contract between the instructor and the students. Both have to keep their side of the deal to make learning as efficient and painless as possible.

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)

Note: I do not literally presuppose the courses on the slide above – most of you do not have a bachelor's degree from FAU, so you cannot have taken them. And indeed some of the content of these courses is irrelevant for AI-1. Stating these courses is just the easiest way to specifying what content I will be building on – and any graduate courses has to build on something.

Many of you will have taken the moral equivalent of these courses in your undergraduate studies at your home university. If you did not, you will have to somehow catch up on the content as we go along in AI-1. This should be possible with enough motivation.
There are essentially three skillsets that are essential for AI-1:

1. A solid understanding and practical skill in programming (whatever programming language)
2. A good understanding and practice in using mathematical language to represent complex structures
3. A solid understanding of formal languages and grammars, as well as applied complexity theory (basics of theoretical computer science).

Without (catching up on) these the AI-1 course will be quite frustrating and hard.

We will briefly go over the most important topics in **??** to synchronize concepts and notation. Note that if you do not have a formal education in courses like the ones mentioned above you will very probably have to do significant remedial work.
Now we come to a topic that is always interesting to the students: the grading scheme.

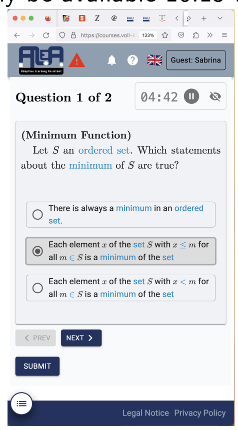
Assessment, Grades

- **Overall (Module) Grade:**
  - Grade via the exam (Klausur) $\leadsto$ 100% of the grade.
  - Up to 10% bonus on-top for an exam with $\geq 50\%$ points.($< 50\%$ $\leadsto$ no bonus)
  - Bonus points $\widehat{=}$ percentage sum of the best 10 prepquizzes divided by 100.
- **Exam:** 90 minutes exam conducted in presence on paper! ($\sim$ April 1. 2025)
- **Retake Exam:** 90 min exam six months later. ($\sim$ October 1. 2025)
- ⚠ Register for exams in `https://campo.fau.de`. (there is a deadine!)
- **Note:** You can de-register from an exam on `https://campo.fau.de` up to three working days before exam. (do not miss that if you are not prepared)

Preparedness Quizzes

- **PrepQuizzes:** Every tuesday 16:15 we start the lecture with a 10 min online quiz – the PrepQuiz – 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 prepquiz will be given in the ALeA system

- https://courses.voll-ki.fau.de/quiz-dash/ai-1
- You have to be logged into ALeA! (via FAU IDM)
- You can take the prepquiz on your laptop or phone, ...
- ... in the lecture or at home ...
- ... via WLAN or 4G Network. (do not overload)
- Prepquizzes will only be available 16:15-16:25!



This Thursday: Pretest

- ⚠ This thursday we will try out the prepquiz infrastructure with a pretest!
  - **Presence**: bring your laptop or cellphone.
  - **Online**: you can and should take the pretest as well.
  - Have a recent firefox or chrome (chrome: younger than March 2023)
  - Make sure that you are logged into ALeA (via FAU IDM; see below)
- **Definition 1.1.1.** 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 prepquizzes.
- Participation in the pretest is optional; it will not influence grades in any way.
- The pretest 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)

Due to the current AI hype, the course Artificial Intelligence is very popular and thus many degree programs at FAU have adopted it for their curricula. Sometimes the course setup that fits for the CS program does not fit the other's very well, therefore there are some special conditions. I want to state here.

Special Admin Conditions

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

I can only warn of what I am aware, so if your degree program lets you jump through extra hoops, please tell me and then I can mention them here.

1.2 Getting Most out of AI-1

In this section we will discuss a couple of measures that students may want to consider to get most out of the AI-1 course.

None of the things discussed in this section – homeworks, tutorials, study groups, and attendance – are mandatory (we cannot force you to do them; we offer them to you as learning opportunities), but most of them are very clearly correlated with success (i.e. passing the exam and getting a good grade), so taking advantage of them may be in your own interest.

AI-1 Homework Assignments

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

It is very well-established experience that without doing the homework assignments (or something similar) on your own, you will not master the concepts, you will not even be able to ask sensible questions, and take very little home from the course. Just sitting in the course and nodding is not enough!

AI-1 Homework Assignments – Howto

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

FAU Michael Kohlhase: Artificial Intelligence 1 7 2025-02-06

If you have questions please make sure you discuss them with the instructor, the teaching assistants, or your fellow students. There are three sensible venues for such discussions: online in the lectures, in the tutorials, which we discuss now, or in the course forum – see below. Finally, it is always a very good idea to form study groups with your friends.

Tutorials for Artificial Intelligence 1

- **Approach:** Weekly tutorials and homework assignments (first one in week two)
- **Goal 1:** Reinforce what was taught in the lectures. (you need practice)
- **Goal 2:** Allow you to ask any question you have in a protected environment.
- **Instructor/Lead TA:** Florian Rabe (KWARC Postdoc)
  - Room: 11.137 @ Händler building, `florian.rabe@fau.de`
- **Tutorials:** One each taught by Florian Rabe (lead); Yasmeen Shawat, Hatem Mousa, Xinyuan Tu, and Florian Guthmann.

▷ **Life-saving Advice:** Go to your tutorial, and prepare for it by having looked at the slides and the homework assignments!

Collaboration

▷ **Definition 1.2.1.** Collaboration (or cooperation) is the process of groups of agents acting together for common, mutual benefit, as opposed to acting in competition for selfish benefit. In a collaboration, every agent contributes to the common goal and benefits from the contributions of others.

▷ In learning situations, the benefit is "better learning".

▷ **Observation:** In collaborative learning, the overall result can be significantly better than in competitive learning.

▷ **Good Practice:** Form study groups. (long- or short-term)

1. ⚠ those learners who work most, learn most!
2. ⚠ freeloaders – individuals 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)

As we said above, almost all of the components of the AI-1 course are optional. That even applies to attendance. But make no mistake, attendance is important to most of you. Let me explain, . . .

Do I need to attend the AI-1 Lectures

▷ Attendance is not mandatory for the AI-1 course. (official version)

▷ **Note:** There are two ways of learning: (both are OK, your mileage may vary)

- ▷ Approach B: Read a book/papers (here: lecture notes)
- ▷ Approach I: come to the lectures, be involved, interrupt the instructor whenever you have a question.

The only advantage of I over B is that books/papers do not answer questions

▷ Approach S: come to the lectures and sleep does not work!

▷ The closer you get to research, the more we need to discuss!

1.3 Learning Resources for AI-1

But what if you are not in a lecture or tutorial and want to find out more about the AI-1 topics?

Textbook, Handouts and Information, Forums, Videos

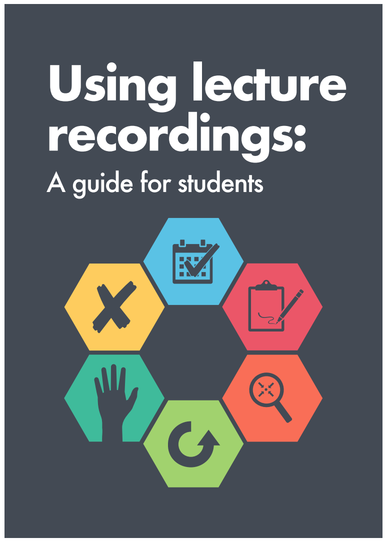
- **Textbook:** *Russel/Norvig: Artificial Intelligence, A modern Approach* [RN09].
  - basically "broad but somewhat shallow"
  - great to get intuitions on the basics of AI

  Make sure that you read the edition ≥ 3 ⇜ vastly improved over ≤ 2.
- **Lecture notes:** will be posted at `https://kwarc.info/teaching/AI`
  - more detailed than [RN09] in some areas
  - I mostly prepare them as we go along (semantically preloaded ⇝ research resource)
  - please e-mail me any errors/shortcomings you notice. (improve for the group)
- **Course Videos:** AI-1 will be streamed/recorded at `https://fau.tv/course/id/4047`
  - **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 exam!
- **StudOn Forum:** `https://www.studon.fau.de/crs5832535.html` for
  - announcements, homeworks (my view on the forum)
  - questions, discussion among your fellow students (your forum too, use it!)

FAU has issued a very insightful guide on using lecture videos. It is a good idea to heed these recommendations, even if they seem annoying at first.

Practical recommendations on Lecture Videos

- **Excellent Guide:** [Nor+18a] (German version at [Nor+18b])



- Attend lectures.
- Take notes.
- Be specific.
- Catch up.
- Ask for help.
- Don't cut corners.

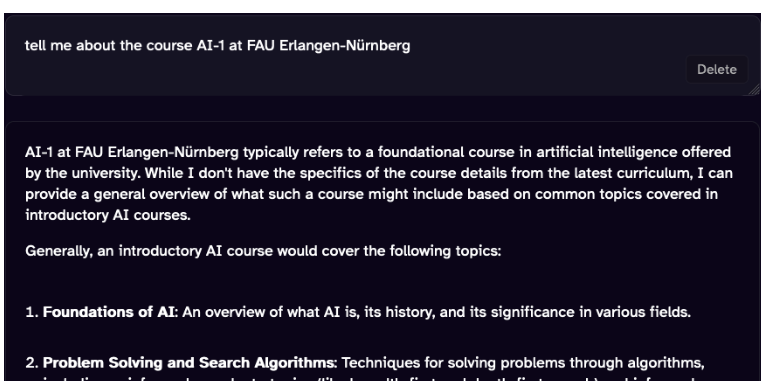
NOT a Resource for : LLMs – AI-based tools like ChatGPT

▷ **Definition 1.3.1.** A large language model (LLM) is a computational model capable of language generation or other natural language processing tasks.

▷ **Example 1.3.2.** OpenAI's GPT, Google's Bard, and Meta's Llama.

▷ **Definition 1.3.3.** A chatbot is a software application or web interface that is designed to mimic human conversation through text or voice interactions. Modern chatbots are usually based on LLMs.

▷ **Example 1.3.4 (ChatGPT talks about AI-1).** (but remains vague)



▷ **Note:** LLM-based chatbots invent *every word*! (surprisingly often correct)

▷ **Example 1.3.5 (In the AI-1 exam).** ChatGPT scores ca. 50% of the points.

- ▷ ChatGPT can almost pass the exam ... (We could award it a Master's degree)
- ▷ But can you? (the AI-1 exams will be in person on paper)

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

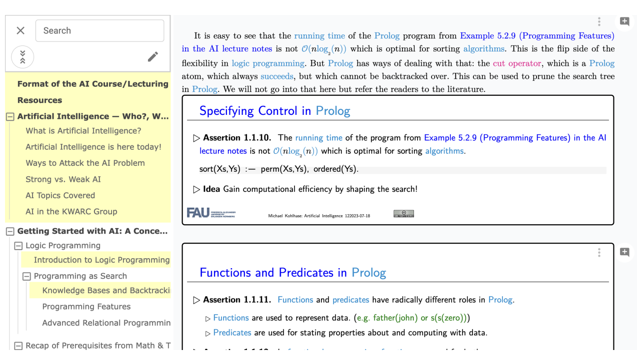
- **Intuition:** AI tools like GhatGPT, CoPilot, etc. (see also [She24])
  - can help you solve problems, (valuable tools in production situations)
  - hinders learning if used for homeworks/quizzes, etc. (like driving instead of jogging)
- **What (not) to do:** (to get most of the brave new AI-supported world)
  - try out these tools to get a first-hand intuition what they can/cannot do
  - challenge yourself while learning so that you can also do it (mind over matter!)

1.4 AI-Supported Learning

In this section we introduce the ALeA (Adaptive Learning Assistant) system, a learning support system we have developed using symbolic AI methods – the stuff we learn about in AI-1 – and which we will use to support students in the course. As such, ALeA does double duty in the AI-1 course it supports learning activities and serves as a showcase, what symbolic AI methods can to in an important application.

ALeA: Adaptive Learning Assistant

- **Idea:** Use AI methods to help teach/learn AI (AI4AI)
- **Concretely:** Provide HTML versions of the AI-1 slides/lecture notes and embed learning support services into them. (for pre/postparation of lectures)
- **Definition 1.4.1.** Call a document active, iff it is interactive and adapts to specific information needs of the readers. (lecture notes on steroids)
- **Intuition:** ALeA serves active course materials. (PDF mostly inactive)
- **Goal:** Make ALeA more like a instructor + study group than like a book!
- **Example 1.4.2 (Course Notes).** $\widehat{=}$ Slides + Comments



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

FAU Michael Kohlhase: Artificial Intelligence 1 14 2025-02-06

The central idea in the AI4AI approach – using AI to support learning AI – and thus the ALeA system is that we want to make course materials – i.e. what we give to students for preparing and postparing lectures – more like teachers and study groups (only available 24/7) than like static books.

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

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

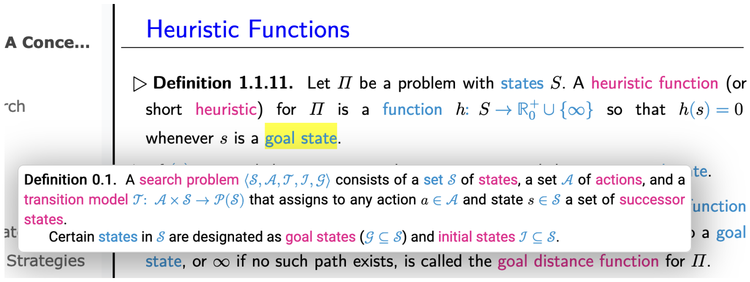
- **AI-1 in ALeA:** https://courses.voll-ki.fau.de/course-home/ai-1
  - All details for the course.
  - recorded syllabus (keep track of material covered in course)
  - syllabus of the last semesters (for over/preview)
- **ALeA Status:** The ALEA system is deployed at FAU for over 1000 students taking eight courses
  - (some) students use the system actively (our logs tell us)
  - reviews are mostly positive/enthusiastic (error reports pour in)

FAU Michael Kohlhase: Artificial Intelligence 1 15 2025-02-06

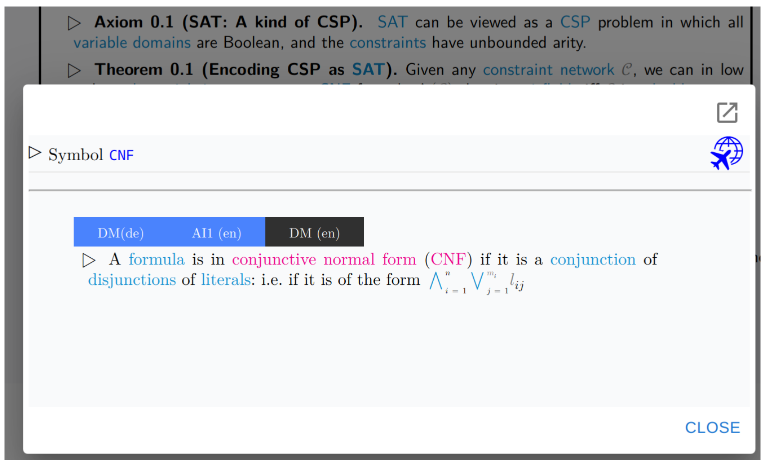
The ALEA AI-1 page is the central entry point for working with the ALeA system. You can get to all the components of the system, including two presentations of the course contents (notes- and slides-centric ones), the flashcards, the localized forum, and the quiz dashboard.
We now come to the heart of the ALeA system: its learning support services, which we will now briefly introduce. Note that this presentation is not really sufficient to undertstand what you may be getting out of them, you will have to try them, and interact with them sufficiently that the learner model can get a good estimate of your competencies to adapt the results to you.

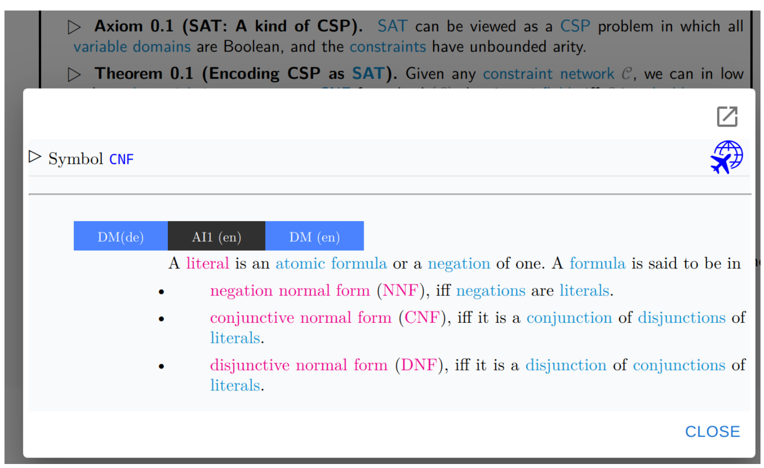
Learning Support Services in ALEA

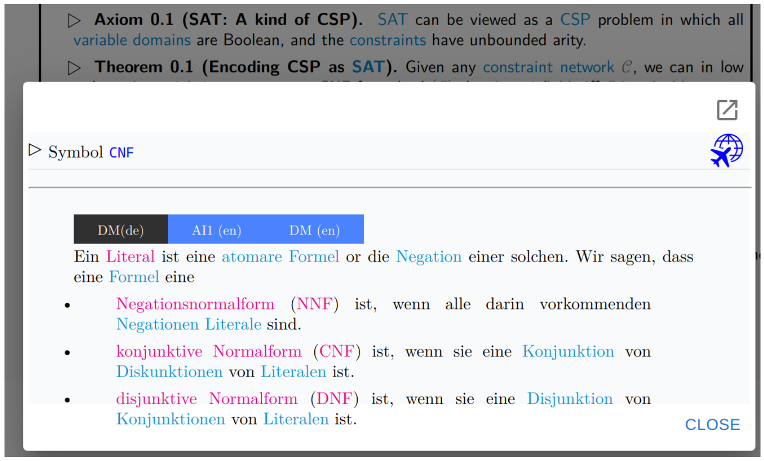
- **Idea:** Embed learning support services into active course materials.
- **Example 1.4.3 (Definition on Hover).** Hovering on a (cyan) term reference reminds us of its definition. (even works recursively)



▷ **Example 1.4.4 (More Definitions on Click).** Clicking on a (cyan) term reference shows us more definitions from other contexts.







▷ **Example 1.4.5 (Guided Tour).** A guided tour for a concept c assembles definitions/etc. into a self-cont

c = countable ⇝

▷ . . . your idea here . . . (the sky is the limit)

Note that this is only an initial collection of learning support services, we are constantly working on additional ones. Look out for feature notifications () on the upper right hand of the ALeA screen.

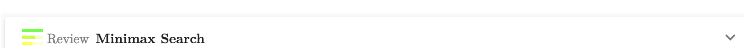
(Practice/Remedial) Problems Everywhere

▷ **Problem:** Learning requires a mix of understanding and test-driven practice.

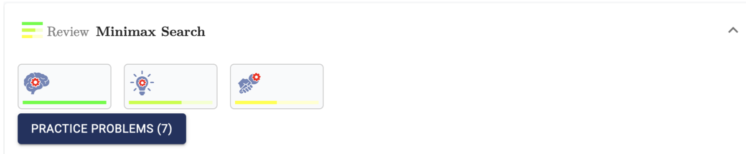
▷ **Idea:** ALeA supplies targeted practice problems everywhere.

▷ **Concretely:** Revision markers at the end of sections.

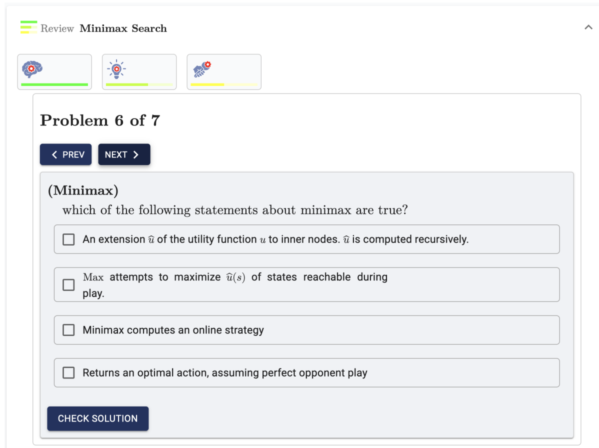
- ▷ A relatively non-intrusive overview over competency



- ▷ Click to extend it for details.



▷ Practice problems as usual. (targeted to your specific competency)



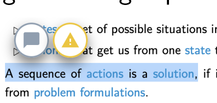
While the learning support services up to now have been adressed to individual learners, we now turn to services addressed to communities of learners, ranging from study groups with three learners, to whole courses, and even – eventually – all the alumni of a course, if they have not de-registered from ALeA.

Currently, the community aspect of ALeA only consists in localized interactions with the course materials.
The ALeA system uses the semantic structure of the course materials to localize some interactions that are otherwise often from separate applications. Here we see two:

1. one for reporting content errors – and thus making the material better for all learners – and‘‘
2. a localized course forum, where forum threads can be attached to learning objects.

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 user helpfulness.

▷ Notes can be anonymous (⇝ generate no karma)

Let us briefly look into how the learning support services introduced above might work, focusing on where the necessary information might come from. Even though some of the concepts in the discussion below may be new to AI-1 students, it is worth looking into them. Bear with us as we try to explain the AI components of the ALeA system.

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

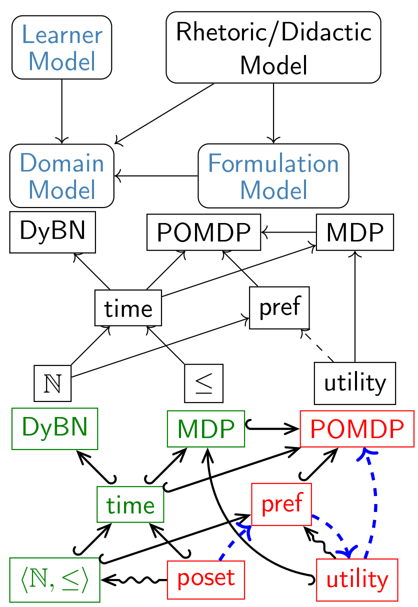
▷ **Idea:** Do what a teacher does! Use/maintain four models:

▷ **Ingredient 1:** Domain model $\widehat{=}$ knowledge/theory graph

▷ **Ingredient 2:** Learner model $\widehat{=}$ adding competency estimations

▷ **Ingredient 3:** A collection of ready-formulated learning objects

▷ **Ingredient 4:** Educational dialogue planner ⇝ guided tours



(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

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.

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

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

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

The dialogue planner assembles learning objects into active course material using

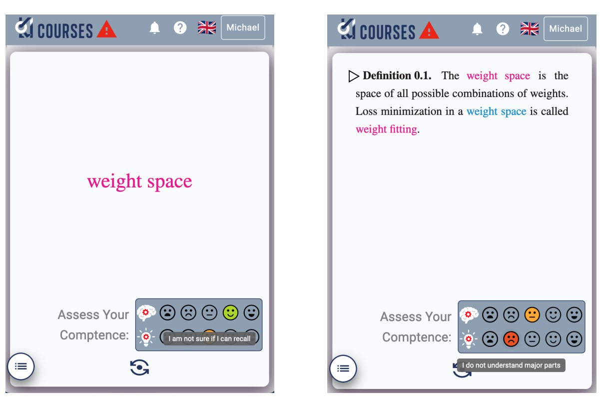
- ▷ the domain model and didactic relations to determine the order of LOs
- ▷ the learner model to determine what to show
- ▷ the rhetoric relations to make the dialogue coherent

Michael Kohlhase: Artificial Intelligence 1 19 2025-02-06

We can use the same four models discussed in the space of guided tours to deploy additional learning support services, which we now discuss.

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)

We have already seen above how the learner model can drive the drilling with flashcards. It can also be used for the configuration of card stacks by configuring a domain e.g. a section in the course materials and a competency threshold. We now come to a very important issue that we always face when we do AI systems that interface with humans. Most web technology companies that take one the approach "the user pays for the services with their personal data, which is sold on" or integrate advertising for renumeration. Both are not acceptable in university setting.

But abstaining from monetizing personal data still leaves the problem how to protect it from intentional or accidental misuse. Even though the GDPR has quite extensive exceptions for research, the ALeA system – a research prototype – adheres to the principles and mandates of the GDPR. In particular it makes sure that personal data of the learners is only used in learning support services directly or indirectly initiated by the learners themselves.

Learner Data and Privacy in ALeA

▷ **Observation:** Learning support services in ALeA use the learner model; they
- ▷ need the learner model data to adapt to the invidivual learner!
- ▷ collect learner interaction data (to update the learner model)

▷ **Consequence:** You need to be logged in (via your FAU IDM credentials) for useful learning support services!

▷ **Problem:** Learner model data is highly sensitive personal data!

▷ **ALeA Promise:** The ALeA team does the utmost to keep your personal data safe. (SSO via FAU IDM/eduGAIN, ALeA trust zone)

▷ **ALeA Privacy Axioms:**

1. ALeA only collects learner models data about logged in users.
2. Personally identifiable learner model data is only accessible to its subject (delegation possible)
3. Learners can always query the learner model about its data.
4. All learner model data can be purged without negative consequences (except usability deterioration)
5. Logging into ALeA is completely optional.

▷ **Observation:** Authentication for bonus quizzes are somewhat less optional, but you can always purge the learner model later.

So, now that you have an overview over what the ALeA system can do for you, let us see what you have to concretely do to be able to use it.

Concrete Todos for ALeA

▷ **Recall:** You will use ALeA for the prepquizzes (or lose bonus points)
All other use is optional. (but AI-supported pre/postparation can be helpful)

▷ To use the ALeA system, you will have to log in via SSO: (do it now)

- ▷ go to `https://courses.voll-ki.fau.de/course-home/ai-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)

FAU Michael Kohlhase: Artificial Intelligence 1 22 2025-02-06

Even if you did not understand some of the AI jargon or the underlying methods (yet), you should be good to go for using the ALeA system in your day-to-day work.

Chapter 2

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

We start the course by giving an overview of (the problems, methods, and issues of) Artificial Intelligence, and what has been achieved so far.

Naturally, this will dwell mostly on philosophical aspects – we will try to understand what the important issues might be and what questions we should even be asking. What the most important avenues of attacks may be and where AI research is being carried out.

In particular the discussion will be very non-technical – we have very little basis to discuss technicalities yet. But stay with me, this will drastically change very soon. **A Video Nugget** covering the introduction of this chapter can be found at `https://fau.tv/clip/id/21467`.

Plot for this chapter

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

FAU Michael Kohlhase: Artificial Intelligence 1 23 2025-02-06

2.1 What is Artificial Intelligence?

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21701`.
The first question we have to ask ourselves is "What is Artificial Intelligence?", i.e. how can we define it. And already that poses a problem since the natural definition *like human intelligence, but artificially realized* presupposes a definition of intelligence, which is equally problematic; even Psychologists and Philosophers – the subjects nominally "in charge" of natural intelligence – have problems defining it, as witnessed by the plethora of theories e.g. found at [WHI].

What is Artificial Intelligence? Definition

▷ **Definition 2.1.1 (According to Wikipedia).** Artificial Intelligence (AI) is intelligence exhibited by machines

▷ **Definition 2.1.2 (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 2.1.3 (Elaine Rich).** Artificial Intelligence (AI) studies how we can make the computer do things that humans can still do better at the moment.

Maybe we can get around the problems of defining "what artificial intelligence is", by just describing the necessary components of AI (and how they interact). Let's have a try to see whether that is more informative.

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.

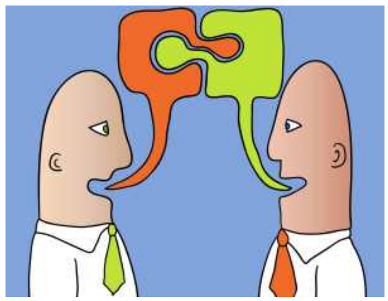
▷ This needs a combination of

the ability to learn

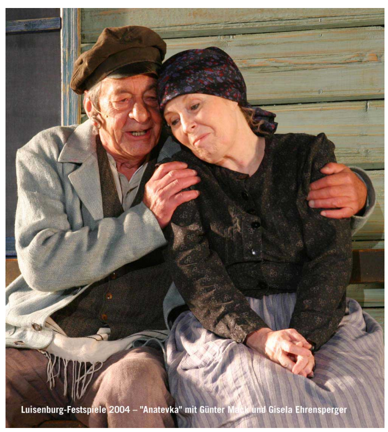
Inference

Perception

Language understanding



Emotion

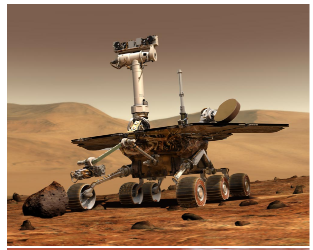


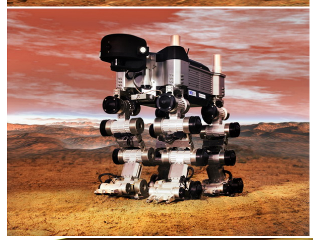
Note that list of components is controversial as well. Some say that it lumps together cognitive capacities that should be distinguished or forgets others, ... We state it here much more to get AI-1 students to think about the issues than to make it normative.

2.2 Artificial Intelligence is here today!

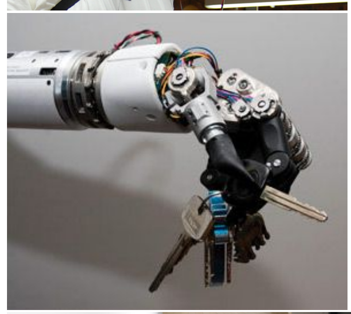
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21697`.
The components of Artificial Intelligence are quite daunting, and none of them are fully understood, much less achieved artificially. But for some tasks we can get by with much less. And indeed that is what the field of Artificial Intelligence does in practice – but keeps the lofty ideal around. This practice of "trying to achieve AI in selected and restricted domains" (cf. the discussion starting with slide 32) has borne rich fruits: systems that meet or exceed human capabilities in such areas. Such systems are in common use in many domains of application.

Artificial Intelligence is here today!



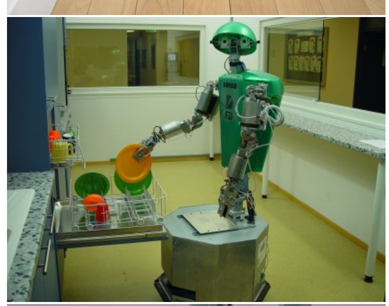


- ▷ in outer space
  - ▹ in outer space systems need autonomous control:
  - ▹ remote control impossible due to time lag



- ▷ in artificial limbs
  - ▹ the user controls the prosthesis via existing nerves, can e.g. grip a sheet of paper.

- ▷ 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
  - ▹ in the USA 90% of the prostate operations are carried out by RoboDoc
  - ▹ Paro is a cuddly robot that eases solitude in nursing homes.

We will conclude this section with a note of caution.

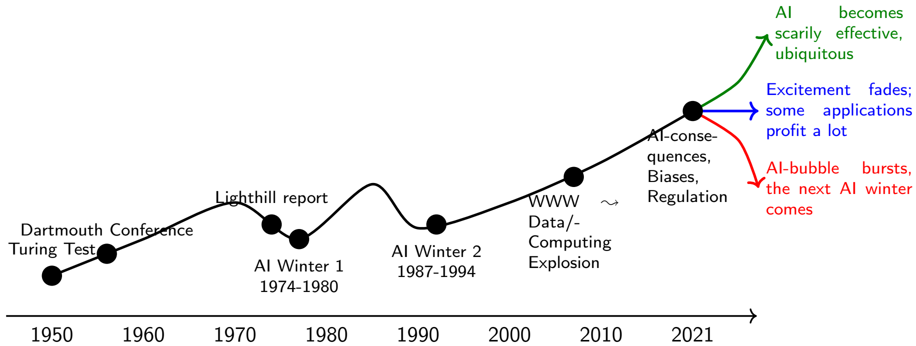
The AI Conundrum

- **Observation:** Reserving the term "Artificial Intelligence" has been quite a land grab!
- **But:** researchers at the Dartmouth Conference (1956) really thought they would solve/reach AI in two/three decades.
- **Consequence:** AI still asks the big questions. (and still promises answers soon)
- **Another Consequence:** AI as a field is an incubator for many innovative technologies.
- **AI Conundrum:** Once AI solves a subfield it is called "computer science". (becomes a separate subfield of CS)
- **Example 2.2.1.** Functional/Logic Programming, automated theorem proving, Planning, machine learning, Knowledge Representation, . . .
- **Still Consequence:** AI research was alternatingly flooded with money and cut off brutally.

All of these phenomena can be seen in the growth of AI as an academic discipline over the course of its now over 70 year long history.

The current AI Hype — Part of a longer Story

- The history of AI as a discipline has been very much tied to the amount of funding – that allows us to do research and development.
- Funding levels are tied to public perception of success (especially for AI)
- **Definition 2.2.2.** An AI winter is a time period of low public perception and funding for AI,
  mostly because AI has failed to deliver on its – sometimes overblown – promises
  An AI summer is a time period of high public perception and funding for AI
- A potted history of AI (AI summers and summers)



Michael Kohlhase: Artificial Intelligence 1 28 2025-02-06

Of course, the future of AI is still unclear, we are currently in a massive hype caused by the advent of deep neural networks being trained on all the data of the Internet, using the computational power of huge compute farms owned by an oligopoly of massive technology companies – we are definitely in an AI summer.

But AI as a academic community and the tech industry also make outrageous promises, and the media pick it up and distort it out of proportion, . . . So public opinion could flip again, sending AI into the next winter.

2.3 Ways to Attack the AI Problem

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21717`.
There are currently three main avenues of attack to the problem of building artificially intelligent systems. The (historically) first is based on the symbolic representation of knowledge about the world and uses inference-based methods to derive new knowledge on which to base action decisions. The second uses statistical methods to deal with uncertainty about the world state and learning methods to derive new (uncertain) world assumptions to act on.

Four Main Approaches to Artificial Intelligence

- ▷ **Definition 2.3.1.** 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.
- ▷ **Definition 2.3.2.** 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 2.3.3.** 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 2.3.4.** 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.

FAU Michael Kohlhase: Artificial Intelligence 1 29 2025-02-06

As a consequence, the field of Artificial Intelligence (AI) is an engineering field at the intersection of computer science (logic, programming, applied statistics), Cognitive Science (psychology, neuroscience), philosophy (can machines think, what does that mean?), linguistics (natural language understanding), and mechatronics (robot hardware, sensors).

Subsymbolic AI and in particular machine learning is currently hyped to such an extent, that many people take it to be synonymous with "Artificial Intelligence". It is one of the goals of this course to show students that this is a very impoverished view.

Two ways of reaching Artificial Intelligence?

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

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

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

FAU Michael Kohlhase: Artificial Intelligence 1 30 2025-02-06

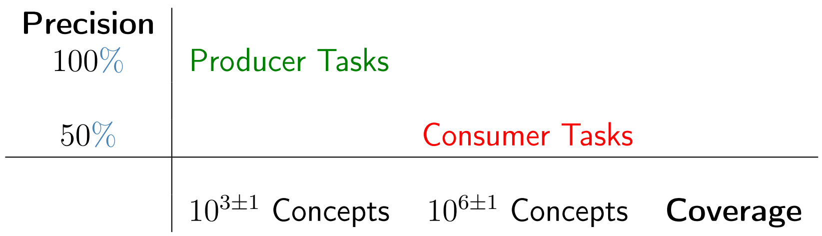
We combine the topics in this way in this course, not only because this reproduces the historical development but also as the methods of statistical and subsymbolic AI share a common basis.

It is important to notice that all approaches to AI have their application domains and strong points. We will now see that exactly the two areas, where symbolic AI and statistical/subsymbolic AI have their respective fortes correspond to natural application areas.

Environmental Niches for both Approaches to AI

- ▷ **Observation:** There are two kinds of applications/tasks in AI
  - ▷ Consumer tasks: consumer grade applications have tasks that must be fully generic and wide coverage. (e.g. machine translation like Google Translate)
  - ▷ Producer tasks: producer grade applications must be high-precision, but can be

domain-specific (e.g. multilingual documentation, machinery-control, program verification, medical technology)



after Aarne Ranta [Ran17].

- ▷ **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.

An example of a producer task – indeed this is where the name comes from – is the case of a machine tool manufacturer T, which produces digitally programmed machine tools worth multiple million Euro and sells them into dozens of countries. Thus T must also provide comprehensive machine operation manuals, a non-trivial undertaking, since no two machines are identical and they must be translated into many languages, leading to hundreds of documents. As those manual share a lot of semantic content, their management should be supported by AI techniques. It is critical that these methods maintain a high precision, operation errors can easily lead to very costly machine damage and loss of production. On the other hand, the domain of these manuals is quite restricted. A machine tool has a couple of hundred components only that can be described by a couple of thousand attributes only.

Indeed companies like T employ high-precision AI techniques like the ones we will cover in this course successfully; they are just not so much in the public eye as the consumer tasks.

2.4 Strong vs. Weak AI

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21724`.
To get this out of the way before we begin: We now come to a distinction that is often muddled in popular discussions about "Artificial Intelligence", but should be cristal clear to students of the course AI-1 – after all, you are upcoming "AI-specialists".

Strong AI vs. Narrow AI

- ▷ **Definition 2.4.1.** 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 2.4.2.** 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 2.4.3.** Problems requiring strong AI to solve are called AI hard, and AI complete, iff AGI should be able to solve them all.

- **In short:** We can characterize the difference intuitively:
  - narrow AI: What (most) computer scientists think AI is / should be.
  - strong AI: What Hollywood authors think AI is / should be.
- **Needless to say** we are only going to cover narrow AI in this course!

One can usually defuse public worries about "is AI going to take control over the world" by just explaining the difference between strong AI and weak AI clearly.
I would like to add a few words on AGI, that – if you adopt them; they are not universally accepted – will strengthen the arguments differentiating between strong and weak AI.

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 2.4.4.** You should not train to be an accountant or truck driver! (bots will replace you soon)

I want to conclude this section with an overview over the recent protagonists – both personal and institutional – of AGI.

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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21719`.
We will now preview the topics covered by the course "Artificial Intelligence" in the next two semesters.

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

▷ **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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21725`.

Now allow me to beat my own drum. In my research group at FAU, we do research on a particular kind of Artificial Intelligence: logic, language, and information. This may not be the most fashionable or well-hyped area in AI, but it is challenging, well-respected, and – most importantly – fun.

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

Research in the KWARC group ranges over a variety of topics, which range from foundations of mathematics to relatively applied web information systems. I will try to organize them into three pillars here.

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:	**KM & Interaction**:	**Semantization**:
▷ MathML, *OpenMath*	▷ Semantic Interpretation (aka. Framing)	▷ LaTeXML: LaTeX ⇝ XML
▷ advanced Type Theories	▷ math-literate interaction	▷ sTeX: Semantic LaTeX
▷ MMT: Meta Meta Theory	▷ MathHub: math archives & active docs	▷ invasive editors
▷ Logic Morphisms/Atlas	▷ Active documents: embedded semantic services	▷ Context-Aware IDEs
▷ Theorem Prover/CAS Interoperability	▷ Model-based Education	▷ Mathematical Corpora
▷ Mathematical Models/Simulation		▷ Linguistics of Math
		▷ ML for Math Semantics Extraction

Foundations: Computational Logic, Web Technologies, OMDoc/MMT

For all of these areas, we are looking for bright and motivated students to work with us. This can take various forms, theses, internships, and paid students assistantships.

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, (Meta)^n-Frameworks
  - ▷ Math Corpus Linguistics: Semantics Extraction
  - ▷ Serious Games, Cognitive Engineering, Math Information Retrieval, Legal Reasoning, . . .
  - ▷ . . . last but not least: KWARC is the home of ALeA!
- ▷ We always try to find a topic at the intersection of your and our interests.
- ▷ We also sometimes have positions!. (HiWi, Ph.D.: $\frac{1}{2}$ E-13, PostDoc: full E-13)

FAU Michael Kohlhase: Artificial Intelligence 1 40 2025-02-06

Sciences like physics or geology, and engineering need high-powered equipment to perform measurements or experiments. computer science and in particular the KWARC group needs high powered human brains to build systems and conduct thought experiments.

The KWARC group may not always have as much funding as other AI research groups, but we are very dedicated to give the best possible research guidance to the students we supervise.

So if this appeals to you, please come by and talk to us.

Part I

Getting Started with AI: A Conceptual Framework

This part of the lecture notes sets the stage for the technical parts of the course by establishing a common framework (Rational Agents) that gives context and ties together the various methods discussed in the course.

After having seen what AI can do and where AI is being employed today (see **??**), we will now

1. introduce a programming language to use in the course,
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 that we will need throughout the course.

ad 1. Prolog: For the programming language we choose Prolog, historically one of the most influential "AI programming languages". While the other AI programming language: `Lisp` which gave rise to the functional programming programming paradigm has been superseded by typed languages like SML, Haskell, `Scala`, and F#, Prolog is still the prime example of the declarative programming paradigm. So using Prolog in this course gives students the opportunity to explore this paradigm. At the same time, Prolog is well-suited for trying out algorithms in symbolic AI the topic of this semester since it internalizes the more complex primitives of the algorithms presented here.

ad 2. Rational Agents: The conceptual framework centers around rational agents which combine aspects of purely cognitive architectures (an original concern for the field of AI) with the more recent realization that intelligence must interact with the world (embodied AI) to grow and learn. The cognitive architectures aspect allows us to place and relate the various algorithms and methods we will see in this course. Unfortunately, the "situated AI" aspect will not be covered in this course due to the lack of time and hardware.

ad 3. Topics of Theoretical Computer Science: When we evaluate the methods and algorithms introduced in AI-1, we will need to judge their suitability as agent functions. The main theoretical tool for that is complexity theory; we will give a short motivation and overview of the main methods and results as far as they are relevant for AI-1 in **??**.

In the second half of the semester we will transition from search-based methods for problem solving to inference-based ones, i.e. where the problem formulation is described as expressions of a formal language which are transformed until an expression is reached from which the solution can be read off. Phrase structure grammars are the method of choice for describing such languages; we will introduce/recap them in **??**.

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

We will now learn a new programming paradigm: logic programming, which is one of the most influential paradigms in AI. We are going to study Prolog (the oldest and most widely used) as a concrete example of ideas behind logic programming and use it for our homeworks in this course.

As Prolog is a representative of a programming paradigm that is new to most students, programming will feel weird and tedious at first. But subtracting the unusual syntax and program organization logic programming really only amounts to recursive programming just as in functional programming (the other declarative programming paradigm). So the usual advice applies, keep staring at it and practice on easy examples until the pain goes away.

3.1 Introduction to Logic Programming and ProLog

Logic programming is a programming paradigm that differs from functional and imperative programming in the basic procedural intuition. Instead of transforming the state of the memory by issuing instructions (as in imperative programming), or computing the value of a function on some arguments, logic programming interprets the program as a body of knowledge about the respective situation, which can be queried for consequences.

This is actually a very natural conception of program; after all we usually run (imperative or functional) programs if we want some question answered. **Video Nuggets** covering this section can be found at `https://fau.tv/clip/id/21752` and `https://fau.tv/clip/id/21753`.

.

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 3.1.1.**

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.

We now formally define the language of Prolog, starting off the atomic building blocks.

Prolog Terms and Literals

- **Definition 3.1.2.** Prolog expresses knowledge about the world via
  - constants denoted by lowercase strings,
  - variables denoted by strings starting with an uppercase letter or _, and
  - functions and predicates (lowercase strings) applied to terms.
- **Definition 3.1.3.** 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 3.1.4.** The following are
  - Prolog terms: john, X, _, father(john), ...
  - Prolog literals: loves(john,mary), loves(john,_), loves(john,wife_of(john)),...

Now we build up Prolog programs from those building blocks.

Prolog Programs: Facts and Rules

- **Definition 3.1.5.** A Prolog program is a sequence of clauses, i.e.
  - facts of the form l., where l is a literal, (a literal and a dot)
  - rules of the form h:−b_1,...,b_n., where $n > 0$. h is called the head literal (or simply head) and the b_i are together called the body of the rule.

  A rule h:−b_1,...,b_n., should be read as *h (is true) if b_1 and ... and b_n are*.
- **Example 3.1.6.** Write "something is a car if it has a motor and four wheels" as car(X) :− has_motor(X),has_wheels(X,4). (variables are uppercase)
  This is just an ASCII notation for $m(x) \wedge w(x,4) \Rightarrow car(x)$.
- **Example 3.1.7.** 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.

The whole point of writing down a knowledge base (a Prolog program with knowledge about the situation), if we do not have to write down *all* the knowledge, but a (small) subset, from which the rest follows. We have already seen how this can be done: with logic. For logic programming we will use a logic called "first-order logic" which we will not formally introduce here.

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 3.1.8.** 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.

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

?? introduces a very important distinction: that between a Prolog program and the knowledge base it induces. Whereas the former is a finite, syntactic object (essentially a string), the latter may be an infinite set of facts, which represents the totality of knowledge about the world or the aspects described by the program.

As knowledge bases can be infinite, we cannot pre-compute them. Instead, logic programming languages compute fragments of the knowledge base by need; i.e. whenever a user wants to check membership; we call this approach querying: the user enters a query expression and the system answers yes or no. This answer is computed in a depth first search process.

Querying the Knowledge Base: Size Matters

- ▷ **Idea:** We want to see whether a fact is in the knowledge base.
- ▷ **Definition 3.1.9.** A query is a list of Prolog literals 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 3.1.10.** 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 3.1.11.** 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 3.1.12.** 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 3.1.13 (Backchaining).** We continue ??

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


Note that backchaining replaces the current query with the body of the rule suitably instantiated. For rules with a long body this extends the list of current goals, but for facts (rules without a body), backchaining shortens the list of current goals. Once there are no goals left, the Prolog interpreter finishes and signals success by issuing the string **true**.

If no rules match the current subgoal, then the interpreter terminates and signals failure with the string false,

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 3.1.14.** We vary ?? using 0 instead of zero.

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


We can extend querying from simple yes/no answers to programs that return values by simply using variables in queries. In this case, the Prolog interpreter returns a substitution.

Querying the Knowledge base: Answer Substitutions

▷ **Definition 3.1.15.** 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 3.1.16.** 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
```


In ?? the first backchaining step binds the variable X to the query variable Y, which gives us the two subgoals has_wheels(Y,4),has_motor(Y). which again have the query variable Y. The next backchaining step binds this to mybmw, and the third backchaining step exhausts the subgoals. So the query succeeds with the (overall) answer substitution Y = mybmw. With this setup, we can already do the "fallible Greeks" example from the introduction.

PROLOG: Are there Fallible Greeks?

▷ **Program:**

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

▷ **Example 3.1.17 (Query).** ?−fallible(X),greek(X).

▷ **Answer substitution:** $[\text{sokrates}/X]$

3.2 Programming as Search

In this section, we want to really use Prolog as a programming language, so let use first get our tools set up. **Video Nuggets** covering this section can be found at https://fau.tv/clip/id/21754 and https://fau.tv/clip/id/21827.

3.2.1 Running Prolog

We will now discuss how to use a Prolog interpreter to get to know the language. The SWI Prolog interpreter can be downloaded from http://www.swi-prolog.org/. To start the Prolog interpreter with pl or prolog or swipl from the shell. The SWI manual is available at http://www.swi-prolog.org/pldoc/

We will introduce working with the interpreter using unary natural numbers as examples: we first add the fact[1] to the knowledge base

```
unat(zero).
```

which asserts that the predicate unat[2] is **true** on the term zero. Generally, we can add a fact to the knowledge base either by writing it into a file (e.g. example.pl) and then "consulting it" by writing one of the following three commands into the interpreter:

```
[example]
consult('example.pl').
consult('example').
```

or by directly typing

```
assert(unat(zero)).
```

into the Prolog interpreter. Next tell Prolog about the following rule

```
assert(unat(suc(X)) :- unat(X)).
```

which gives the Prolog runtime an initial (infinite) knowledge base, which can be queried by

```
?- unat(suc(suc(zero))).
```

Even though we can use any text editor to program Prolog, but running Prolog in a modern editor with language support is incredibly nicer than at the command line, because you can see the whole history of what you have done. Its better for debugging too.

3.2.2 Knowledge Bases and Backtracking

Depth-First Search with Backtracking

- So far, all the examples led to direct success or to failure. (simple KB)
- **Definition 3.2.1 (Prolog Search Procedure).** The Prolog interpreter employs top-down, left-right depth first search, concretely, Prolog search:
  - works on the subgoals in left right order.
  - matches first query with the head literals of the clauses in the program in top-down order.
  - if there are no matches, fail and backtracks to the (chronologically) last backtrack point.
  - otherwise backchain on the first match, keep the other matches in mind for backtracking via backtrack points.

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

Note: With the Prolog search procedure detailed above, computation can easily go into infinite loops, even though the knowledge base could provide the correct answer. Consider for instance the simple program

[1] for "unary natural numbers"; we cannot use the predicate nat and the constructor function s here, since their meaning is predefined in Prolog

[2] for "unary natural numbers".

```
p(X):− p(X).
p(X):− q(X).
q(X).
```

If we query this with ?− p(john), then DFS will go into an infinite loop because Prolog expands by default the first predicate. However, we can conclude that p(john) is true if we start expanding the second predicate.

In fact this is a necessary feature and not a bug for a programming language: we need to be able to write non-terminating programs, since the language would not be Turing complete otherwise. The argument can be sketched as follows: we have seen that for Turing machines the halting problem is undecidable. So if all Prolog programs were terminating, then Prolog would be weaker than Turing machines and thus not Turing complete.

We will now fortify our intuition about the Prolog search procedure by an example that extends the setup from ?? by a new choice of a vehicle that could be a car (if it had a motor).

Backtracking by Example

▷ **Example 3.2.2.** 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
```

FAU Michael Kohlhase: Artificial Intelligence 1 52 2025-02-06

In general, a Prolog rule of the form A:−B,C reads as *A, if B and C*. If we want to express *A if B or C*, we have to express this two separate rules A:−B and A:−C and leave the choice which one to use to the search procedure.

In ?? we indeed have two clauses for the predicate car/1; one each for the cases of cars with three and four wheels. As the three-wheel case comes first in the program, it is explored first in the search process.

Recall that at every point, where the Prolog interpreter has the choice between two clauses for a predicate, chooses the first and leaves a backtrack point. In ?? this happens first for the predicate car/1, where we explore the case of three-wheeled cars. The Prolog interpreter immediately has to choose again – between the tricycle and the rollerblade, which both have three wheels. Again, it chooses the first and leaves a backtrack point. But as tricycles do not have motors, the subgoal has_motor(mytricycle) fails and the interpreter backtracks to the chronologically nearest backtrack point (the second one) and tries to fulfill has_motor(myrollerblade). This fails again, and the next backtrack point is point 1 – note the stack-like organization of backtrack points which is in keeping with the depth-first search strategy – which chooses the case of four-wheeled cars. This ultimately succeeds as before with y=mybmw.

3.2.3 Programming Features

We now turn to a more classical programming task: computing with numbers. Here we turn to our initial example: adding unary natural numbers. If we can do that, then we have to consider Prolog a programming language.

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 3.2.3.** 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).
```


Note: Viewed through the right glasses logic programming is very similar to functional programming; the only difference is that we are using $n+1$ ary relations rather than n ary function. To see how this works let us consider the addition function/relation example above: instead of a binary function $+$ we program a ternary relation add, where relation add(X,Y,Z) means $X+Y=Z$. We start with the same defining equations for addition, rewriting them to relational style.

The first equation is straight-forward via our correspondence and we get the Prolog fact add(X,zero,X). For the equation $X + s(Y) = s(X + Y)$ we have to work harder, the straight-forward relational translation add(X,s(Y),s(X+Y)) is impossible, since we have only partially replaced the function $+$ with the relation add. Here we take refuge in a very simple trick that we can always do in logic (and mathematics of course): we introduce a new name Z for the offending expression $X + Y$ (using a variable) so that we get the fact add(X,s(Y),s(Z)). Of course this is not universally true (remember that this fact would say that "$X + s(Y) = s(Z)$ for all X, Y, and Z"), so we have to extend it to a Prolog rule add(X,s(Y),s(Z)):−add(X,Y,Z). which relativizes to mean "$X + s(Y) = s(Z)$ for all X, Y, and Z with $X + Y = Z$".

Indeed the rule implements addition as a recursive predicate, we can see that the recursion relation is terminating, since the left hand sides have one more constructor for the successor function. The examples for multiplication and exponentiation can be developed analogously, but we have to use the naming trick twice.

We now apply the same principle of recursive programming with predicates to other examples to reinforce our intuitions about the principles.

More Examples from elementary Arithmetic

▷ **Example 3.2.4.** 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 3.2.5.** 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 3.2.6.** Using Prolog's internal floating-point arithmetic: a goal of the form ?− D **is** e. — where e is a ground arithmetic expression binds D to the result of evaluating e.

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


Note: Note that the **is** relation does not allow "generate and test" inversion as it insists on the right hand being ground. In our example above, this is not a problem, if we call the fib with the first ("input") argument a ground term. Indeed, it matches the last rule with a goal ?− g,Y., where g is a ground term, then g−1 and g−2 are ground and thus D and E are bound to the (ground) result terms. This makes the input arguments in the two recursive calls ground, and we get ground results for Z and W, which allows the last goal to succeed with a ground result for Y. Note as well that re-ordering the bodys literal of the rule so that the recursive calls are called before the computation literals will lead to failure.

We will now add the primitive data structure of lists to Prolog; they are constructed by prepending an element (the head) to an existing list (which becomes the rest list or "tail" of the constructed one).

Adding Lists to Prolog

▷ **Definition 3.2.7.** In Prolog, lists are represented by list terms of the form

1. [a,b,c,. . .] for list literals, and
2. a first/rest constructor that represents a list with head F and rest list R as [F|R].

▷ **Observation:** Just as in functional programming, we can define list operations by recursion, only that we program with relations instead of with functions.

▷ **Example 3.2.8.** Predicates for member, append and reverse of lists in default Prolog representation.

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

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

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


Logic programming is the third large programming paradigm (together with functional programming and imperative programming).

Relational Programming Techniques

▷ **Example 3.2.9.** Parameters have no unique direction "in" or "out"

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

▷ **Example 3.2.10.** Symbolic programming by structural induction:

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

▷ **Example 3.2.11.** Generate and test:

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


From a programming practice point of view it is probably best understood as "relational programming" in analogy to functional programming, with which it shares a focus on recursion.
The major difference to functional programming is that "relational programming" does not have a fixed input/output distinction, which makes the control flow in functional programs very direct and predictable. Thanks to the underlying search procedure, we can sometime make use of the flexibility afforded by logic programming.
If the problem solution involves search (and depth first search is sufficient), we can just get by with specifying the problem and letting the Prolog interpreter do the rest. In **??** we just specify that list Xs can be sorted into Ys, iff Ys is a permutation of Xs and Ys is ordered. Given a concrete (input) list Xs, the Prolog interpreter will generate all permutations of Ys of Xs via the predicate perm/2 and then test them whether they are ordered.

This is a paradigmatic example of logic programming. We can (sometimes) directly use the specification of a problem as a program. This makes the argument for the correctness of the program immediate, but may make the program execution non optimal.

3.2.4 Advanced Relational Programming

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

Specifying Control in Prolog

▷ *Remark 3.2.12.* The running time of the program from **??** 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 3.2.13.* Functions and predicates have radically different roles in Prolog.

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

▷ *Remark 3.2.14.* In functional programming, functions are used for both. (even more confusing than in Prolog if you think about it)

▷ **Example 3.2.15.** Consider again the reverse predicate for lists below:
An input datum is e.g. [1,2,3], then the output datum is [3,2,1].

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

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

▷ **Example 3.2.16 (Trees and Leaf Counting).** We represent (unlabelled) trees via the function t from tree lists to trees. For instance, a balanced binary tree of depth 2 is t([t([t([]),t([])]),t([t([]),t([])])]). We count leaves by

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


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

In this chapter we will briefly recap some of the prerequisites from theoretical computer science that are needed for understanding Artificial Intelligence 1.

4.1 Recap: Complexity Analysis in AI?

We now come to an important topic which is not really part of Artificial Intelligence but which adds an important layer of understanding to this enterprise: We (still) live in the era of Moore's law (the computing power available on a single CPU doubles roughly every two years) leading to an exponential increase. A similar rule holds for main memory and disk storage capacities. And the production of computer (using CPUs and memory) is (still) very rapidly growing as well; giving mankind as a whole, institutions, and individual exponentially grow of computational resources.
In public discussion, this development is often cited as the reason why (strong) AI is inevitable. But the argument is fallacious if all the algorithms we have are of very high complexity (i.e. at least exponential in either time or space). So, to judge the state of play in Artificial Intelligence, we have to know the complexity of our algorithms.
In this section, we will give a very brief recap of some aspects of elementary complexity theory and make a case of why this is a generally important for computer scientists.

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21839` and `https://fau.tv/clip/id/21840`.
To get a feeling what we mean by "fast algorithm", we do some preliminary computations.

Performance and Scaling

- ▷ Suppose we have three algorithms to choose from. (which one to select)
- ▷ Systematic analysis reveals performance characteristics.
- ▷ **Example 4.1.1.** For a computational problem of size n we have

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

The last number in the rightmost column may surprise you. Does the run time really grow that fast? Yes, as a quick calculation shows; and it becomes much worse, as we will see.

What?! One year?

- $2^{10} = 1\,024$ $(1024\mu s \simeq 1ms)$
- $2^{45} = 35\,184\,372\,088\,832$ $(3.5{\times}10^{13}\mu s \simeq 3.5{\times}10^7 s \simeq 1.1Y)$
- **Example 4.1.2.** 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	$1.6min$	$2.5mon$	–

So it does make a difference for larger computational problems what algorithm we choose. Considerations like the one we have shown above are very important when judging an algorithm. These evaluations go by the name of "complexity theory".
Let us now recapitulate some notions of elementary complexity theory: we are interested in the worst-case growth of the resources (time and space) required by an algorithm in terms of the sizes of its arguments. Mathematically we look at the functions from input size to resource size and classify them into "big-O" classes, abstracting from constant factors (which depend on the machine thealgorithm runs on and which we cannot control) and initial (algorithm startup) factors.

Recap: Time/Space Complexity of Algorithms

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

▷ **Definition 4.1.3.** We say that an algorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha) := t$.

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

▷ Time/space complexity depends on size measures. (no canonical one)

▷ **Definition 4.1.4.** The following sets are often used for S in $T(\alpha)$:

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

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

▷ **Lemma 4.1.5 (Growth Ranking).** *For $k' > 2$ and $k > 1$ we have*

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

▷ **For AI-1:** I expect that given an algorithm, you can determine its complexity class. (next)

Advantage: Big-Oh Arithmetics

▷ **Practical Advantage:** Computing with Landau sets is quite simple. (good simplification)

▷ **Theorem 4.1.6 (Computing with Landau Sets).**

1. *If $\mathcal{O}(c \cdot f) = \mathcal{O}(f)$ for any constant $c \in \mathbb{N}$.* (*drop constant factors*)
2. *If $\mathcal{O}(f) \subseteq \mathcal{O}(g)$, then $\mathcal{O}(f + g) = \mathcal{O}(g)$.* (*drop low-complexity summands*)
3. *If $\mathcal{O}(f \cdot g) = \mathcal{O}(f) \cdot \mathcal{O}(g)$.* (*distribute over products*)

▷ These are not all of "big-Oh calculation rules", but they're enough for most purposes

▷ **Applications:** Convince yourselves using the result above that

- ▷ $\mathcal{O}(4n^3 + 3n + 7^{1000n}) = \mathcal{O}(2^n)$
- ▷ $\mathcal{O}(n) \subset \mathcal{O}(n \cdot \log_2(n)) \subset \mathcal{O}(n^2)$

OK, that was the theory, . . . but how do we use that in practice?

What I mean by this is that given an algorithm, we have to determine the time complexity.

This is by no means a trivial enterprise, but we can do it by analyzing the algorithm instruction by instruction as shown below.

Determining the Time/Space Complexity of Algorithms

- **Definition 4.1.7.** Given a function Γ that assigns variables v to functions $\Gamma(v)$ and α an imperative algorithm, we compute the
  - time complexity $T_\Gamma(\alpha)$ of program α and
  - the context $C_\Gamma(\alpha)$ introduced by α

  by joint induction on the structure of α:
  - constant: can be accessed in constant time
    If $\alpha = \delta$ for a data constant δ, then $T_\Gamma(\alpha) \in \mathcal{O}(1)$.
  - variable: need the complexity of the value
    If $\alpha = v$ with $v \in \mathbf{dom}(\Gamma)$, then $T_\Gamma(\alpha) \in \mathcal{O}(\Gamma(v))$.
  - application: compose the complexities of the function and the argument
    If $\alpha = \varphi(\psi)$ with $T_\Gamma(\varphi) \in \mathcal{O}(f)$ and $T_{\Gamma \cup C_\Gamma(\varphi)}(\psi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f \circ g)$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\varphi)}(\psi)$.
  - assignment: has to compute the value $\rightsquigarrow$ has its complexity
    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
    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
    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
    If α is **while**γ**do**φ**end**, with $T_\Gamma(\gamma) \in \mathcal{O}(f)$, $T_{\Gamma \cup C_\Gamma(\gamma)}(\varphi) \in \mathcal{O}(g)$, then $T_\Gamma(\alpha) \in \mathcal{O}(f(n) \cdot g(n))$ and $C_\Gamma(\alpha) = C_{\Gamma \cup C_\Gamma(\gamma)}(\varphi)$.
  - The time complexity $T(\alpha)$ is just $T_\emptyset(\alpha)$, where $\emptyset$ is the empty function.
- Recursion is much more difficult to analyze $\rightsquigarrow$ recurrences and Master's theorem.

As instructions in imperative programs can introduce new variables, which have their own time complexity, we have to carry them around via the introduced context, which has to be defined co-recursively with the time complexity. This makes ?? rather complex. The main two cases to note here are

- the variable case, which "uses" the context Γ and
- the assignment case, which extends the introduced context by the time complexity of the value.

The other cases just pass around the given context and the introduced context systematically. Let us now put one motivation for knowing about complexity theory into the perspective of the job market; here the job as a scientist.

Please excuse the chemistry pictures, public imagery for CS is really just quite boring, this is what people think of when they say "scientist". So, imagine that instead of a chemist in a lab, it's me sitting in front of a computer.

Why Complexity Analysis? (General)

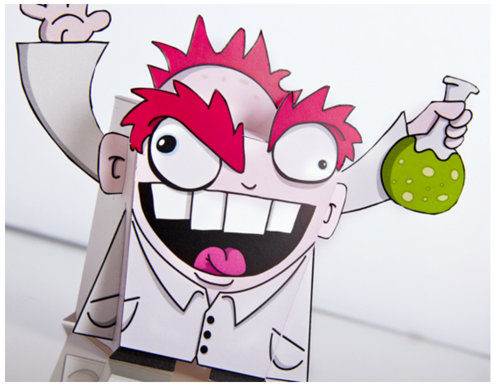
▷ **Example 4.1.8.** Once upon a time I was trying to invent an efficient algorithm.

▹ My first algorithm attempt didn't work, so I had to try harder.

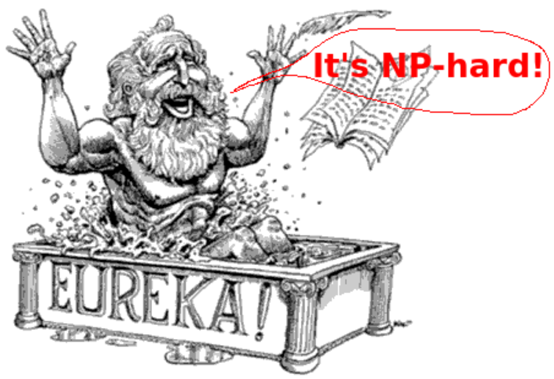
▹ But my 2nd attempt didn't work either, which got me a bit agitated.

▹ The 3rd attempt didn't work either. . .

▹ And neither the 4th. But then:



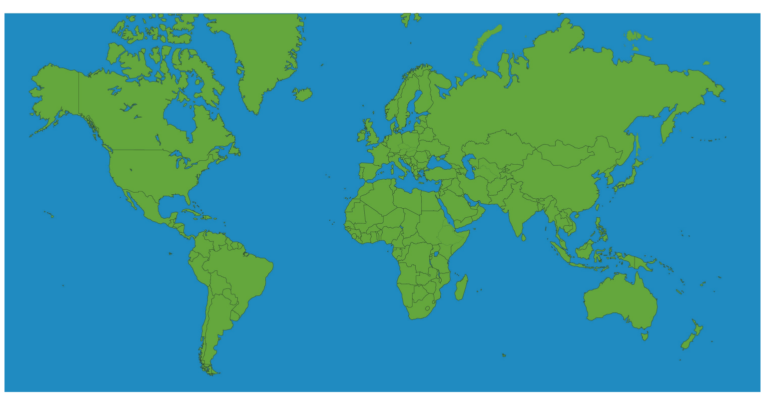
▷ Ta-da ... when, for once, I turned around and looked in the other direction–CAN one actually solve this efficiently? – NP hardness was there to rescue me.



The meat of the story is that there is no profit in trying to invent an algorithm, which we could have known that cannot exist. Here is another image that may be familiar to you.

Why Complexity Analysis? (General)

▷ **Example 4.1.9.** 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.

It's like, you're trying to find a route to India (from Spain), and you presume it's somewhere to the east, and then you hit a coast, but no; try again, but no; try again, but no; ... if you don't have a map, that's the best you can do. But NP hardness gives you the map: you can check that there actually is no way through here. But what is this notion of NP completeness alluded to above? We observe that we can analyze the complexity of problems by the complexity of the algorithms that solve them. This gives us a notion of what to expect from solutions to a given problem class, and thus whether efficient (i.e. polynomial time) algorithms can exist at all.

Reminder (?): NP and PSPACE (details ⇝ 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 $\mathbf{PSPACE} = \mathbf{NPSPACE}$, and hence (trivially) $\mathbf{NP} \subseteq \mathbf{PSPACE}$.

  It is commonly believed that $\mathbf{NP} \not\supseteq \mathbf{PSPACE}$. (similar to $\mathbf{P} \subseteq \mathbf{NP}$)

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?
- ▷ **Answer:** reserved for the plenary sessions ⇝ be there!

4.2 Recap: Formal Languages and Grammars

One of the main ways of designing rational agents in this course will be to define formal languages that represent the state of the agent environment and let the agent use various inference techniques to predict effects of its observations and actions to obtain a world model. In this section we recap the basics of formal languages and grammars that form the basis of a compositional theory for them.

The Mathematics of Strings

▷ **Definition 4.2.1.** 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 4.2.2.** 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 "abc" for $\langle \mathtt{a}, \mathtt{b}, \mathtt{c}\rangle$

▷ **Example 4.2.3.** Take $A = \{\mathtt{h}, \mathtt{1}, /\}$ as an alphabet. Each of the members h, 1, and / is a character. The vector $\langle /, /, \mathtt{1}, \mathtt{h}, \mathtt{1}\rangle$ is a string of length 5 over A.

▷ **Definition 4.2.4 (String Length).** Given a string s we denote its length with $|s|$.

▷ **Definition 4.2.5.** The concatenation $\mathrm{conc}(s,t)$ of two strings $s = \langle s_1, ..., s_n\rangle \in A^n$ and $t = \langle t_1, ..., t_m\rangle \in A^m$ is defined as $\langle s_1, ..., s_n, t_1, ..., t_m\rangle \in A^{n+m}$.

We will often write $\mathrm{conc}(s,t)$ as $s + t$ or simply st

▷ **Example 4.2.6.** conc("text","book") = "text" + "book" = "textbook"

We have multiple notations for concatenation, since it is such a basic operation, which is used so often that we will need very short notations for it, trusting that the reader can disambiguate based on the context.

Now that we have defined the concept of a string as a sequence of characters, we can go on to give ourselves a way to distinguish between good strings (e.g. programs in a given programming language) and bad strings (e.g. such with syntax errors). The way to do this by the concept of a formal language, which we are about to define.

Formal Languages

▷ **Definition 4.2.7.** 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 4.2.8.** 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 4.2.9.** A set $L \subseteq A^*$ is called a formal language over A.

▷ **Definition 4.2.10.** We use $\mathtt{c}^{[n]}$ for the string that consists of the character c repeated n times.

▷ **Example 4.2.11.** $\#^{[5]} = \langle \#, \#, \#, \#, \#\rangle$

▷ **Example 4.2.12.** The set $M := \{\mathtt{ba}^{[n]} \mid n \in \mathbb{N}\}$ of strings that start with character b followed by an arbitrary numbers of a's is a formal language over $A = \{\mathtt{a}, \mathtt{b}\}$.

▷ **Definition 4.2.13.** Let $L_1, L_2, L \subseteq \Sigma^*$ be formal languages over Σ.

- ▹ Intersection and union: $L_1 \cap L_2$, $L_1 \cup L_2$.
- ▹ Language complement L: $\overline{L} := \Sigma^* \backslash L$.
- ▹ The language concatenation of L_1 and L_2: $L_1 L_2 := \{uw \,|\, u \in L_1, w \in L_2\}$. We often use $L_1 L_2$ instead of $L_1 L_2$.
- ▹ Language power L: $L^0 := \{\epsilon\}$, $L^{n+1} := LL^n$, where $L^n := \{\mathtt{w_1} \ldots \mathtt{w_n} \,|\, w_i \in L$, for $i = 1 \ldots n\}$, (for $n \in \mathbb{N}$).
- ▹ language Kleene closure L: $L^* := \bigcup_{n \in \mathbb{N}} L^n$ and also $L^+ := \bigcup_{n \in \mathbb{N}^+} L^n$.
- ▹ The reflection of a language L: $L^R := \{w^R \,|\, w \in L\}$.

There is a common misconception that a formal language is something that is difficult to understand as a concept. This is not true, the only thing a formal language does is separate the "good" from the bad strings. Thus we simply model a formal language as a set of stings: the "good" strings are members, and the "bad" ones are not.

Of course this definition only shifts complexity to the way we construct specific formal languages (where it actually belongs), and we have learned two (simple) ways of constructing them: by repetition of characters, and by concatenation of existing languages. As mentioned above, the purpose of a formal language is to distinguish "good" from "bad" strings. It is maximally general, but not helpful, since it does not support computation and inference. In practice we will be interested in formal languages that have some structure, so that we can represent formal languages in a finite manner (recall that a formal language is a subset of A^*, which may be infinite and even undecidable – even though the alphabet A is finite).

To remedy this, we will now introduce phrase structure grammars (or just grammars), the standard tool for describing structured formal languages.

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 4.2.14.** A phrase structure grammar (also called type 0 grammar, unrestricted grammar, or just grammar) is a tuple $\langle N, \Sigma, P, S\rangle$ where

- ▹ N is a finite set of nonterminal symbols,
- ▹ Σ is a finite set of terminal symbols, members of $\Sigma \cup N$ are called symbols.
- ▹ P is a finite set of production rules: pairs $p := h \rightarrow b$ (also written as $h \Rightarrow b$), where $h \in (\Sigma \cup N)^* N (\Sigma \cup N)^*$ and $b \in (\Sigma \cup N)^*$. The string h is called the head of p and b the body.
- ▹ $S \in N$ is a distinguished symbol called the start symbol (also sentence symbol).

The sets N and Σ are assumed to be disjoint. Any word $w \in \Sigma^*$ is called a terminal word.

▷ **Intuition:** Production rules map strings with at least one nonterminal to arbitrary other strings.

- **Notation:** If we have n rules $h \to b_i$ sharing a head, we often write $h \to b_1 \mid \ldots \mid b_n$ instead.

We fortify our intuition about these – admittedly very abstract – constructions by an example and introduce some more vocabulary.

Phrase Structure Grammars (cont.)

- **Example 4.2.15.** A simple phrase structure grammar G:

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

  Here S, is the start symbol, NP, $Article$, N, and Vi are nonterminals.

- **Definition 4.2.16.** A production rule whose head is a single non-terminal and whose body consists of a single terminal is called lexical or a lexical insertion rule.

  Definition 4.2.17. The subset of lexical rules of a grammar G is called the lexicon of G and the set of body symbols the vocabulary (or alphabet). The nonterminals in their heads are called lexical categories of G.

- **Definition 4.2.18.** The non-lexicon production rules are called structural, and the nonterminals in the heads are called phrasal or syntactic categories.

Now we look at just how a grammar helps in analyzing formal languages. The basic idea is that a grammar accepts a word, iff the start symbol can be rewritten into it using only the rules of the grammar.

Phrase Structure Grammars (Theory)

- **Idea:** Each symbol sequence in a formal language can be analyzed/generated by the grammar.

- **Definition 4.2.19.** Given a phrase structure grammar $G := \langle N, \Sigma, P, S\rangle$, we say G derives $t \in (\Sigma \cup N)^*$ from $s \in (\Sigma \cup N)^*$ in one step, iff there is a production rule $p \in P$ with $p = h \to b$ and there are $u, v \in (\Sigma \cup N)^*$, such that $s = suhv$ and $t = ubv$. We write $s \to_G^p t$ (or $s \to_G t$ if p is clear from the context) and use $\to_G^*$ for the reflexive transitive closure of $\to_G$. We call $s \to^*{}_G t$ a G derivation of t from s.

  TEST1: $\begin{array}{ccc} A & \to_G & B \\ C & \to_G & D \end{array}$

TEST2: $A \to_G B \to_G C \to_G D$ TEST3: $s \to_{G_2} asb \to_{G_2} aaSbb \to_{G_2} aaaSbbb \to_{G_2} aaaaSbbbb \to_{G_2} aaaabbbb$

▷ **Definition 4.2.20.** Given a phrase structure grammar $G := \langle N, \Sigma, P, S\rangle$, we say that $s \in (N \cup \Sigma)^*$ is a sentential form of G, iff $S \to^*_G s$. A sentential form that does not contain nontermials is called a sentence of G, we also say that G accepts s. We say that G rejects s, iff it is not a sentence of G.

▷ **Definition 4.2.21.** The language $\mathbf{L}(G)$ of G is the set of its sentences. We say that $\mathbf{L}(G)$ is generated by G.

Definition 4.2.22. We call two grammars equivalent, iff they have the same languages.

Definition 4.2.23. A grammar G is said to be universal if $\mathbf{L}(G) = \Sigma^*$.

▷ **Definition 4.2.24.** Parsing, syntax analysis, or syntactic analysis is the process of analyzing a string of symbols, either in a formal or a natural language by means of a grammar.

Again, we fortify our intuitions with ??.

Phrase Structure Grammars (Example)

▷ **Example 4.2.25.** In the grammar G from ??:

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

$$\begin{aligned} S &\to_G NP\ Vi \\ &\to_G Article\ N\ Vi \\ &\to_G Article\ \textbf{teacher}\ Vi \end{aligned}$$

2. *The teacher sleeps* is a sentence.

$$\begin{aligned} S &\to^*_G Article\ \textbf{teacher}\ Vi \\ &\to_G \textbf{the teacher}\ Vi \\ &\to_G \textbf{the teacher sleeps} \end{aligned}$$

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

Note that this process indeed defines a formal language given a grammar, but does not provide an efficient algorithm for parsing, even for the simpler kinds of grammars we introduce below.

Grammar Types (Chomsky Hierarchy [Cho65])

▷ **Observation:** The shape of the grammar determines the "size" of its language.

▷ **Definition 4.2.26.** We call a grammar:

1. context-sensitive (or type 1), if the bodies of production rules have no less symbols than the heads,
2. context-free (or type 2), if the heads have exactly one symbol,
3. regular (or type 3), if additionally the bodies are empty or consist of a nonterminal, optionally followed by a terminal symbol.

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

▷ **Example 4.2.27 (Context-sensitive).** The language $\{a^{[n]}b^{[n]}c^{[n]}\}$ is accepted by

$$\begin{aligned} S &\rightarrow \mathbf{a}\ \mathbf{b}\ \mathbf{c} \mid A \\ A &\rightarrow \mathbf{a}\ A\ B\ \mathbf{c} \mid \mathbf{a}\ \mathbf{b}\ \mathbf{c} \\ \mathbf{c}\ B &\rightarrow B\ \mathbf{c} \\ \mathbf{b}\ B &\rightarrow \mathbf{b}\ \mathbf{b} \end{aligned}$$

▷ **Example 4.2.28 (Context-free).** The language $\{a^{[n]}b^{[n]}\}$ is accepted by $S \rightarrow \mathbf{a}\ S\ \mathbf{b} \mid \epsilon$.

▷ **Example 4.2.29 (Regular).** The language $\{a^{[n]}\}$ is accepted by $S \rightarrow S\ \mathbf{a}$

▷ **Observation:** Natural languages are probably context-sensitive but parsable in real time! (like languages low in the hierarchy)

While the presentation of grammars from above is sufficient in theory, in practice the various grammar rules are difficult and inconvenient to write down. Therefore computer science – where grammars are important to e.g. specify parts of compilers – has developed extensions – notations that can be expressed in terms of the original grammar rules – that make grammars more readable (and writable) for humans. We introduce an important set now.

Useful Extensions of Phrase Structure Grammars

▷ **Definition 4.2.30.** The Bachus Naur form or Backus normal form (BNF) is a metasyntax notation for context-free grammars.

It extends the body of a production rule by mutiple (admissible) constructors:

- ▷ alternative: $s_1 \mid \ldots \mid s_n$,
- ▷ repetition: s^* (arbitrary many s) and s^+ (at least one s),
- ▷ optional: $[s]$ (zero or one times),
- ▷ grouping: $(s_1 ; \ldots ; s_n)$, useful e.g. for repetition,
- ▷ character sets: $[s{-}t]$ (all characters c with $s \leq c \leq t$ for a given ordering on the characters), and
- ▷ complements: $[\hat{}\, s_1, \ldots, s_n]$, provided that the base alphabet is finite.

- **Observation:** All of these can be eliminated, .e.g (⇝ many more rules)
  - replace $X \to Z\ (s^*)\ W$ with the production rules $X \to Z\ Y\ W$, $Y \to \epsilon$, and $Y \to Y\ s$.
  - replace $X \to Z\ (s^+)\ W$ with the production rules $X \to Z\ Y\ W$, $Y \to s$, and $Y \to Y\ s$.

We will now build on the notion of BNF grammar notations and introduce a way of writing down the (short) grammars we need in AI-1 that gives us even more of an overview over what is happening.

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

We will generally get by with context-free grammars, which have highly efficient into parsing algorithms, for the formal language we use in this course, but we will not cover the algorithms in AI-1.

4.3 Mathematical Language Recap

We already clarified above that we will use mathematical language as the main vehicle for specifying the concepts underlying the AI algorithms in this course.

In this section, we will recap (or introduce if necessary) an important conceptual practice of modern mathematics: the use of mathematical structures.

Mathematical Structures

- **Observation:** Mathematicians often cast classes of complex objects as mathematical structures.

- We have just seen an example of a mathematical structure: (repeated here for convenience)
- **Definition 4.3.1.** A phrase structure grammar (also called type 0 grammar, unrestricted grammar, or just grammar) is a tuple $\langle N, \Sigma, P, S\rangle$ where
  - N is a finite set of nonterminal symbols,
  - Σ is a finite set of terminal symbols, members of $\Sigma \cup N$ are called symbols.
  - P is a finite set of production rules: pairs $p := h \rightarrow b$ (also written as $h \Rightarrow b$), where $h \in (\Sigma \cup N)^* N (\Sigma \cup N)^*$ and $b \in (\Sigma \cup N)^*$. The string h is called the head of p and b the body.
  - $S \in N$ is a distinguished symbol called the start symbol (also sentence symbol).

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

Note that the idea of mathematical structures has been picked up by most programming languages in various ways and you should therefore be quite familiar with it once you realize the parallelism.

Mathematical Structures in Programming

- **Observation:** Most programming languages have some way of creating "named structures". Referencing components is usually done via "dot notation".
- **Example 4.3.2 (Structs in C).** C data structures for representing grammars:

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

- **Example 4.3.3 (Classes in OOP).** Classes in object-oriented programming languages are based on the same ideas as mathematical structures, only that OOP adds powerful inheritance mechanisms.

Even if the idea of mathematical structures may be familiar from programming, it may be quite intimidating to some students in the mathematical notation we will use in this course. Therefore will – when we get around to it – use a special overview notation in AI-1. We introduce it below.

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

- In AI-1 we use mathematical notation, ...
- **Definition 4.3.4.** A structure signature combines the components, their "types", and accessor names of a mathematical structure in a tabular overview.
- **Example 4.3.5.**

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

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

  Read the first line "N **Set** nonterminal symbols" in the structure above as "N is in an (unspecified) set and is a nonterminal symbol".

  Here – and in the future – we will use **Set** for the class of sets ⇝ "N is a set".
- I will try to give structure signatures where necessary.

Chapter 5

Rational Agents: a Unifying Framework for Artificial Intelligence

In this chapter, we introduce a framework that gives a comprehensive conceptual model for the multitude of methods and algorithms we cover in this course. The framework of rational agents accommodates two traditions of AI.

Initially, the focus of AI research was on symbolic methods concentrating on the mental processes of problem solving, starting from Newell/Simon's "physical symbol hypothesis":

> *A physical symbol system has the necessary and sufficient means for general intelligent action.* [NS76]

Here a symbol is a representation an idea, object, or relationship that is physically manifested in (the brain of) an intelligent agent (human or artificial).

Later – in the 1980s – the proponents of embodied AI posited that most features of cognition, whether human or otherwise, are shaped – or at least critically influenced – by aspects of the entire body of the organism. The aspects of the body include the motor system, the perceptual system, bodily interactions with the environment (situatedness) and the assumptions about the world that are built into the structure of the organism. They argue that symbols are not always necessary since

> *The world is its own best model. It is always exactly up to date. It always has every detail there is to be known. The trick is to sense it appropriately and often enough.* [Bro90]

The framework of rational agents initially introduced by Russell and Wefald in [RW91] – accommodates both, it situates agents with percepts and actions in an environment, but does not preclude physical symbol systems – i.e. systems that manipulate symbols as agent functions. Russell and Norvig make it the central metaphor of their book "Artificial Intelligence – A modern approach" [RN03], which we follow in this course.

5.1 Introduction: Rationality in Artificial Intelligence

We now introduce the notion of rational agents as entities in the world that act optimally (given the available information). We situate rational agents in the scientific landscape by looking at variations of the concept that lead to slightly different fields of study.

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)

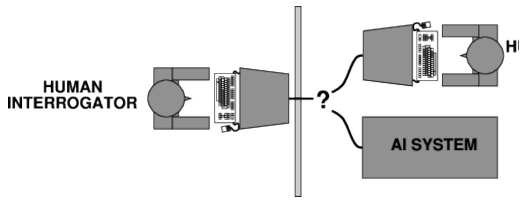
We now discuss all of the four facets in a bit more detail, as they all either contribute directly to our discussion of AI methods or characterize neighboring disciplines.

Acting humanly: The Turing test

▷ Introduced by Alan Turing (1950) "Computing machinery and intelligence" [Tur50]:

▷ "Can machines think?" ⟶ "Can machines behave intelligently?"

▷ **Definition 5.1.1.** 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 5.1.2.** 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 5.1.3.** 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 5.1.4.** 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 5.1.5.** 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

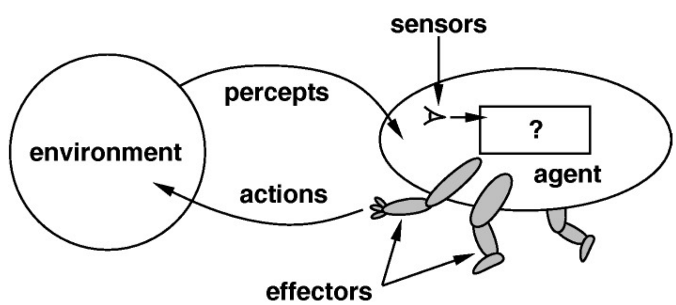
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21843`.
Given the discussion in the previous section, especially the ideas that "behaving rationally" could be a suitable – since operational – goal for AI research, we build this into the paradigm "rational agents" introduced by Stuart Russell and Eric H. Wefald in [RW91].

Agents and Environments

▷ **Definition 5.2.1.** 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).

Definition 5.2.2. Any recognizable, coherent employment of the actuators of an agent is called an action.



▷ **Example 5.2.3.** Agents include humans, robots, softbots, thermostats, etc.

▷ **remark:** The notion of an agent and its environment is intentionally designed to be inclusive. We will classify and discuss subclasses of both later

One possible objection to this is that the agent and the environment are conceptualized as separate entities; in particular, that the image suggests that the agent itself is not part of the environment. Indeed that is intended, since it makes thinking about agents and environments easier and is of little consequence in practice. In particular, the offending separation is relatively easily fixed if needed.
Let us now try to express the agent/environment ideas introduced above in mathematical language to add the precision we need to start the process towards the implementation of rational agents.

Modeling Agents Mathematically and Computationally

▷ **Definition 5.2.4.** A percept is the perceptual input of an agent at a specific time instant.

▷ **Definition 5.2.5.** Any recognizable, coherent employment of the actuators of an agent is called an action.

▷ **Definition 5.2.6.** The agent function f_a of an agent a maps from percept histories to actions:

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

▷ We assume that agents can always perceive their own actions. (but not necessarily their consequences)

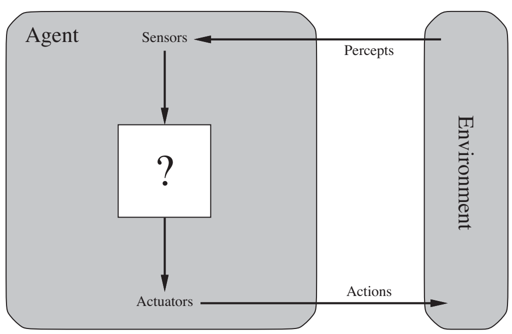
▷ **Problem:** Agent functions can become very big and may be uncomputable. (theoretical tool only)

▷ **Definition 5.2.7.** An agent function can be implemented by an agent program that runs on a (physical or hypothetical) agent architecture.

Here we already see a problem that will recur often in this course: The mathematical formulation gives us an abstract specification of what we want (here the agent function), but not directly a way of how to obtain it. Here, the solution is to choose a computational model for agents (an agent architecture) and see how the agent function can be implemented in a agent program.

Agent Schema: Visualizing the Internal Agent Structure

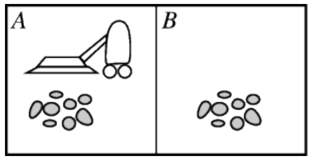
▷ **Agent Schema:** We will use the following kind of agent schema to visualize the internal structure of an agent:



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

Let us fortify our intuition about all of this with an example, which we will use often in the course of the AI-1 course.

Example: Vacuum-Cleaner World and Agent



▷ percepts: location and contents, e.g., $[A, Dirty]$

▷ actions: $Left$, $Right$, $Suck$, $NoOp$

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean]$, $[A, Clean]$	$Right$
$[A, Clean]$, $[A, Dirty]$	$Suck$
$[A, Clean]$, $[B, Clean]$	$Left$
$[A, Clean]$, $[B, Dirty]$	$Suck$
$[A, Dirty]$, $[A, Clean]$	$Right$
$[A, Dirty]$, $[A, Dirty]$	$Suck$
⋮	⋮
$[A, Clean]$, $[A, Clean]$, $[A, Clean]$	$Right$
$[A, Clean]$, $[A, Clean]$, $[A, Dirty]$	$Suck$
⋮	⋮

▷ **Science Question:** What is the *right* agent function?

▷ **AI Question:** Is there an agent architecture and agent program that implements it.

The first implementation idea inspired by the table in last slide would just be table lookup algorithm.

Table-Driven Agents

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

▷ We can directly implement this:

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

▷ **Problem:** Why is this not a good idea?

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

Example: Vacuum-Cleaner Agent Program

▷ A much better implementation idea is to trigger actions from specific percepts.

▷ **Example 5.2.8 (Agent Program).**

```
procedure Reflex−Vacuum−Agent [location,status] returns an action
```

if $status$ = Dirty **then return** Suck
else if $location$ = A **then return** Right
else if $location$ = B **then return** Left

- This is the kind of agent programs we will be looking for in AI-1.

5.3 Good Behavior ⇝ Rationality

Now we try understand the mathematics of rational behavior in our quest to make the rational agents paradigm implementable and take steps for realizing AI. **A Video Nugget** covering this section can be found at `https://fau.tv/clip/id/21844`.

Rationality

- **Idea:** Try to design agents that are successful! (aka. "do the right thing")
- **Problem:** What do we mean by "successful", how do we measure "success"?
- **Definition 5.3.1.** A performance measure is a function that evaluates a sequence of environments.
- **Example 5.3.2.** A performance measure for a vacuum cleaner could
  - award one point per "square" cleaned up in time T?
  - award one point per clean "square" per time step, minus one per move?
  - penalize for $> k$ dirty squares?
- **Definition 5.3.3.** An agent is called rational, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date.
- **Critical Observation:** We only need to maximize the expected value, not the actual value of the performance measure!
- **Question:** Why is rationality a good quality to aim for?

Let us see how the observation that we only need to maximize the expected value, not the actual value of the performance measure affects the consequences.

Consequences of Rationality: Exploration, Learning, Autonomy

- **Note:** A rational agent need not be perfect:
  - It only needs to maximize expected value (rational ≠ omniscient)
    - need not predict e.g. very unlikely but catastrophic events in the future
  - Percepts may not supply all relevant information (rational ≠ clairvoyant)
    - if we cannot perceive things we do not need to react to them.
    - but we may need to try to find out about hidden dangers (exploration)

- Action outcomes may not be as expected (rational $\neq$ successful)
  - but we may need to take action to ensure that they do (more often) (learning)
- **Note:** Rationality may entail exploration, learning, autonomy (depending on the environment / task)
- **Definition 5.3.4.** An agent is called autonomous, if it does not rely on the prior knowledge about the environment of the designer.
- Autonomy avoids fixed behaviors that can become unsuccessful in a changing environment. (anything else would be irrational)
- The agent may have to learn all relevant traits, invariants, properties of the environment and actions.

For the design of agent for a specific task – i.e. choose an agent architecture and design an agent program, we have to take into account the performance measure, the environment, and the characteristics of the agent itself; in particular its actions and sensors.

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 5.3.5.** 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 5.3.6 (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)

The PEAS criteria are essentially a laundry list of what an agent design task description should include.

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.

▷ **Answer:** reserved for the plenary sessions ⇝ be there!

5.4 Classifying Environments

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21869`.
It is important to understand that the kind of the environment has a very profound effect on the agent design. Depending on the kind, different kinds of agents are needed to be successful. So before we discuss common kind of agents in **??**, we will classify kinds environments.

Environment types

▷ **Observation 5.4.1.** *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 5.4.2.** For an agent a we classify the environment e of a by its type, which is one of the following. We call e

1. fully observable, iff the a's sensors give it access to the complete state of the environment at any point in time, else partially observable.

2. deterministic, iff the next state of the environment is completely determined by the current state and a's action, else stochastic.
3. episodic, iff a's experience is divided into atomic episodes, where it perceives and then performs a single action. Crucially, the next episode does not depend on previous ones. Non-episodic environments are called sequential.
4. dynamic, iff the environment can change without an action performed by a, else static. If the environment does not change but a's performance measure does, we call e semidynamic.
5. discrete, iff the sets of e's state and a's actions are countable, else continuous.
6. single-agent, iff only a acts on e; else multi-agent (when must we count parts of e as agents?)

Some examples will help us understand the classification of environments better.

Environment Types (Examples)

▷ **Example 5.4.3.** 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

▷ **Note:** Take the example above with a grain of salt. There are often multiple interpretations that yield different classifications and different agents. (agent designer's choice)

▷ **Example 5.4.4.** Seen as a multi-agent game, chess is deterministic, as a single-agent game, it is stochastic.

▷ **Observation 5.4.5.** *The real world is (of course) a partially observable, stochastic, sequential, dynamic, continuous, and multi-agent environment.* (*worst case for AI*)

▷ **Preview:** We will concentrate on the "easy" environment types (fully observable, deterministic, episodic, static, and single-agent) in AI-1 and extend them to "realworld"-compatible ones in AI-2.

In the AI-1 course we will work our way from the simpler environment types to the more general ones. Each environment type wil need its own agent types specialized to surviving and doing well in them.

5.5 Types of Agents

We will now discuss the main types of agents we will encounter in this course, get an impression of the variety, and what they can and cannot do. We will start from simple reflex agents, add state, and utility, and finally add learning. **A Video Nugget** covering this section can be found at `https://fau.tv/clip/id/21926`.

Agent Types

▷ **Observation:** So far we have described (and analyzed) agents only by their behavior (cf. agent function $f\colon \mathcal{P}^* \to \mathcal{A}$).

▷ **Problem:** This does not help us to build agents. (the goal of AI)

▷ To build an agent, we need to fix an agent architecture and come up with an agent program that runs on it.

▷ **Preview:** Four basic types of agent architectures in order of increasing generality:

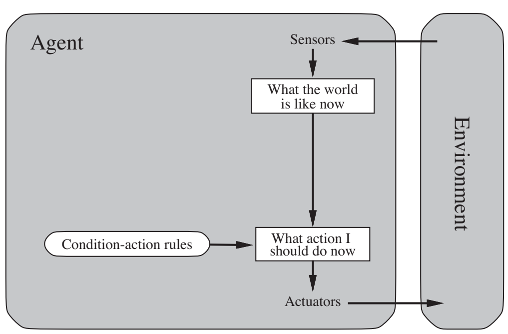
1. simple reflex agents
2. model-based agents
3. goal-based agents
4. utility-based agents

All these can be turned into learning agents.

Simple reflex agents

▷ **Definition 5.5.1.** 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 5.5.2 (Agent Program).**

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

Simple reflex agents (continued)

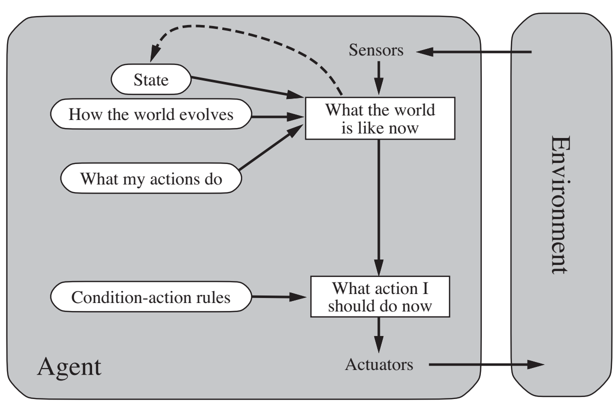
- **General Agent Program:**

```
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 5.5.3.** 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 5.5.4.** Vacuum cleaner robot with defective location sensor ⇝ 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 5.5.5.** 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 $\mathcal{T}$, that predicts a new state $\mathcal{T}(s, a)$ 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 $s' = \mathcal{T}(S(p, s), a)$ and take action $a' = f(s')$.

▷ **Note:** As different percept sequences lead to different states, so the agent function $f_a : \mathcal{P}^* \to \mathcal{A}$ no longer depends only on the last percept.

▷ **Example 5.5.6 (Tail Lights Again).** Model-based agents can do the ?? if the states include a concept of tail light brightness.

Model-Based Agents (continued)

▷ **Observation 5.5.7.** *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).

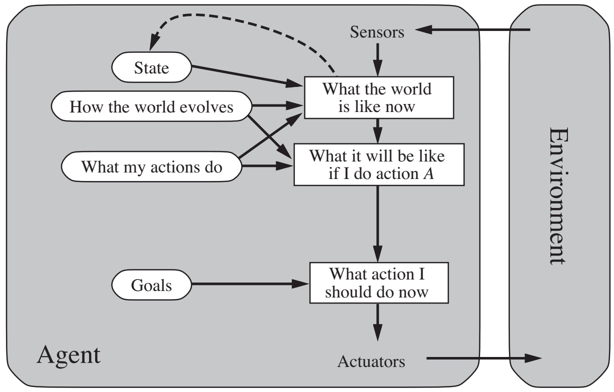
▷ **Example 5.5.8.** Coming to an intersection, where the agent has to decide between going left and right.

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 5.5.9.** A goal-based agent is a model-based agent with transition model T that deliberates actions based on 3 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 $s \mapsto f(T(s), \mathcal{G})$.

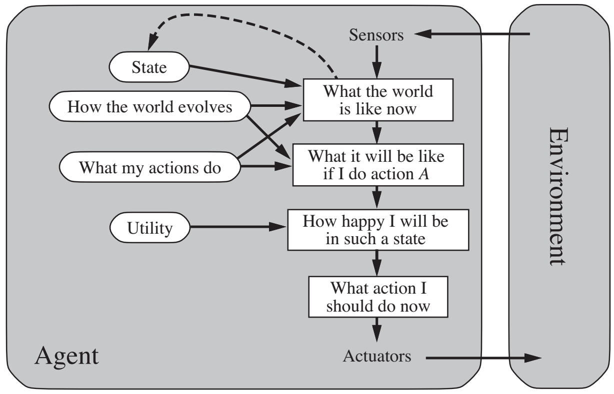
▷ **Observation:** A goal-based agent is more flexible in the knowledge it can utilize.

▷ **Example 5.5.10.** 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 5.5.11.** 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

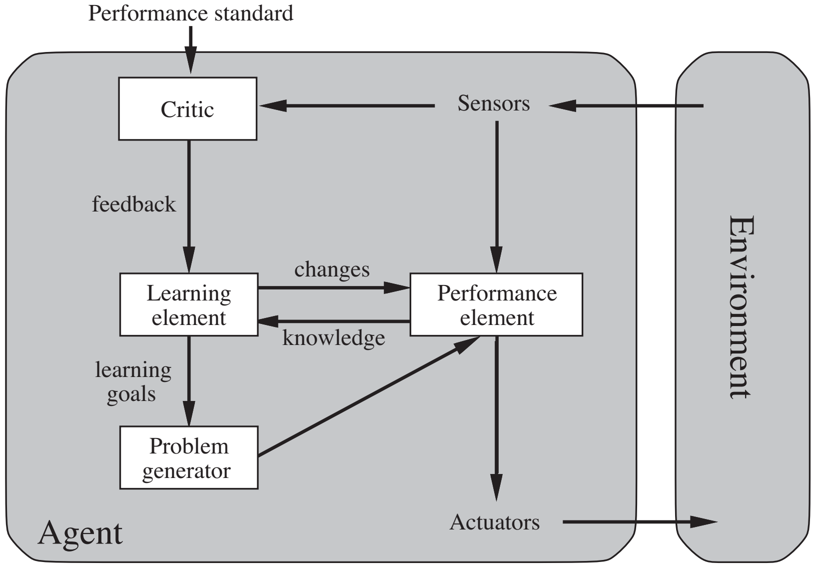
- **Question:** What is the difference between goal-based and utility-based agents?
- **Utility-based Agents are a Generalization:** We can always force goal-directedness by a utility function that only rewards goal states.
- **Goal-based Agents can do less:** A utility function allows rational decisions 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 5.5.12.** 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 5.5.13 (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

We now come to a very important topic, which has a great influence on agent design: how does the agent represent the environment. After all, in all agent designs above (except the simple reflex agent) maintain a notion of world state and how the world state evolves given percepts and actions. The form of this model crucially influences the algorithms we can build. **A Video Nugget** covering this section can be found at `https://fau.tv/clip/id/21925`.

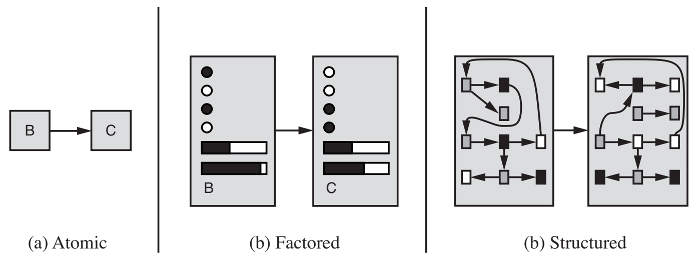
Representing the Environment in Agents

- We have seen various components of agents that answer questions like
  - *What is the world like now?*
  - *What action should I do now?*
  - *What do my actions do?*
- **Next natural question:** How do these work? (see the rest of the course)
- **Important Distinction:** How the agent implements the world model.
- **Definition 5.6.1.** 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, their properties and relationships.
- **Intuition:** From atomic to structured, the representations agent designer more flexibility and the algorithms more components to process.
- **Also** The additional internal structure will make the algorithms more complex.

Again, we fortify our intuitions with a an illustration and an example.

Atomic/Factored/Structured State Representations

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



▷ **Example 5.6.2.** 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.
- ▷ In a factored representation we may have attributes "gps-location", "gas",... (allows information sharing between states and uncertainty)
- ▷ But how to represent a situation, where a large truck blocking the road, since it is trying to back into a driveway, but a loose cow is blocking its path. (attribute "TruckAheadBackingIntoDairyFarmDrivewayBlockedByLooseCow" is unlikely)
- ▷ In a structured representation, we can have objects for trucks, cows, etc. and their relationships. (at "run-time")

Note: The set of states in atomic representations and attributes in factored ones is determined at design time, while the objects and their relationships in structured ones are discovered at "runtime".
Here – as always when we evaluate representations – the crucial aspect to look out for are the idendity conditions: when do we consider two representations equal, and when can we (or more crucially algorithms) distinguish them.

For instance for factored representations, make world representations equal, iff the values of the attributes – that are determined at agent design time and thus immutable by the agent – are all equual. So the agent designer has to make sure to add all the attributes to the chosen representation that are necessary to distinguish environments that the agent program needs to treat differently.

It is tempting to think that the situation with atomic representations is easier, since we can "simply" add enough states for the necesssary distictions, but in practice this set of states may have to be infinite, while in factored or structured representations we can keep representations finite.

5.7 Rational Agents: Summary

Summary

▷ Agents interact with environments through actuators and sensors.

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

Corollary: We are Agent Designers!

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

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

Part II

General Problem Solving

This part introduces search-based methods for general problem solving using atomic and factored representations of states.

Concretely, we discuss the basic techniques of search-based symbolic AI. First in the shape of classical and heuristic search and adversarial search paradigms. Then in constraint propagation, where we see the first instances of inference-based methods.

Chapter 6

Problem Solving and Search

In this chapter, we will look at a class of algorithms called search algorithms. These are algorithms that help in quite general situations, where there is a precisely described problem, that needs to be solved. Hence the name "General Problem Solving" for the area.

6.1 Problem Solving

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/21927`.

Before we come to the search algorithms themselves, we need to get a grip on the types of problems themselves and how we can represent them, and on what the various types entail for the problem solving process.

The first step is to classify the problem solving process by the amount of knowledge we have available. It makes a difference, whether we know all the factors involved in the problem before we actually are in the situation. In this case, we can solve the problem in the abstract, i.e. make a plan before we actually enter the situation (i.e. offline), and then when the problem arises, only execute the plan. If we do not have complete knowledge, then we can only make partial plans, and have to be in the situation to obtain new knowledge (e.g. by observing the effects of our actions or the actions of others). As this is much more difficult we will restrict ourselves to offline 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 6.1.1.** In offline problem solving an agent computing an action sequence based complete knowledge of the environment.

▷ *Remark 6.1.2.* Offline problem solving only works in fully observable, deterministic, static, and episodic environments.

▷ **Definition 6.1.3.** In online problem solving an agent computes one action at a time based on incoming perceptions.

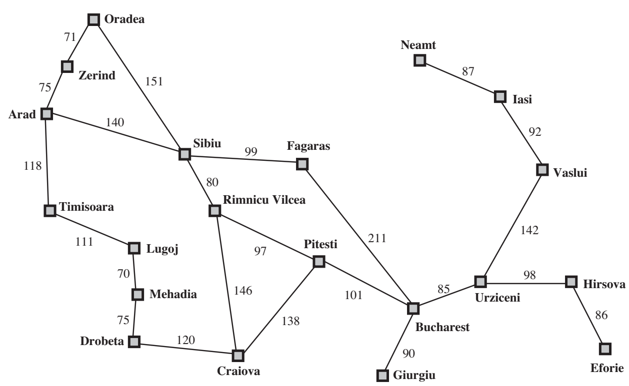
▷ **This Semester:** We largely restrict ourselves to offline problem solving. (easier)

FAU Michael Kohlhase: Artificial Intelligence 1 119 2025-02-06

We will use the following problem as a running example. It is simple enough to fit on one slide and complex enough to show the relevant features of the problem solving algorithms we want to talk about.

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

FAU Michael Kohlhase: Artificial Intelligence 1 120 2025-02-06

Given this example to fortify our intuitions, we can now turn to the formal definition of problem formulation and their solutions.

Problem Formulation

▷ **Definition 6.1.4.** 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 6.1.5.** A search problem $\Pi := \langle\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G}\rangle$ consists of a set $\mathcal{S}$ of states, a set $\mathcal{A}$ of actions, and a transition model $\mathcal{T}\colon \mathcal{A}\times\mathcal{S}\to\mathcal{P}(\mathcal{S})$ that assigns to any action $a \in \mathcal{A}$ and state $s \in \mathcal{S}$ a set of successor states.

Certain states in $\mathcal{S}$ are designated as goal states (also called terminal state) ($\mathcal{G} \subseteq \mathcal{S}$ with $\mathcal{G} \neq \emptyset$) and initial states $\mathcal{I} \subseteq \mathcal{S}$.

▷ **Definition 6.1.6.** We say that an action $a \in \mathcal{A}$ is applicable in state $s \in \mathcal{S}$, iff $\mathcal{T}(a, s) \neq \emptyset$ and that any $s' \in \mathcal{T}(a, s)$ is a result of applying action a to state s.

We call $\mathcal{T}_a\colon \mathcal{S} \to \mathcal{P}(\mathcal{S})$ with $\mathcal{T}_a(s) := \mathcal{T}(a, s)$ the result relation for a and $\mathcal{T}_\mathcal{A} := \bigcup_{a\in\mathcal{A}}\mathcal{T}_a$ the result relation of Π.

▷ **Definition 6.1.7.** The graph $\langle\mathcal{S}, \mathcal{T}_\mathcal{A}\rangle$ is called the state space induced by Π.

▷ **Definition 6.1.8.** A solution for Π consists of a sequence $a_1, \ldots, a_n$ of actions such that for all $1 < i \leq n$

  ▷ a_i is applicable to state s_{i-1}, where $s_0 \in \mathcal{I}$ and

  ▷ $s_i \in \mathcal{T}_{a_i}(s_{i-1})$, and $s_n \in \mathcal{G}$.

▷ **Idea:** A solution bring us from $\mathcal{I}$ to a goal state via applicable actions.

▷ **Definition 6.1.9.** 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.

Observation: The formulation of problems from **??** uses an atomic (black-box) state representation. It has enough functionality to construct the state space but nothing else. We will come back to this in slide **??**.

Remark 6.1.10. Note that search problems formalize problem formulations by making many of the implicit constraints explicit.

Structure Overview: Search Problem

▷ The structure overview for search problems:

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

We will now specialize **??** to deterministic, fully observable environments, i.e. environments where actions only have one – assured – outcome state.

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 6.1.11. We call a search problem with transition model $\mathcal{T}$ deterministic, iff $|\mathcal{T}(a, s)| \leq 1$.

▷

▷ **Definition 6.1.12.** In a deterministic search problem, $\mathcal{T}_a$ induces partial function $\mathcal{S}_a : \mathcal{S} \rightharpoonup \mathcal{S}$ whose natural domain is the set of states where a is applicable: $\mathcal{S}_a(s) := s'$ if $\mathcal{T}_a = \{s'\}$ and undefined at s otherwise. We call $\mathcal{S}_a$ the successor function for a and $\mathcal{S}_a(s)$ the successor state of s.

▷ **Definition 6.1.13.** The predicate that tests for goal states is called a goal test.

6.2 Problem Types

Note that the definition of a search problem is very general, it applies to many many real-world problems. So we will try to characterize these by difficulty. **A Video Nugget** covering this section can be found at `https://fau.tv/clip/id/21928`.

Problem types

▷ **Definition 6.2.1.** 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 6.2.2.** A search problem is called a multi state problem

- ▷ states partially observable (e.g. multiple initial states)
- ▷ deterministic, static, discrete

▷ **Definition 6.2.3.** 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)

We will explain these problem types with another example. The problem $\mathcal{P}$ is very simple: We have a vacuum cleaner and two rooms. The vacuum cleaner is in one room at a time. The floor can be dirty or clean.

The possible states are determined by the position of the vacuum cleaner and the information, whether each room is dirty or not. Obviously, there are eight states: $\mathcal{S} = \{1, 2, 3, 4, 5, 6, 7, 8\}$ for simplicity.

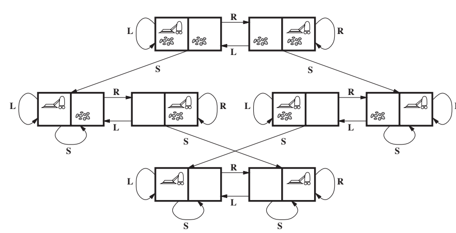
The goal is to have both rooms clean, the vacuum cleaner can be anywhere. So the set $\mathcal{G}$ of goal states is $\{7, 8\}$. In the single-state version of the problem, $[right, suck]$ shortest solution, but $[suck, right, suck]$ is also one. In the multiple-state version we have

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

Example: vacuum-cleaner world

▷ **Single-state Problem:**

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



▷ **Multiple-state Problem:**

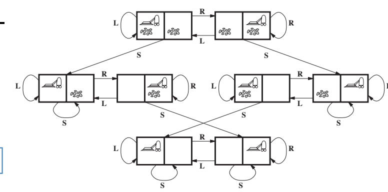
- ▷ Start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
- ▷ **Solution**: $[right, suck, left, suck]$

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

Example: Vacuum-Cleaner World (continued)

▷ **Contingency Problem:**

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



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

In the contingency version of $\mathcal{P}$ a solution is the following:

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

etc. Of course, local sensing can help: narrow $\{6,8\}$ to $\{6\}$ or $\{8\}$, if we are in the first, then suck.

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.

"Path cost": There may be more than one solution and we might want to have the "best" one in a certain sense.

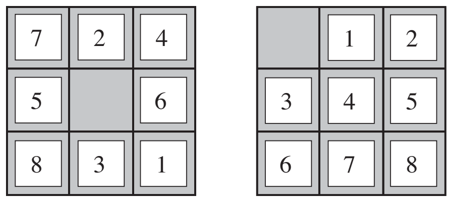
Selecting a state space

- **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.

"State": e.g., we don't care about tourist attractions found in the cities along the way. But this is problem dependent. In a different problem it may well be appropriate to include such information in the notion of state.

"Realizability": one could also say that the abstraction must be sound wrt. reality.

Example: The 8-puzzle



Start State Goal State

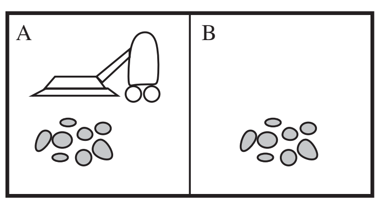
States? Actions?...

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

How many states are there? N factorial, so it is not obvious that the problem is in **NP**. One needs to show, for example, that polynomial length solutions do always exist. Can be done by combinatorial arguments on state space graph (really ?).

Some rule-books give a different goal state for the 8-puzzle: starting with 1, 2, 3 in the top row and having the hold in the lower right corner. This is completely irrelevant for the example and its significance to AI-1.

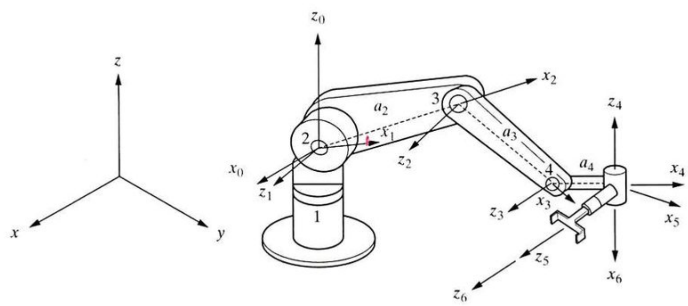
Example: Vacuum-cleaner



States? Actions?...

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? Actions?...

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.

▷ **Answer:** reserved for the plenary sessions ⇝ be there!

6.3 Search

A Video Nugget covering this section can be found at https://fau.tv/clip/id/21956.

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)

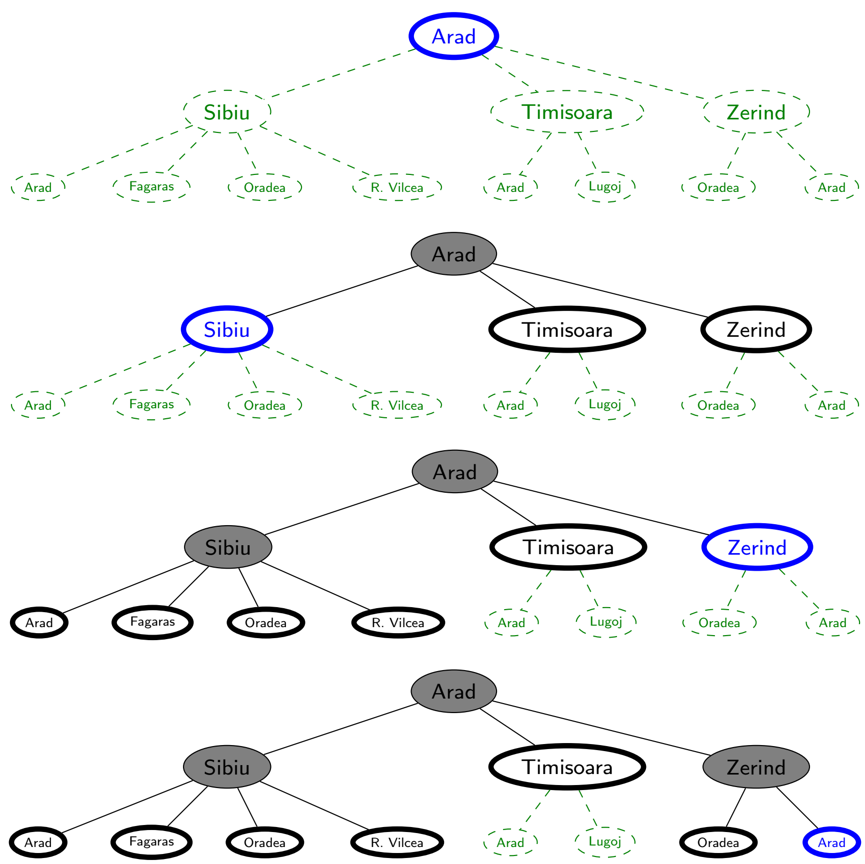
▷ **Definition 6.3.1.** Given a search problem $\mathcal{P} := \langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, the tree search algorithm consists of the simulated exploration of state space $\langle\mathcal{S},\mathcal{T}_\mathcal{A}\rangle$ in a search tree formed by successively 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

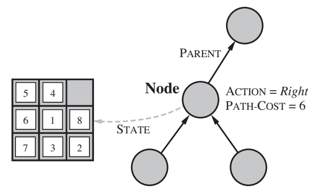


Let us now think a bit more about the implementation of tree search algorithms based on the ideas discussed above. The abstract, mathematical notions of a search problem and the induced tree search algorithm gets further refined here.

Implementation: States vs. nodes

- **Recap:** A state is a (representation of) a physical configuration.
- **Definition 6.3.2 (Implementing a Search Tree).**

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



- **Observation:** A set of search tree nodes that can all (recursively) reach a single initial node form a search tree. (they implement it)
- **Observation:** Paths in the search tree correspond to paths in the state space.
- **Definition 6.3.3.** We define the path cost of a node n in a search tree T to be the sum of the step costs on the path from n to the root of T.
- **Observation:** As a search tree node has access to parents, we can read off the solution from a goal node.

It is very important to understand the fundamental difference between a state in a search problem, a node search tree employed by the tree search algorithm, and the implementation in a search tree node. The implementation above is faithful in the sense that the implemented data structures contain all the information needed in the tree search algorithm.

So we can use it to refine the idea of a tree search algorithm into an implementation.

Implementation of Search Algorithms

- **Definition 6.3.4 (Implemented Tree Search Algorithm).**

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

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

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

Note: The pseudocode in ?? is still relatively underspecified – leaves many implementation details unspecified. Here are the specifications of the functions used without.

-
- make_node constructs a search tree node from a state.
- initial_state accesses the initial state of a search problem.
- State returns the state associated with its aregument.
- GoalNode checks whether its argument is a goal node
- expand = creates new search tree nodes by for all successor states.

Essentially, only the first function is non-trivial (as the strategy argument shows) In fact it is the only place, where the strategy is used in the algorithm.

An alternative implementation would have been to make the fringe a queue, and insert order the fringe as the strategy sees fit. Then first can just return the first element of the queue. This would have lead to a different signature, possibly different runtimes, but the same overall result of the algorithm.

Search strategies

▷ **Definition 6.3.5.** 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 6.3.6 (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!

Note that there can be infinite branches, see the search tree for Romania.

6.4 Uninformed Search Strategies

Video Nuggets covering this section can be found at `https://fau.tv/clip/id/21994` and `https://fau.tv/clip/id/21995`.

Uninformed search strategies

▷ **Definition 6.4.1.** 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

The opposite of uninformed search is informed or heuristic search that uses a heuristic function that adds external guidance to the search process. In the Romania example, one could add the heuristic to prefer cities that lie in the general direction of the goal (here SE).

Even though heuristic search is usually much more efficient, uninformed search is important nonetheless, because many problems do not allow to extract good heuristics.

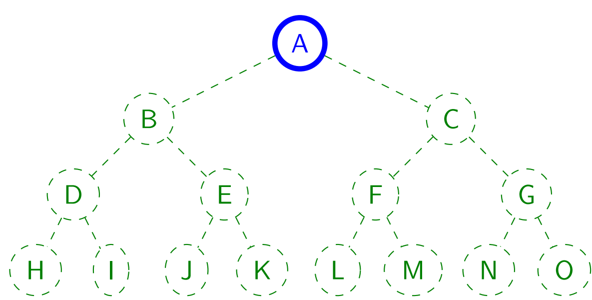
6.4.1 Breadth-First Search Strategies

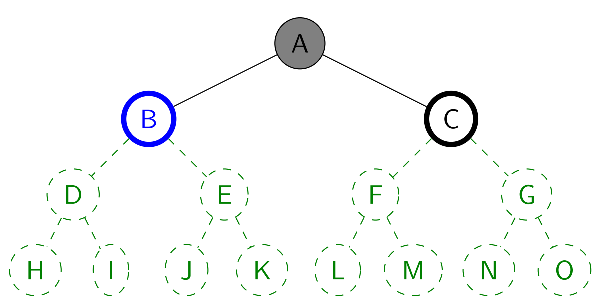
Breadth-First Search

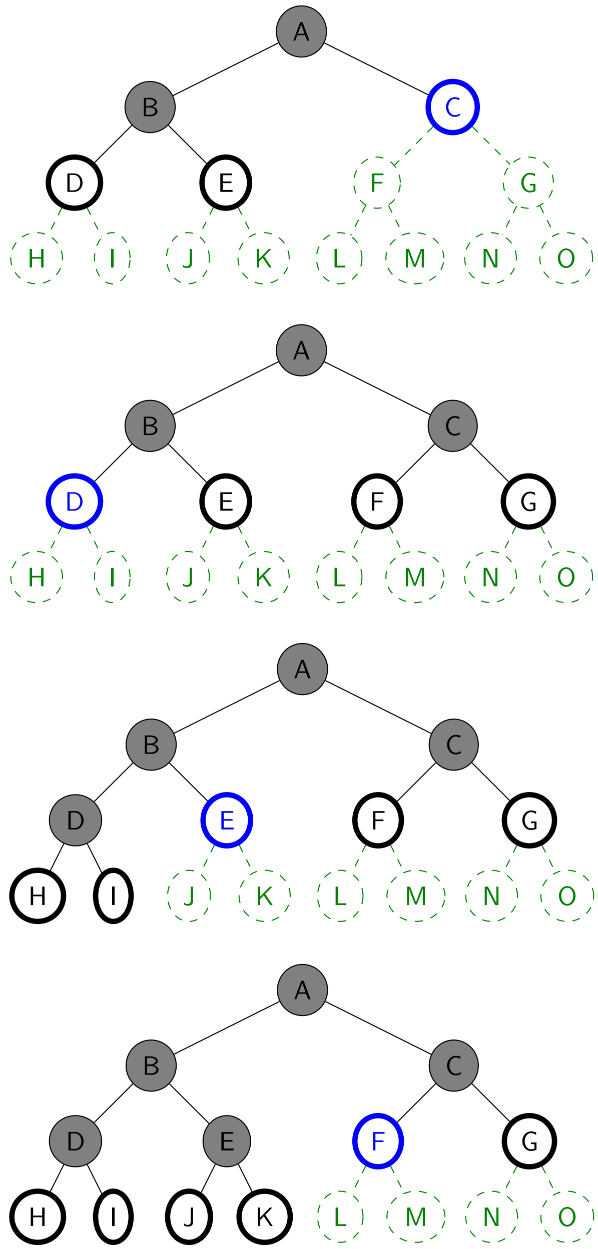
▷ **Idea:** Expand the shallowest unexpanded node.

▷ **Definition 6.4.2.** 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 6.4.3 (Synthetic).**



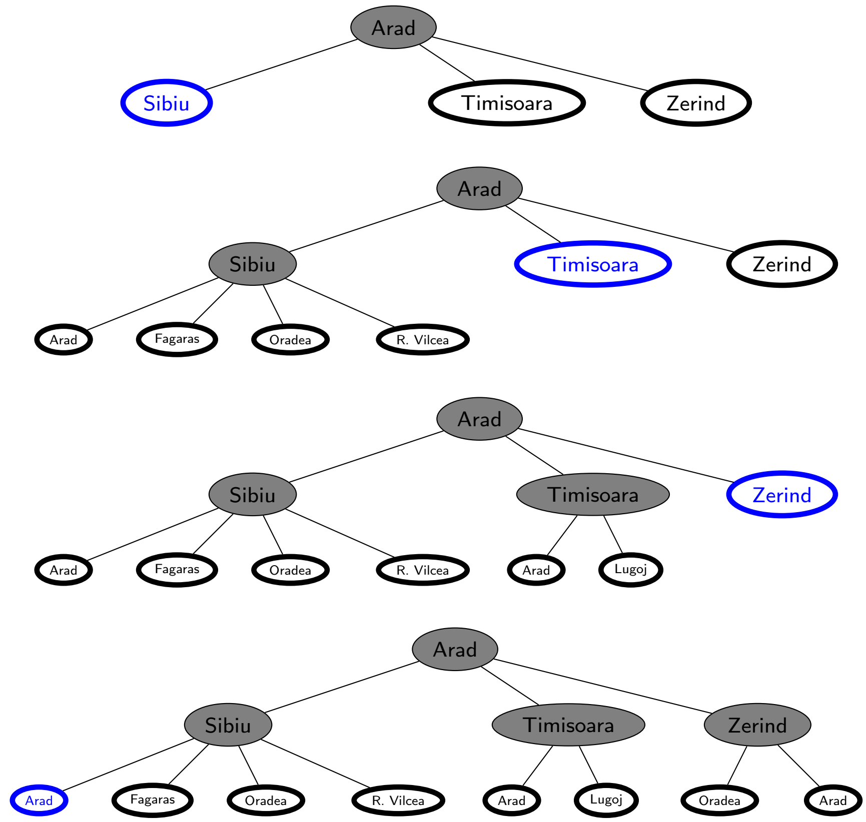




We will now apply the breadth first search strategy to our running example: Traveling in Romania. Note that we leave out the green dashed nodes that allow us a preview over what the search tree will look like (if expanded). This gives a much cleaner picture we assume that the readers already have grasped the mechanism sufficiently.

Breadth-First Search: Romania

▷ **Example 6.4.4.**



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)

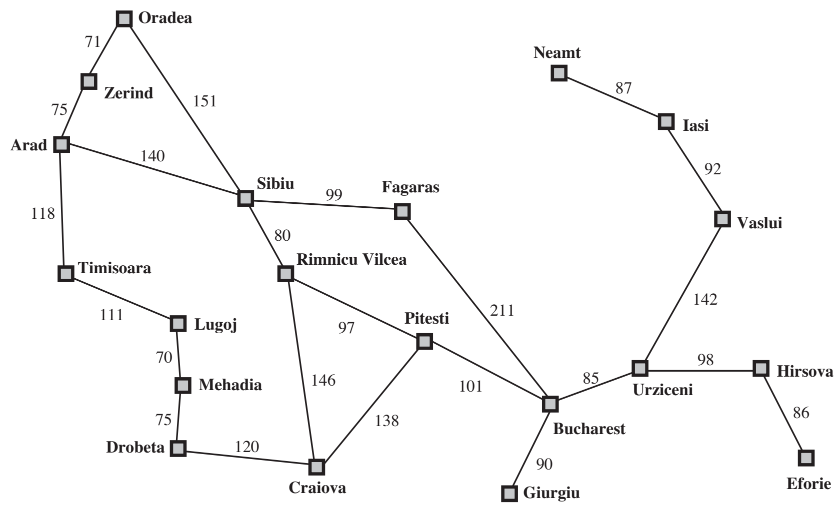
▷ **Optimal?:** No! If cost varies for different steps, there might be better solutions below the level of the first one.

▷ An alternative is to generate *all* solutions and then pick an optimal one. This works only, if m is finite.

The next idea is to let cost drive the search. For this, we will need a non-trivial cost function: we will take the distance between cities, since this is very natural. Alternatives would be the driving time, train ticket cost, or the number of tourist attractions along the way.

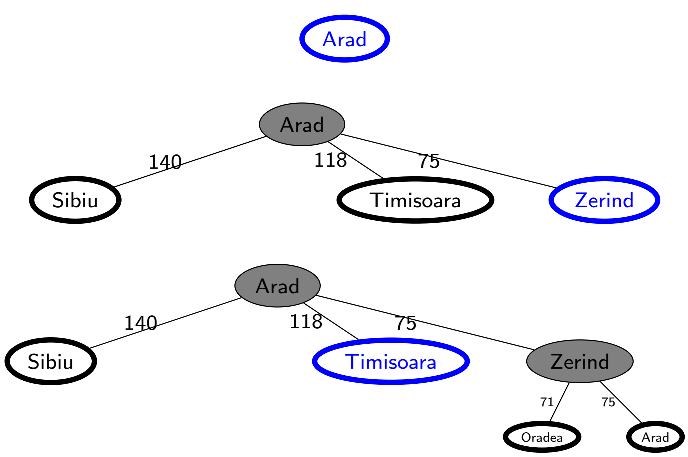
Of course we need to update our problem formulation with the necessary information.

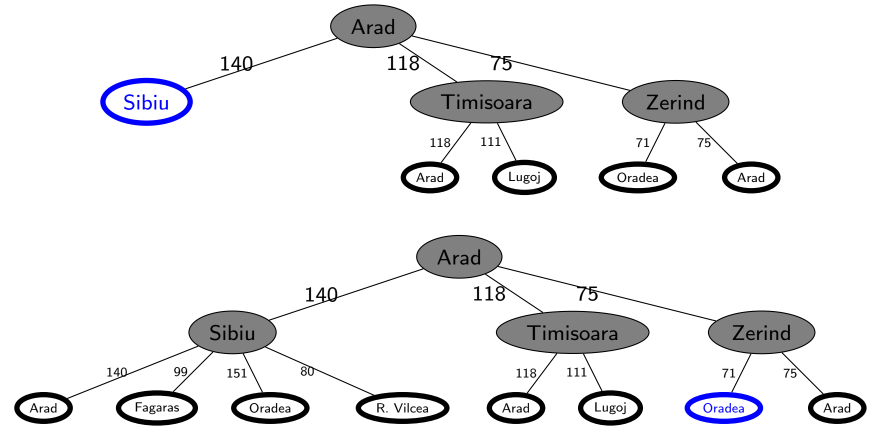
Romania with Step Costs as Distances



Uniform-cost search

- ▷ **Idea:** Expand least cost unexpanded node.
- ▷ **Definition 6.4.5.** 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:**





FAU Michael Kohlhase: Artificial Intelligence 1 144 2025-02-06

Note that we must sum the distances to each leaf. That is, we go back to the first level after the third step.

Uniform-cost search: Properties

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

FAU Michael Kohlhase: Artificial Intelligence 1 145 2025-02-06

If step cost is negative, the same situation as in breadth first search can occur: later solutions may be cheaper than the current one.

If step cost is 0, one can run into infinite branches. UCS then degenerates into depth first search, the next kind of search algorithm we will encounter. Even if we have infinite branches, where the sum of step costs converges, we can get into trouble, since the search is forced down these infinite paths before a solution can be found.

Worst case is often worse than BFS, because large trees with small steps tend to be searched first. If step costs are uniform, it degenerates to BFS.

6.4.2 Depth-First Search Strategies

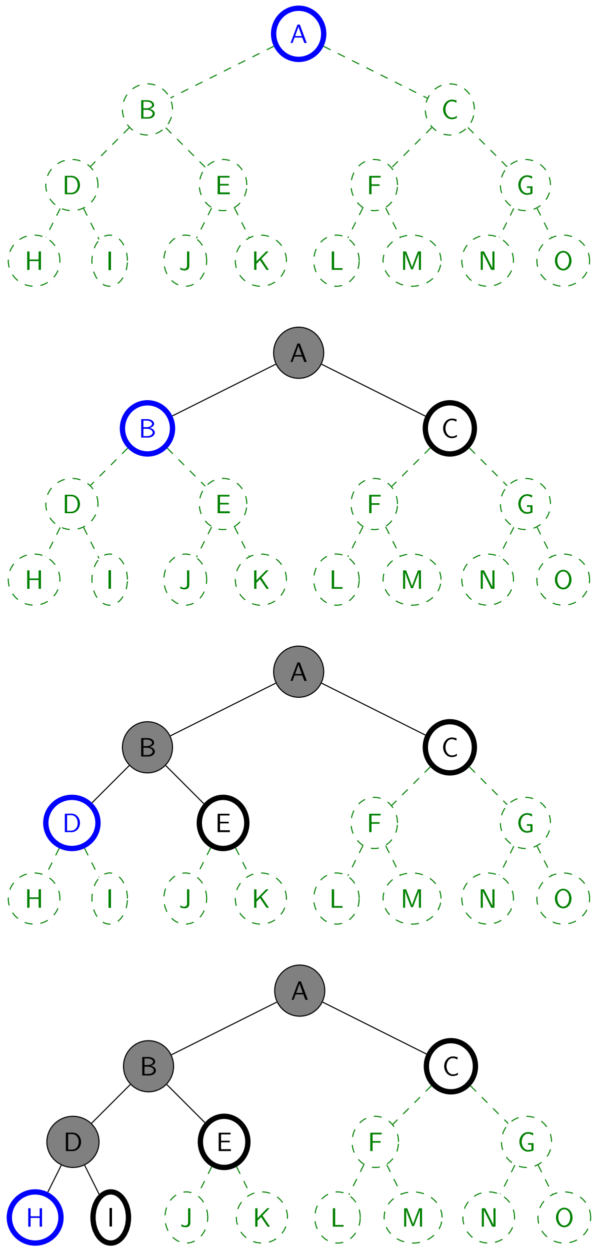
Depth-first Search

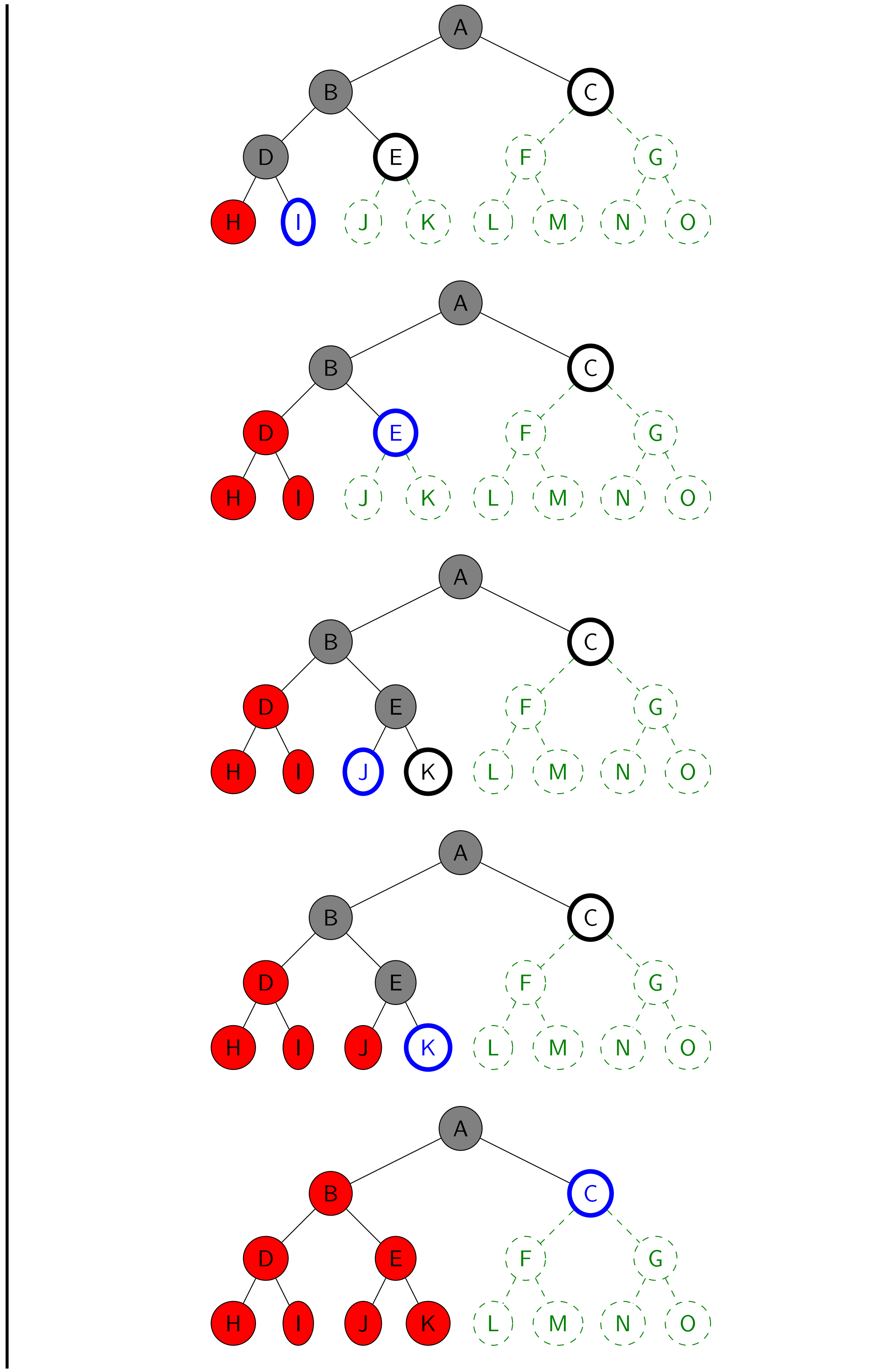
- ▷ **Idea:** Expand deepest unexpanded node.
- ▷ **Definition 6.4.6.** Depth-first search (DFS) is the strategy where the fringe is organized as a (LIFO) stack i.e. successors go in at front of the fringe.
- ▷ **Definition 6.4.7.** Every node that is pushed to the stack is called a backtrack point. 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.

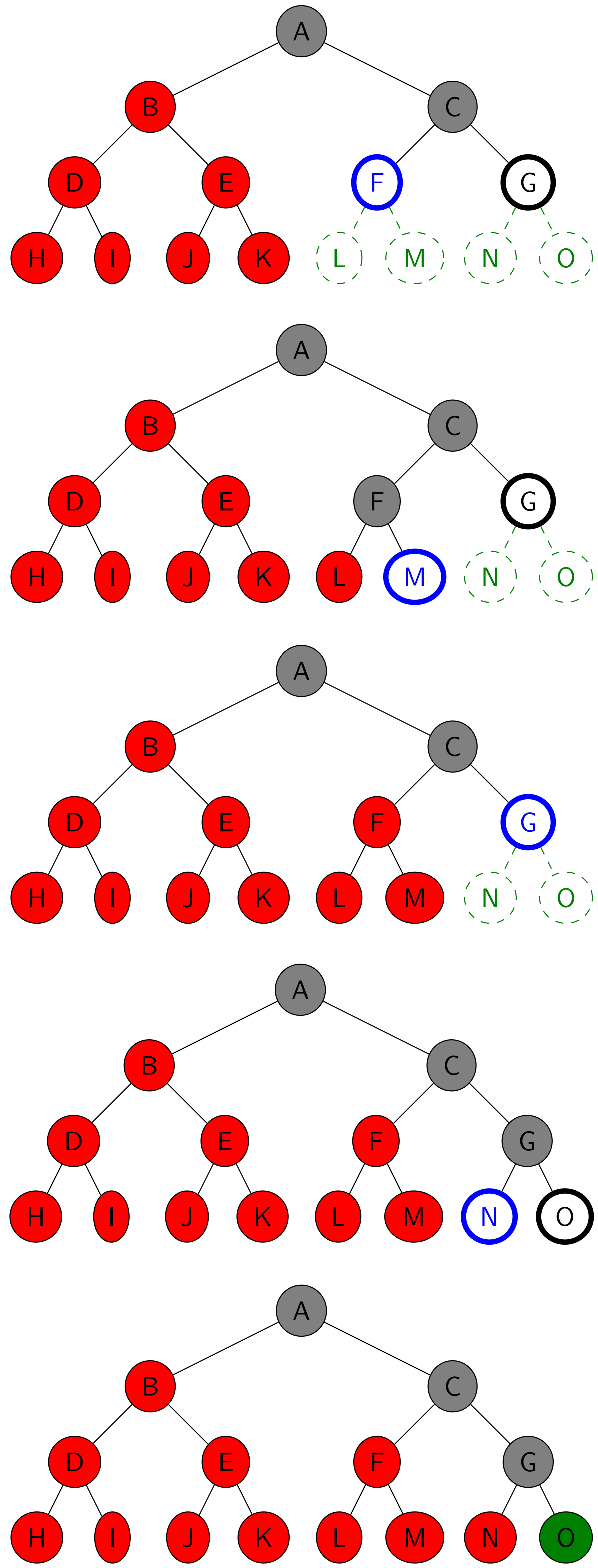
▷ **Note:** Depth first search can perform infinite cyclic excursions
Need a finite, non cyclic state space (or repeated state checking)

Depth-First Search

▷ **Example 6.4.8 (Synthetic).**

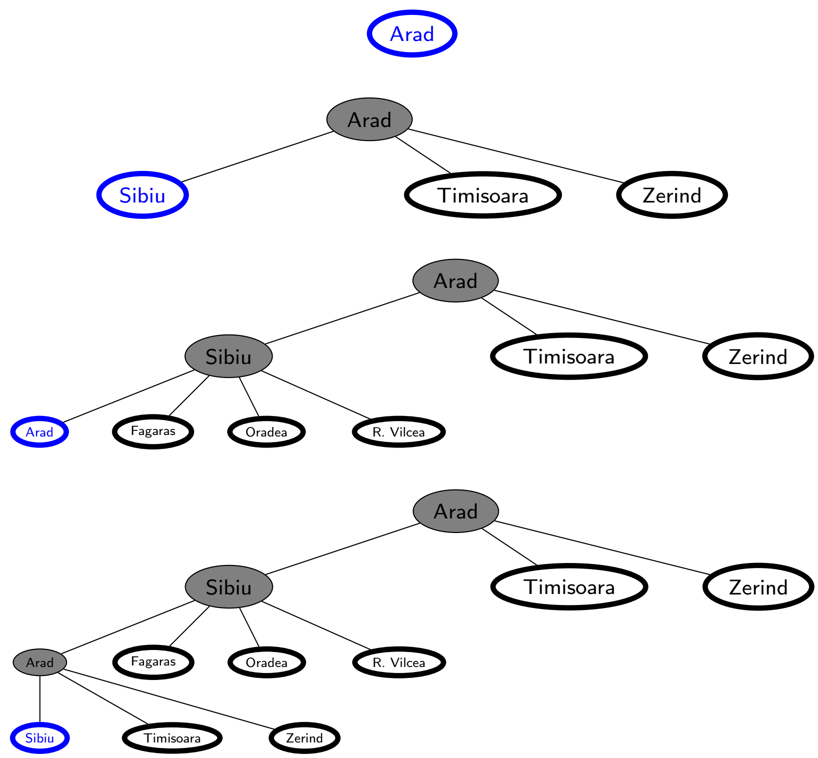



A
B
C
D
E
F
G
H
I
J
K
L
M
N
O


A
B C
D E F G
H I J K L M N O
A
B C
D E F G
H I J K L M N O
A
B C
D E F G
H I J K L M N O
A
B C
D E F G
H I J K L M N O
A
B C
D E F G
H I J K L M N O

Depth-First Search: Romania

▷ **Example 6.4.9 (Romania).**



Depth-first search: Properties

▷

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

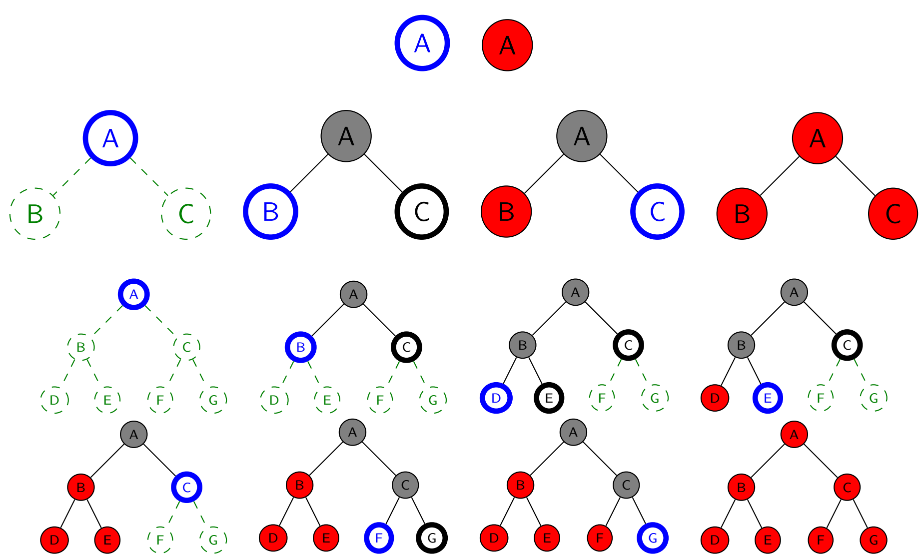
▷ **Disadvantage:** Time terrible if m much larger than d.

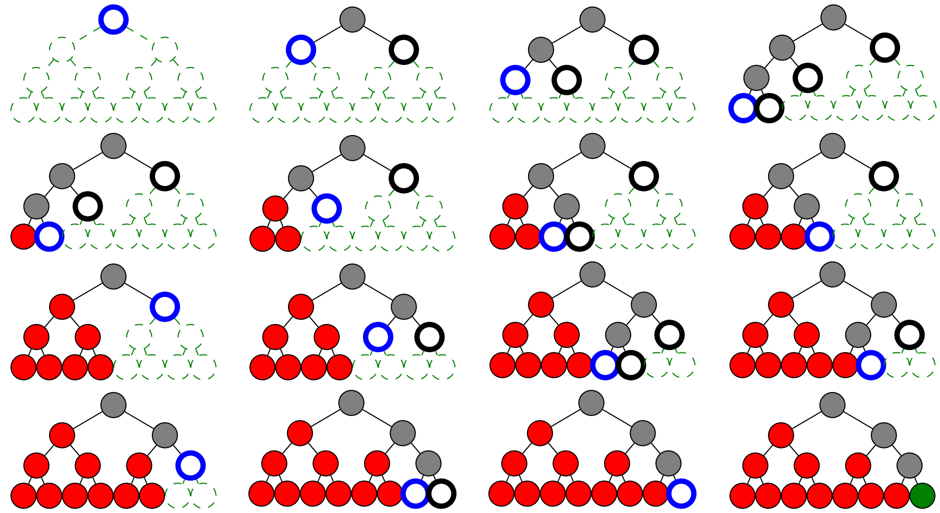
▷ **Advantage:** Time may be much less than breadth first search if solutions are dense.

Iterative deepening search

- **Definition 6.4.10.** Depth limited search is depth first search with a depth limit.
- **Definition 6.4.11.** Iterative deepening search (IDS) is depth limited search with ever increasing depth limits. We call the difference between successive depth limits the step size.
- **procedure** Tree_Search (problem)
  <*initialize the search tree using the initial state of problem*>
  for depth = 0 **to** ∞
  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





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.

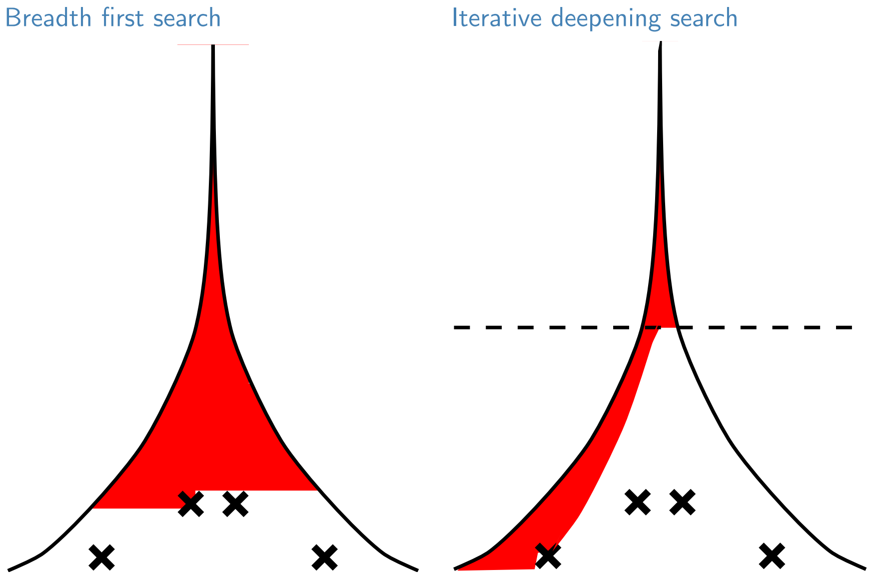
Note: To find a solution (at depth d) we have to search the whole tree up to d. Of course since we do not save the search state, we have to re-compute the upper part of the tree for the next level. This seems like a great waste of resources at first, however, IDS tries to be complete without the space penalties.

However, the space complexity is as good as DFS, since we are using DFS along the way. Like in BFS, the whole tree on level d (of optimal solution) is explored, so optimality is inherited from there. Like BFS, one can modify this to incorporate uniform cost search behavior.

As a consequence, variants of IDS are the method of choice if we do not have additional information.

Comparison BFS (optimal) and IDS (not)

▷ **Example 6.4.12.** IDS may fail to be be optimal at step sizes > 1.



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 6.4.13.** 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 6.4.14.** *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 6.4.15.** 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

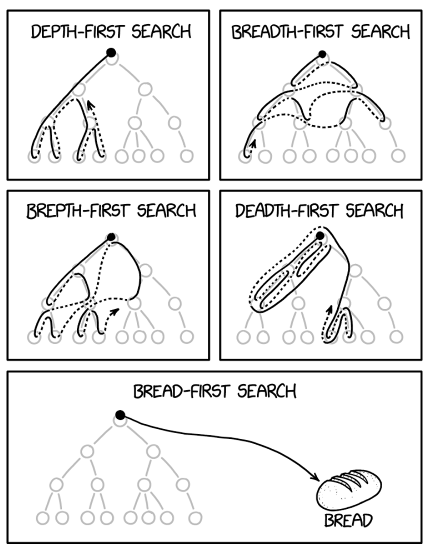
- ▷ **Tree/Graph Search Algorithms:** Systematically explore the state tree/graph induced by a search problem in search of a goal state. Search strategies only differ by the treatment of the fringe.

▷ **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] $0 < \epsilon \leq$ 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 6.5.1.** 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

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/22015`.

Best-first search

▷ **Idea:** Order the fringe by estimated "desirability" (Expand most desirable unexpanded node)

▷ **Definition 6.5.2.** 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 6.5.3.** In best first search, the fringe is a queue sorted in decreasing order of desirability.

▷ **Special cases:** Greedy search, A^* search

This is like UCS, but with an evaluation function related to problem at hand replacing the path cost function.

If the heuristic is arbitrary, we expect incompleteness!

Depends on how we measure "desirability".

Concrete examples follow.

Greedy search

▷ **Idea:** Expand the node that *appears* to be closest to the goal.

▷ **Definition 6.5.4.** 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 6.5.5.** Given a heuristic h, greedy search is the strategy where the fringe is organized as a queue sorted by increasing h value.

▷ **Example 6.5.6.** 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

internal search control ⇝ external search control
partial solution cost ⇝ goal cost estimation

FAU Michael Kohlhase: Artificial Intelligence 1 159 2025-02-06

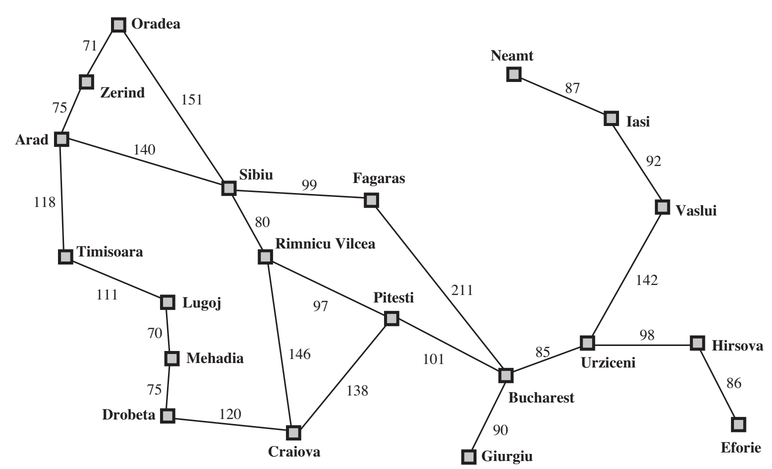
In greedy search we replace the *objective* cost to *construct* the current solution with a heuristic or *subjective* measure from which we think it gives a good idea how far we are from a solution. Two things have shifted:

- we went from internal (determined only by features inherent in the search space) to an external/heuristic cost
- instead of measuring the cost to build the current partial solution, we estimate how far we are from the desired goal

Romania with Straight-Line Distances

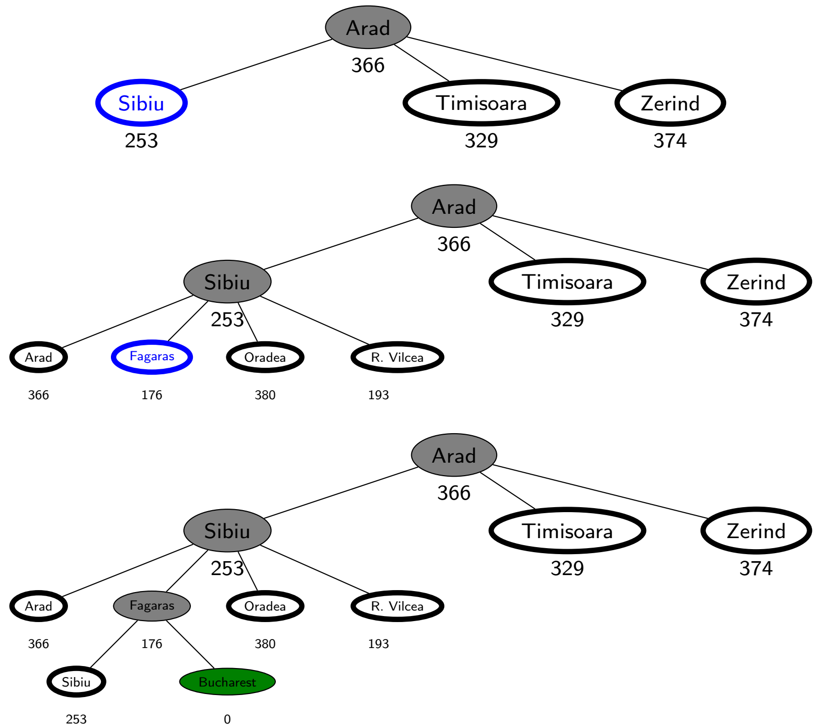
▷ **Example 6.5.7 (Informed Travel).** $h_{\mathrm{SLD}}(n) = \mathit{straight-line\ distance\ to\ Bucharest}$

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



FAU Michael Kohlhase: Artificial Intelligence 1 160 2025-02-06

Greedy Search: Romania



Let us fortify our intuitions with another example: navigation in a simple maze. Here the states are the cells in the grid underlying the maze and the actions navigating to one of the adjoining cells. The initial and goal states are the left upper and right lower corners of the grid. To see the influence of the chosen heuristic (indicated by the red number in the cell), we compare the search induced goal distance function with a heuristic based on the Manhattan distance. Just follow the greedy search by following the heuristic gradient.

Heuristic Functions in Path Planning

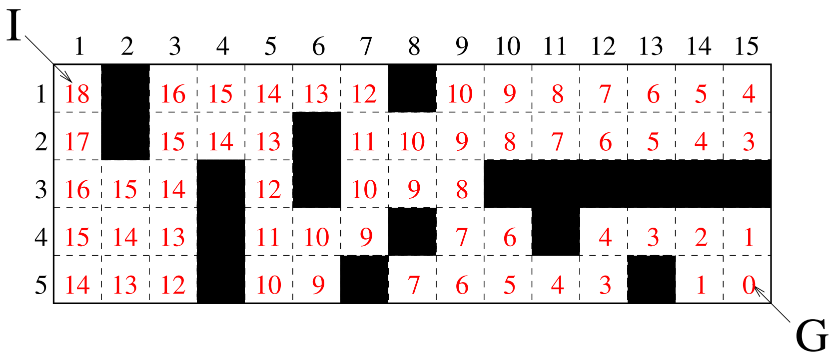
▷ **Example 6.5.8 (The maze solved).** We indicate h^* by giving the goal distance:

I (initial state: cell 1,1)

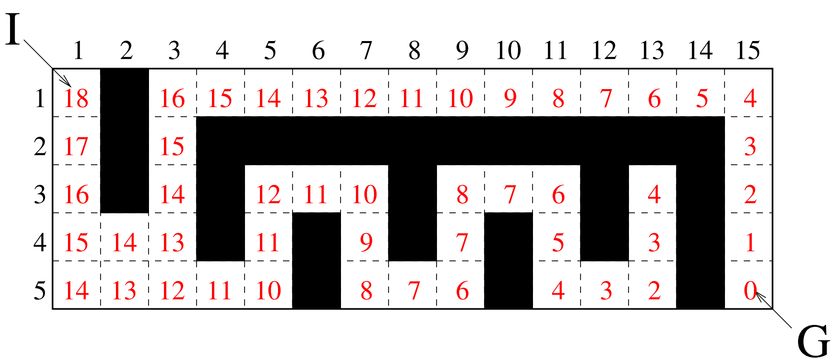
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	24	■	18	17	16	15	14	■	12	13	14	15	16	17	18
2	23	■	19	18	17	■	13	12	11	12	13	14	15	16	17
3	22	21	20	■	16	■	12	11	10	■	■	■	■	■	■
4	23	22	21	■	15	14	13	■	9	8	■	4	3	2	1
5	24	23	22	■	16	15	■	9	8	7	6	5	■	1	0

G (goal state: cell 5,15)

▷ **Example 6.5.9 (Maze Heuristic: The good case).** We use the Manhattan distance to the goal as a heuristic:



▷ **Example 6.5.10 (Maze Heuristic: The bad case).** We use the Manhattan distance to the goal as a heuristic again:



Not surprisingly, the first maze is searchless, since we are guided by the perfect heuristic. In cases, where there is a choice, the this has no influence on the length (or in other cases cost) of the solution.

In the "good case" example, greedy search performs well, but there is some limited backtracking needed, for instance when exploring the left lower corner 3×3 area before climbing over the second wall.

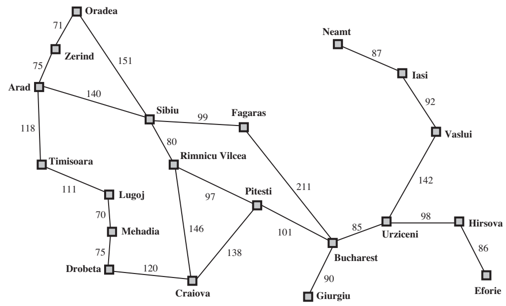
In the "bad case", greedy search is led down the lower garden path, which has a dead end, and does not lead to the goal. This suggests that there we can construct adversary examples – i.e. example mazes where we can force greedy search into arbitrarily bad performance.

Greedy search: Properties

▷

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

▷ **Example 6.5.11.** Greedy search can get stuck going from Iasi to Oradea: Iasi → Neamt → Iasi → Neamt → ···



- ▷ **Worst-case Time:** Same as depth first search.
- ▷ **Worst-case Space:** Same as breadth first search. (⇝ repeated state checking)
- ▷ **But:** A good heuristic can give dramatic improvements.

Remark 6.5.12. Greedy search is similar to UCS. Unlike the latter, the node evaluation function has nothing to do with the nodes explored so far. This can prevent nodes from being enumerated systematically as they are in UCS and BFS.

For completeness, we need repeated state checking as the example shows. This enforces complete enumeration of the state space (provided that it is finite), and thus gives us completeness.

Note that nothing prevents from *all* nodes being searched in worst case; e.g. if the heuristic function gives us the same (low) estimate on all nodes except where the heuristic mis-estimates the distance to be high. So in the worst case, greedy search is even worse than BFS, where d (depth of first solution) replaces m.

The search procedure cannot be optimal, since actual cost of solution is not considered.

For both, completeness and optimality, therefore, it is necessary to take the actual cost of partial solutions, i.e. the path cost, into account. This way, paths that are known to be expensive are avoided.

6.5.2 Heuristics and their Properties

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/22019`.

Heuristic Functions

- ▷ **Definition 6.5.13.** 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 6.5.14.** 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 6.5.15 (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 6.5.16.** *We need to trade off the accuracy of h against the overhead for computing it.*

Properties of Heuristic Functions

- **Definition 6.5.17.** 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 search problem, and let h be a heuristic for Π. If h is consistent, then h is admissible.

▷ *Proof:* we prove $h(s) \leq h^*(s)$ for all $s \in S$ by induction over the length of the cheapest path to a goal node.

1. base case
   1.1. $h(s) = 0$ by definition of heuristic, so $h(s) \leq h^*(s)$ as desired.
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 6.5.18.** 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 6.5.19 (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 Video Nugget covering this subsection can be found at https://fau.tv/clip/id/22020.

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 6.5.20.** 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 6.5.21.** Best first search with evaluation function $g + h$ is called A^* search.

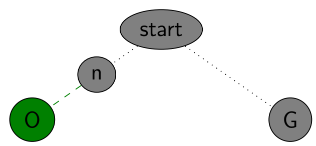
This works, provided that h does not overestimate the true cost to achieve the goal. In other words, h must be *optimistic* wrt. the real cost h^*. If we are too pessimistic, then non-optimal solutions have a chance.

A^* Search: Optimality

▷ **Theorem 6.5.22.** *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:

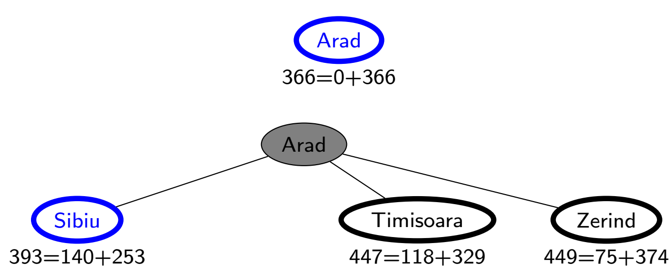


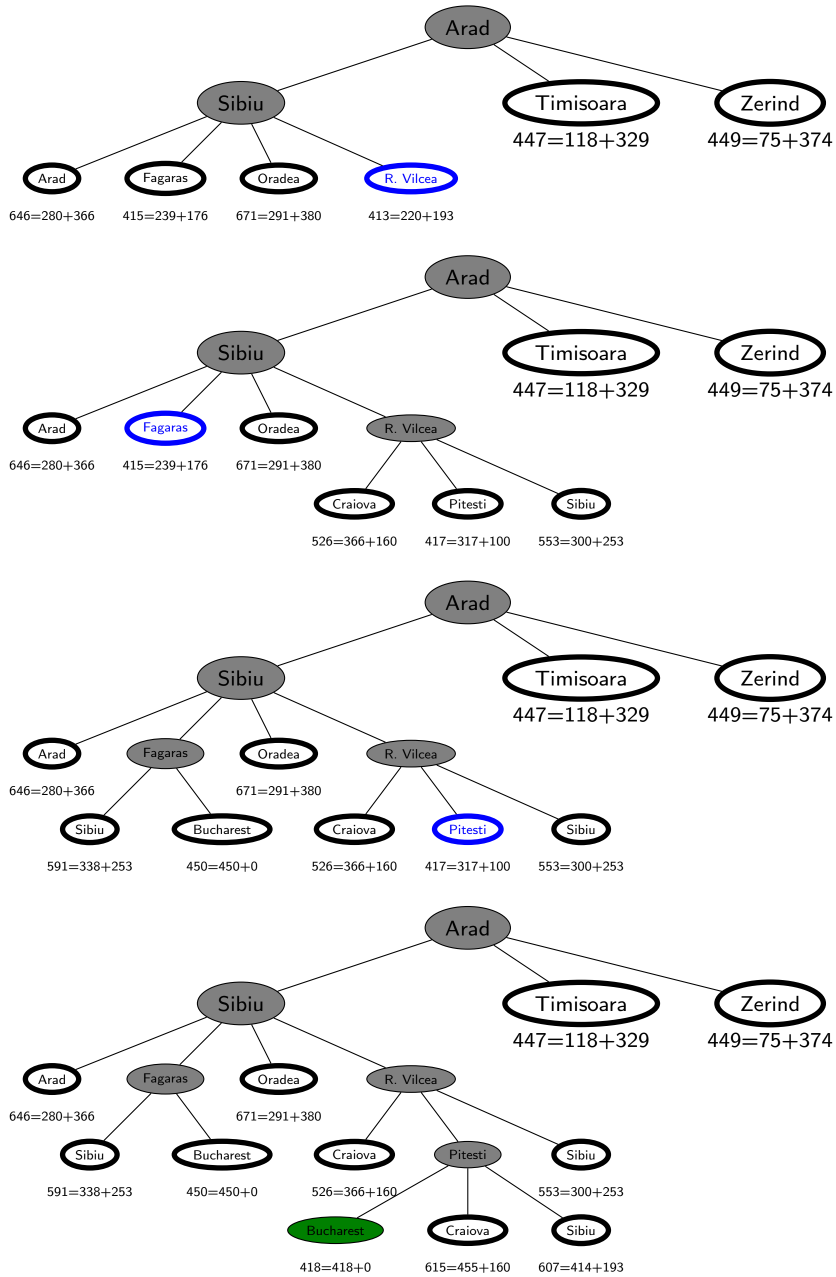
2. Let n be an unexpanded node on a path to an optimality goal node O, then

$f(G) = g(G)$	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

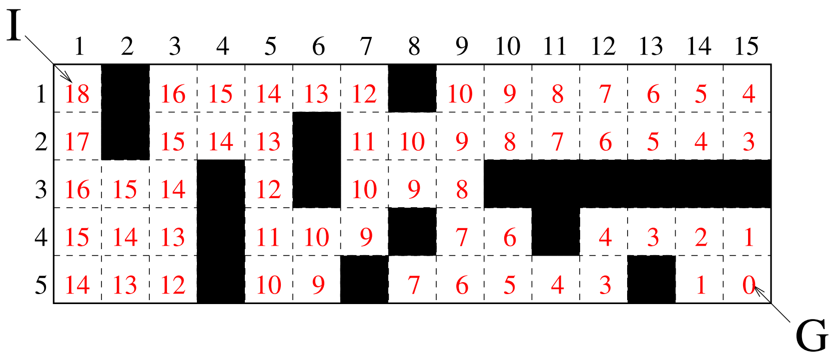




To extend our intuitions about informed search algorithms to A^*-search, we take up the maze examples from above again. We first show the good maze with Manhattan distance again.

Additional Observations (Not Limited to Path Planning)

▷ **Example 6.5.23 (Greedy best-first search, "good case").**

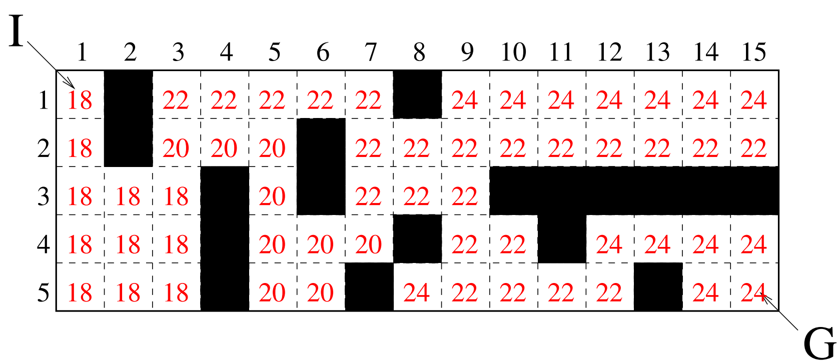


We will find a solution with little search.

To compare it to A^*-search, here is the same maze but now with the numbers in red for the evaluation function f where h is the Manhattan distance.

Additional Observations (Not Limited to Path Planning)

▷ **Example 6.5.24 (A^* $(g + h)$, "good case").**

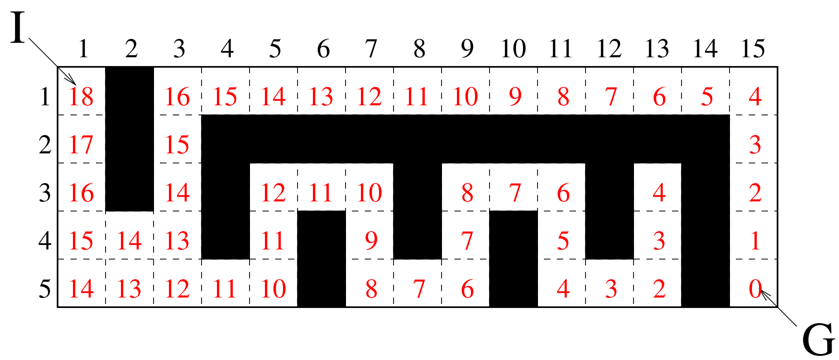


- ▷ In A^* with a consistent heuristic, $g + h$ always increases monotonically (h cannot decrease more than g increases)
- ▷ We need more search, in the "right upper half". This is typical: Greedy best first search tends to be faster than A^*.

Let's now consider the "bad maze" with Manhattan distance again.

Additional Observations (Not Limited to Path Planning)

▷ **Example 6.5.25 (Greedy best-first search, "bad case").**

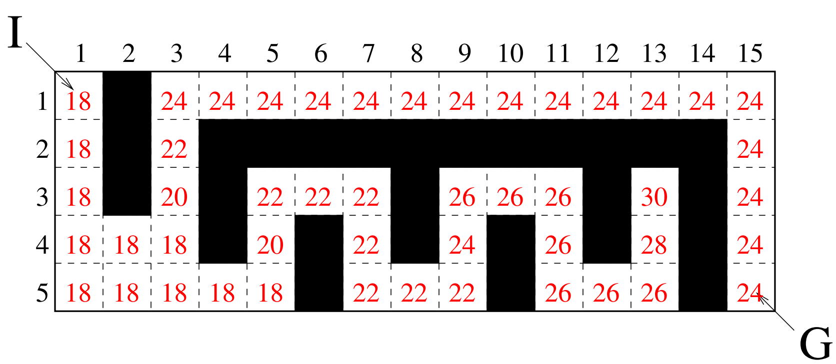


Search will be mis-guided into the "dead-end street".

And we compare it to A^*-search; again the numbers in red are for the evaluation function f.

Additional Observations (Not Limited to Path Planning)

▷ **Example 6.5.26 (A^* ($g+h$), "bad case").**

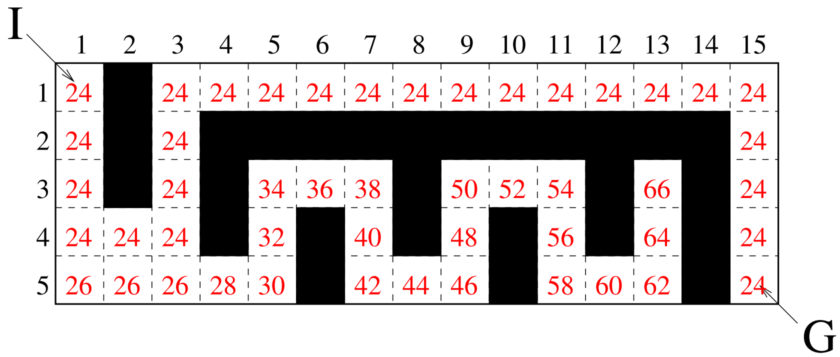


We will search less of the "dead-end street". Sometimes $g+h$ gives better search guidance than h. ($\leadsto$ A^* is faster there)

Finally, we compare that with the goal distance function for the "bad maze". Here we see that the lower garden path is under-estimated by the evaluation function f, but still large enough to keep the search out of it, thanks to the admissibility of the Manhattan distance.

Additional Observations (Not Limited to Path Planning)

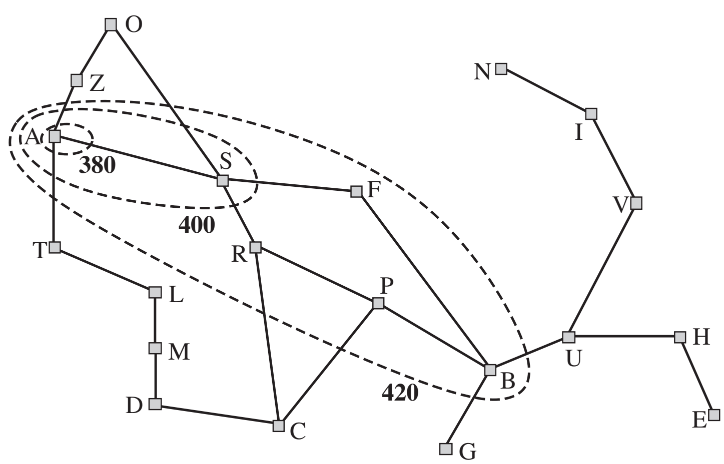
▷ **Example 6.5.27 (A^* ($g+h$) using h^*).**



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

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



A^* search: Properties

▷ Properties or A^*-search:

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

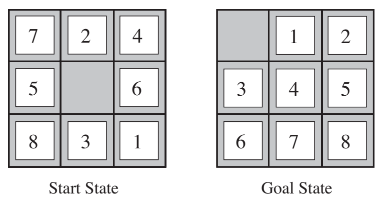
- ▷ A^*-search expands all (some/no) nodes with $f(n) < h^*(n)$
- ▷ The run-time depends on how well we approximated the real cost h^* with h.

Michael Kohlhase: Artificial Intelligence 1 179 2025-02-06

6.5.4 Finding Good Heuristics

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/22021`.
Since the availability of admissible heuristics is so important for informed search (particularly for A^*-search), let us see how such heuristics can be obtained in practice. We will look at an example, and then derive a general procedure from that.

Admissible heuristics: Example 8-puzzle



- ▷ **Example 6.5.28.** Let $h_1(n)$ be the number of misplaced tiles in node n. ($h_1(S) = 9$)
- ▷ **Example 6.5.29.** 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 6.5.30 (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*

Michael Kohlhase: Artificial Intelligence 1 180 2025-02-06

Actually, the crucial difference between the heuristics h_1 and h_2 is that – not only in the example configuration above, but for all configurations – the the value of the latter is larger than that of the former. We will explore this next.

Dominance

- ▷ **Definition 6.5.31.** 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 6.5.32.** *If h_2 dominates h_1, then h_2 is better for search than h_1.*
- ▷ *Proof sketch:* If h_2 dominates h_1, then h_2 is "closer to h^*" than h_1, which means better search performance.

Michael Kohlhase: Artificial Intelligence 1 181 2025-02-06

We now try to generalize these insights into (the beginnings of) a general method for obtaining admissible heuristics.

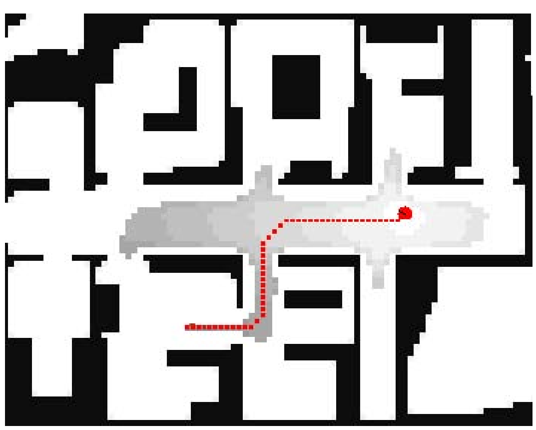
Relaxed problems

- **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 6.5.33.** If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then we get heuristic h_1.
- **Example 6.5.34.** If the rules are relaxed so that a tile can move to *any adjacent square*, then we get heuristic h_2. (Manhattan distance)
- **Definition 6.5.35.** 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 6.5.36.** *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.

Relaxation means to remove some of the constraints or requirements of the original problem, so that a solution becomes easy to find. Then the cost of this easy solution can be used as an optimistic approximation of the problem.

Empirical Performance: A^* in Path Planning

- **Example 6.5.37 (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

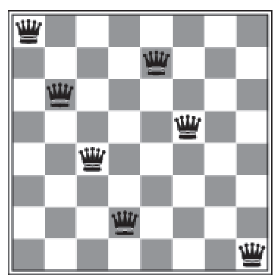
Video Nuggets covering this section can be found at https://fau.tv/clip/id/22050 and https://fau.tv/clip/id/22051.

Systematic Search vs. Local Search

- ▷ **Definition 6.6.1.** We call a search algorithm systematic, if it considers all states at some point.
- ▷ **Example 6.6.2.** All tree search algorithms (except pure depth first search) are systematic. (given reasonable assumptions e.g. about costs.)
- ▷ **Observation 6.6.3.** *Systematic search algorithms are complete.*
- ▷ **Observation 6.6.4.** *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 6.6.5 (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 6.6.6.** A local search algorithm is a search algorithm that operates on a single state, the current state (rather than multiple paths). (advantage: constant space)

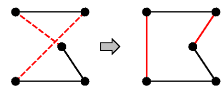


- ▷ Typically local search algorithms only move to successor of the current state, and do not retain search paths.
- ▷ Applications include: integrated circuit design, factory-floor layout, job-shop scheduling, portfolio management, fleet deployment,. . .

Local Search: Iterative improvement algorithms

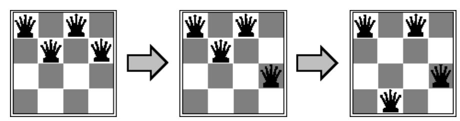
- ▷ **Definition 6.6.7.** The traveling salesman problem (TSP is to find shortest trip through set of cities such that each city is visited exactly once.

▷ **Idea:** Start with any complete tour, perform pairwise exchanges



▷ **Definition 6.6.8.** 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 6.6.9.** 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.

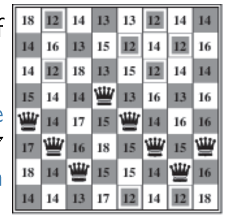
In order to understand the procedure on a more intuitive level, let us consider the following scenario: We are in a dark landscape (or we are blind), and we want to find the highest hill. The search procedure above tells us to start our search anywhere, and for every step first feel around, and then take a step into the direction with the steepest ascent. If we reach a place, where the next step would take us down, we are finished.

Of course, this will only get us into local maxima, and has no guarantee of getting us into global ones (remember, we are blind). The solution to this problem is to re-start the search at random (we do not have any information) places, and hope that one of the random jumps will get us to a slope that leads to a global maximum.

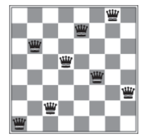
Example Hill Climbing with 8 Queens

▷ **Idea:** Consider $h \widehat{=}$ number of queens that threaten each other.

▷ **Example 6.6.10.** An 8-queens state with heuristic cost estimate $h = 17$ showing h-values for moving a queen within its column:



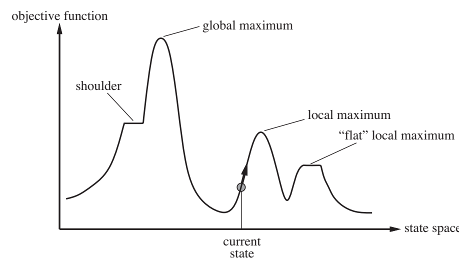
▷ **Problem:** The state space has local minima. e.g. the board on the right has $h = 1$ but every successor has $h > 1$.



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 6.6.11.** local search, simulated annealing, . . .

▷ **Properties:** All are incomplete, nonoptimal.

▷ Sometimes performs well in practice (if (optimal) solutions are dense)

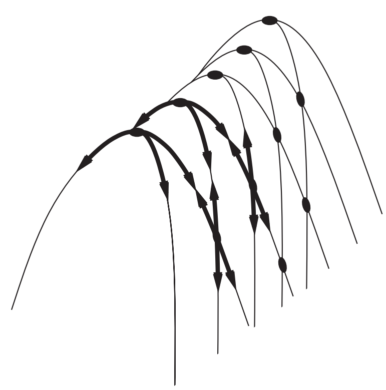
Recent work on hill climbing algorithms tries to combine complete search with randomization to escape certain odd phenomena occurring in statistical distribution of solutions.

Simulated annealing (Idea)

▷ **Definition 6.6.12.** 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 6.6.13.** 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 $\leadsto$ always reach best state x^* because

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

for small T.

▷ **Question:** Is this necessarily an interesting guarantee?

Local beam search

▷ **Definition 6.6.14.** 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!)

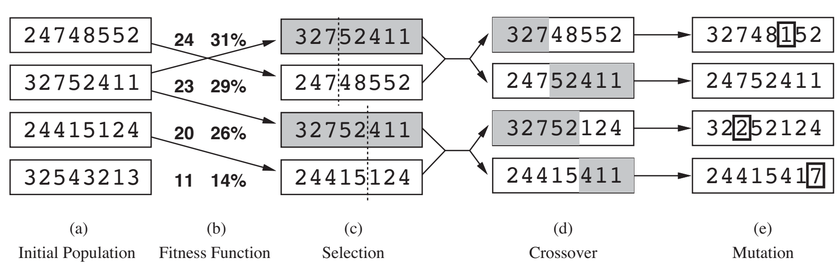
Genetic algorithms (very briefly)

▷ **Definition 6.6.15.** 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 6.6.16.** Generating successors for 8 queens

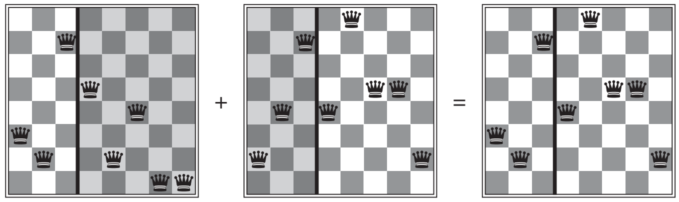


Genetic algorithms (continued)

▷ **Problem:** Genetic algorithms require states encoded as strings.

▷ Crossover only helps iff substrings are meaningful components.

▷ **Example 6.6.17 (Evolving 8 Queens).** First crossover



▷ **Note:** Genetic algorithms ≠ evolution: e.g., real genes also encode replication machinery!

Chapter 7

Adversarial Search for Game Playing

A Video Nugget covering this chapter can be found at `https://fau.tv/clip/id/22079`.

7.1 Introduction

Video Nuggets covering this section can be found at `https://fau.tv/clip/id/22060` and `https://fau.tv/clip/id/22061`.

The Problem

- ▷ **The Problem of Game-Play:** cf. ??
- ▷ **Example 7.1.1.**

- ▷ **Definition 7.1.2.** 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 7.1.3 (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 7.1.4.** 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 7.1.5 (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 7.1.6.** 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 7.1.7.** A strategy is called optimal if it yields the best possible utility for X assuming perfect opponent play (not formalized here).
- **Problem:** In (almost) all games, computing an optimal strategy is infeasible. (state/search tree too huge)
- **Solution:** Compute the next move "on demand", given the current state instead.

Why Games are hard to solve II

- **Example 7.1.8.** Number of reachable states in chess: 10^{40}.
- **Example 7.1.9.** 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 7.1.10.**
  - 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 description 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

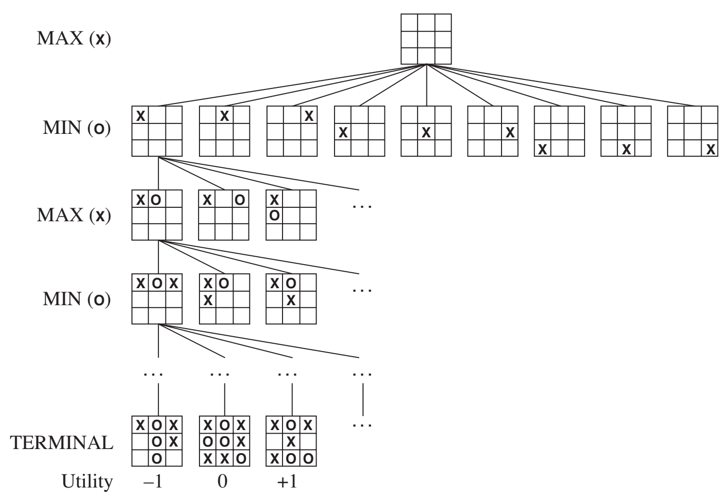
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22061`.

"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 $\rightsquigarrow$ hence "minimax".

Example Tic-Tac-Toe

- **Example 7.2.1.** A full game tree for tic-tac-toe



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

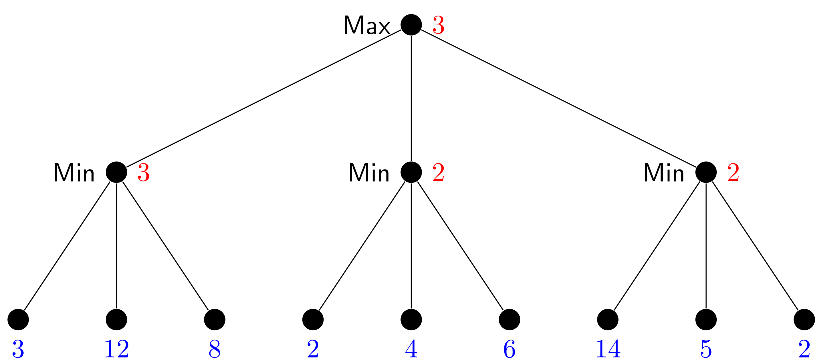
Minimax: Outline

- **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

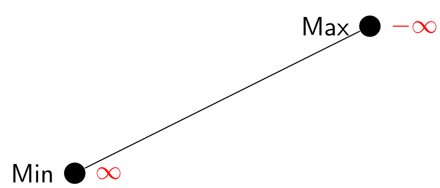
▷ **Definition 7.2.2.** The minimax algorithm (often just called minimax) is given by the following functions whose argument is a state $s \in \mathcal{S}^{\mathrm{Max}}$, in which Max is to move.

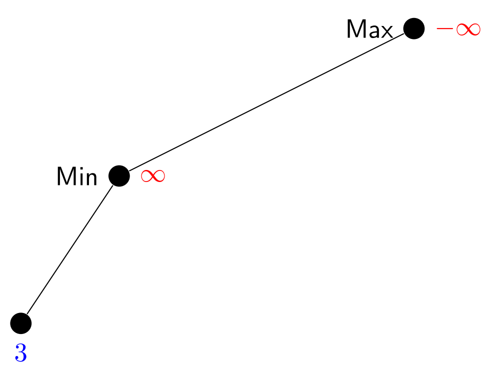
```
function Minimax−Decision(s) returns an action
  v := Max−Value(s)
  return an action yielding value v in the previous function call

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

function Min−Value(s) returns a utility value
```

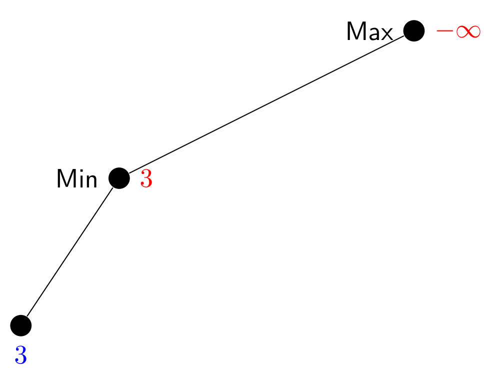
if Terminal−Test(s) **then return** $u(s)$
v := $+\infty$
for each $a \in$ Actions(s) **do**
  v := min(v,Max−**Value**(ChildState(s,a)))
return v

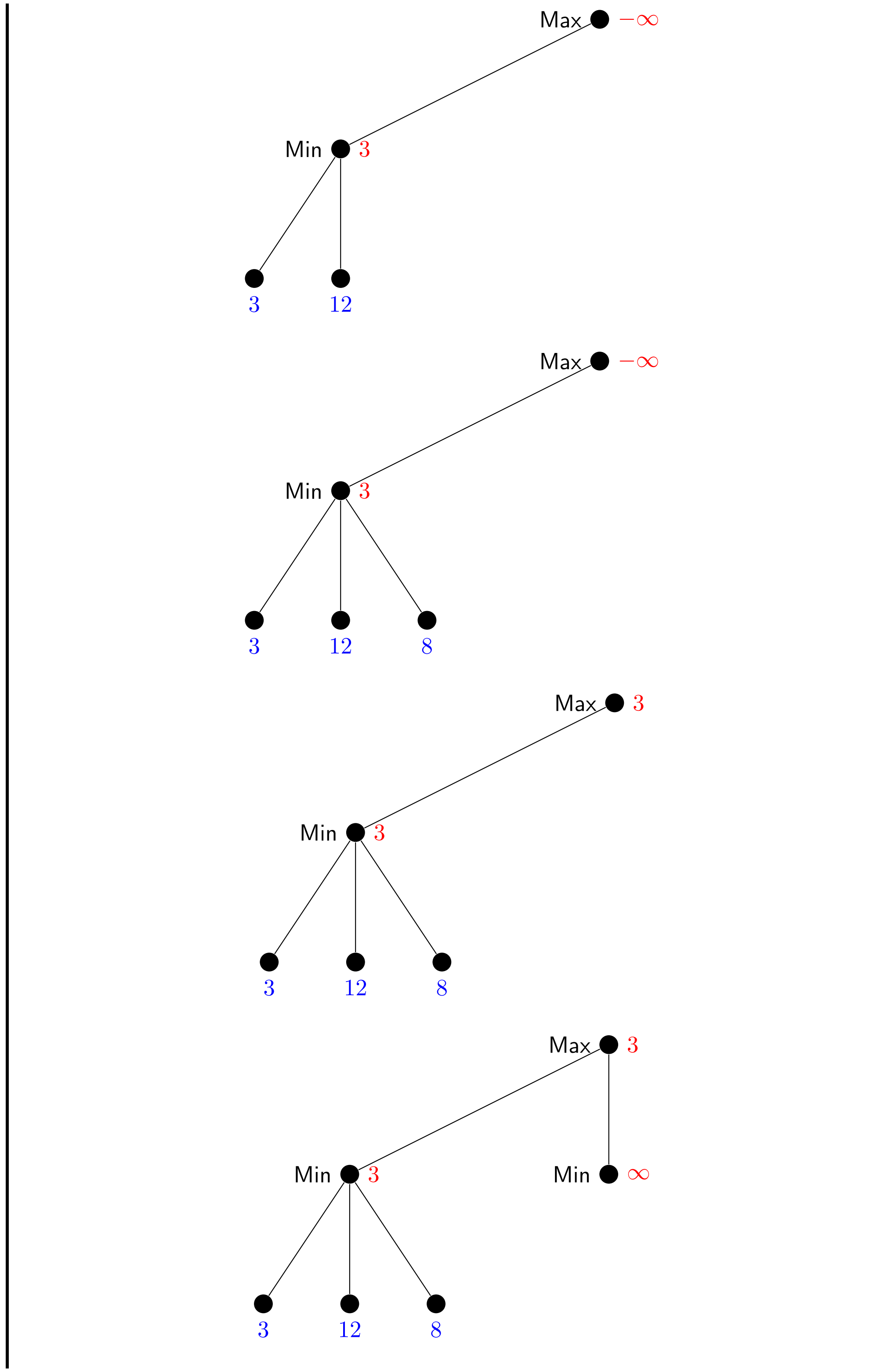
We call nodes, where Max/Min acts Max-nodes/Min-nodes.

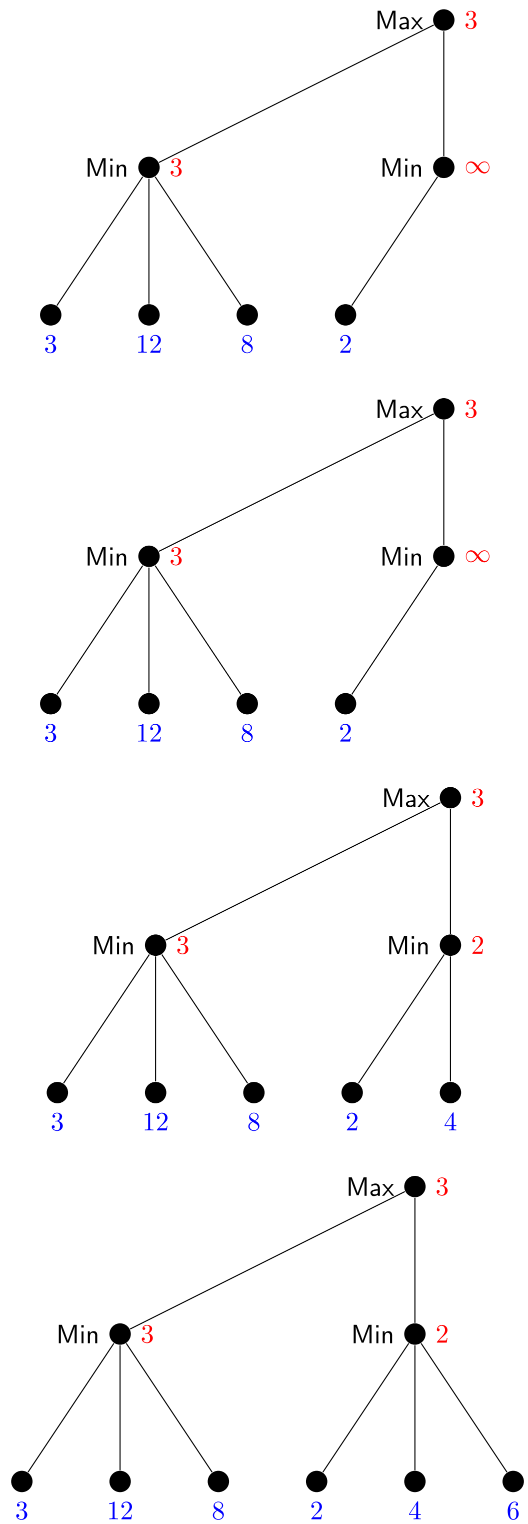
Minimax: Example, Now in Detail

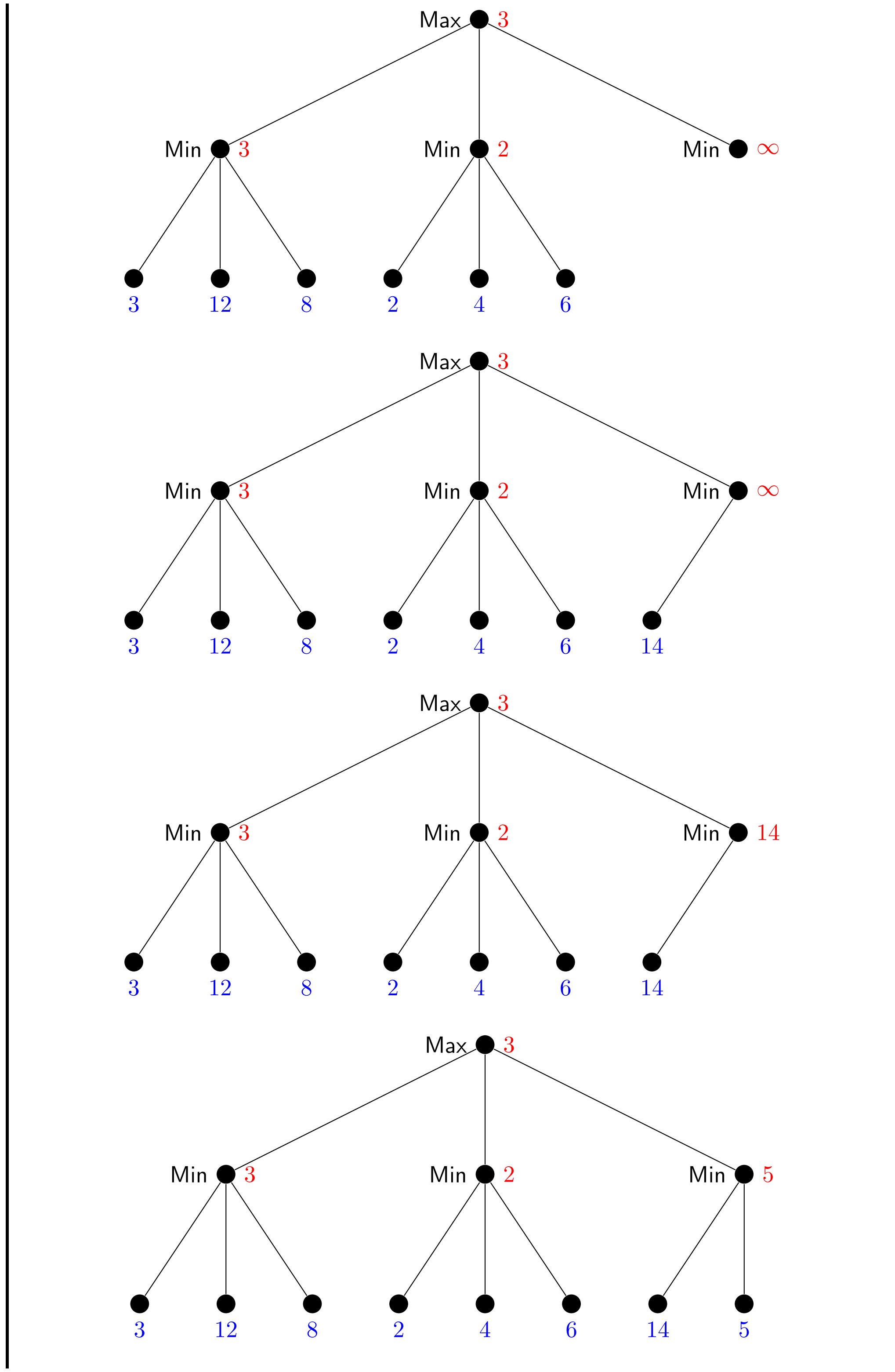


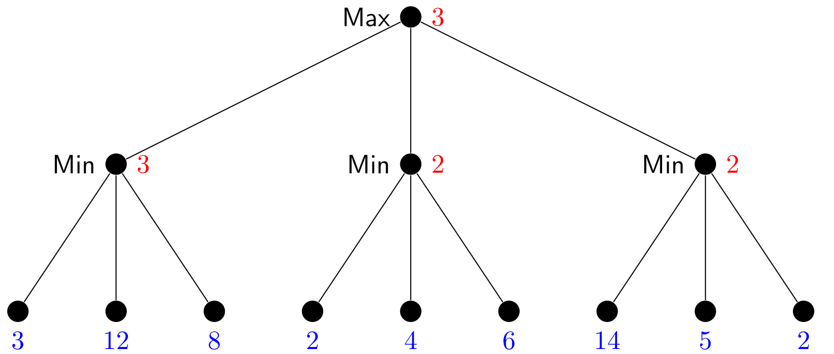


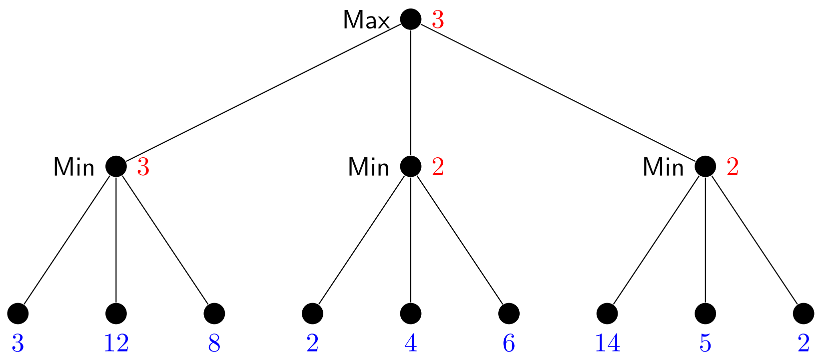


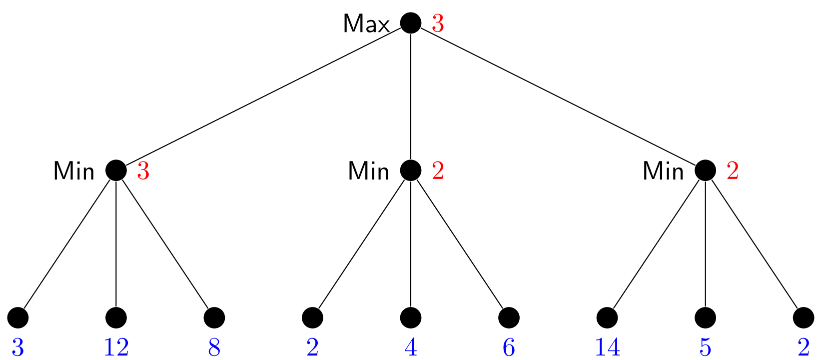

Max −∞
Min 3
3
12
Max −∞
Min 3
3
12
8
Max 3
Min 3
3
12
8
Max 3
Min 3
Min ∞
3
12
8


Max 3
Min 3
Min ∞
3
12
8
2
Max 3
Min 3
Min ∞
3
12
8
2
Max 3
Min 3
Min 2
3
12
8
2
4
Max 3
Min 3
Min 2
3
12
8
2
4
6


Max 3
Min 3
Min 2
Min ∞
3 12 8 2 4 6
Max 3
Min 3
Min 2
Min ∞
3 12 8 2 4 6 14
Max 3
Min 3
Min 2
Min 14
3 12 8 2 4 6 14
Max 3
Min 3
Min 2
Min 5
3 12 8 2 4 6 14 5







- ▷ 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 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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22064`.
We now address the problem that minimax is infeasible in practice. As so often, the solution is to eschew optimal strategies and to approximate them. In this case, instead of a computed utility function, we estimate one that is easy to compute: the evaluation function.

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 7.3.1.** An evaluation function f maps game states to numbers:

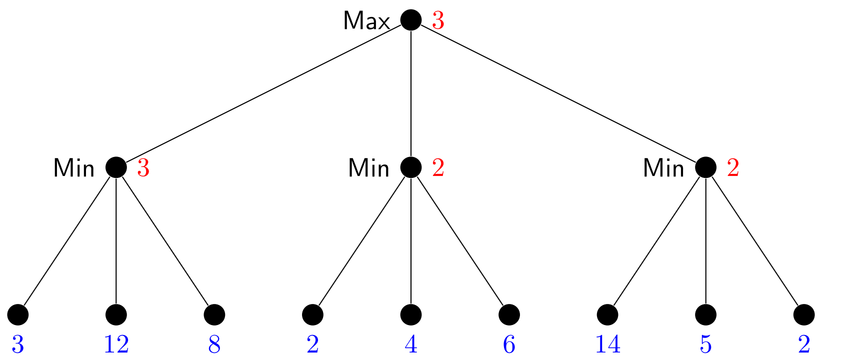
- ▷ $f(s)$ is an estimate of the actual value of s (as would be computed by unlimited-depth minimax for s).
- ▷ If cut-off state is terminal: Just use $\hat{u}$ instead of f.

▷ Analogy to heuristic functions (cf. ??): We want f to be both (a) accurate and (b) fast.

▷ Another analogy: (a) and (b) are in contradiction $\leadsto$ need to trade-off accuracy against overhead.

- ▷ In typical game playing algorithms today, f is inaccurate but very fast. (usually no good methods known for computing accurate f)

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 ⇝ 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)

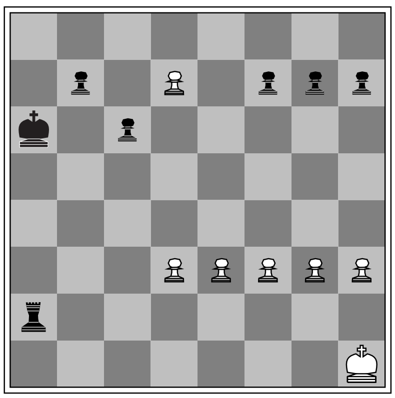
Linear Evaluation Functions

- **Problem:** How to come up with evaluation functions?
- **Definition 7.3.2.** A common approach is to use a weighted linear function for f, i.e. given a sequence of features $f_i\colon S\to\mathbb{R}$ and a corresponding sequence of weights $w_i\in\mathbb{R}$, f is of the form $f(s):=w_1\cdot f_1(s)+w_2\cdot f_2(s)+\cdots+w_n\cdot f_n(s)$
- **Problem:** How to obtain these weighted linear functions?
  - Weights w_i can be learned automatically. (learning agent)
  - The features f_i, however, have to be designed by human experts.
- **Note:** Very fast, very simplistic.
- **Observation:** Can be computed incrementally: In transition $s\xrightarrow{a}s'$, adapt $f(s)$ to $f(s')$ by considering only those features whose values have changed.

This assumes that the features (their contribution towards the actual value of the state) are independent. That's usually not the case (e.g. the value of a rook depends on the pawn structure).

The Horizon Problem

- **Problem:** Critical aspects of the game can be cut off by the horizon. We call this the horizon problem.
- **Example 7.3.3.**



Black to move

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

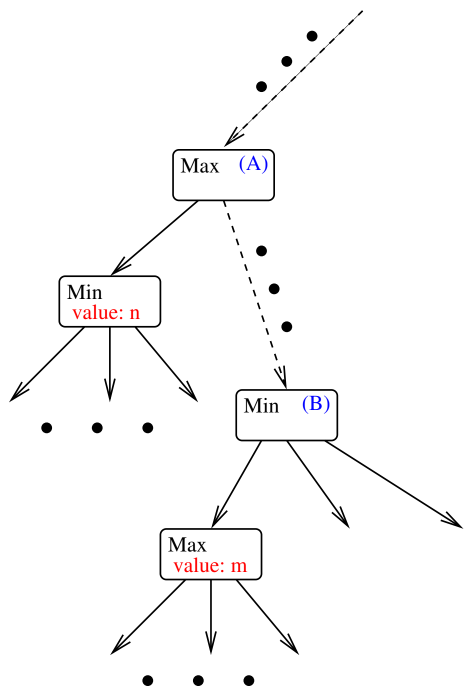
So, How Deeply to Search?

- **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 7.3.4 (Better Solution).** The quiescent search algorithm uses a dynamically adapted search depth d: It searches more deeply in unquiet positions, where value of evaluation function changes a lot in neighboring states.
- **Example 7.3.5.** In quiescent search for chess:
  - piece exchange situations ("you take mine, I take yours") are very unquiet
  - $\leadsto$ Keep searching until the end of the piece exchange is reached.

7.4 Alpha-Beta Search

We have seen that evaluation functions can overcome the combinatorial explosion induced by minimax search. But we can do even better: certain parts of the minimax search tree can be safely ignored, since we can prove that they will only sub-optimal results. We discuss the technique of alphabeta-pruning in detail as an example of such pruning methods in search algorithms.

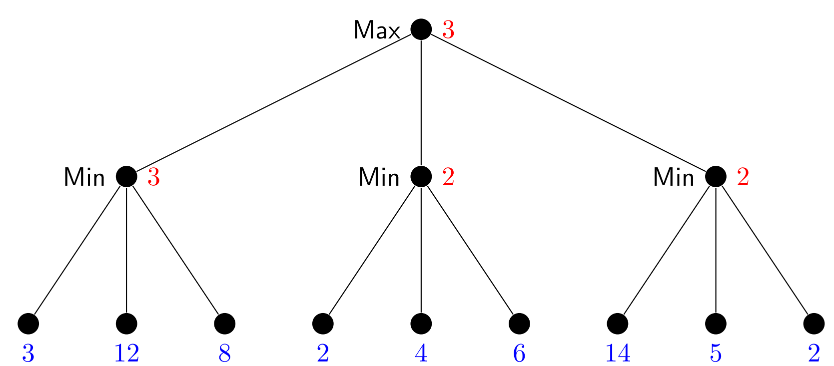
When We Already Know We Can Do Better Than This



- 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).

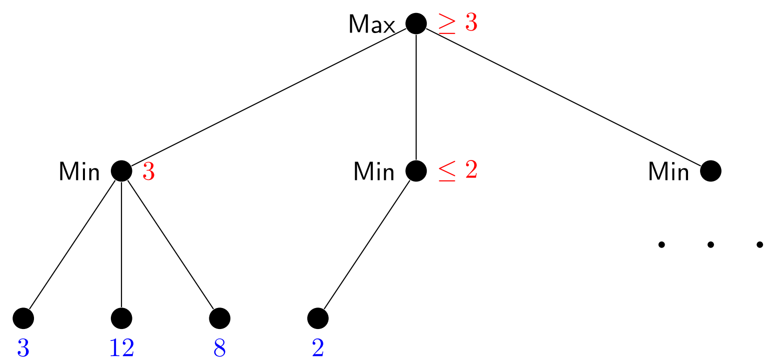
Alpha Pruning: Basic Idea

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



Alpha Pruning: Basic Idea (Continued)

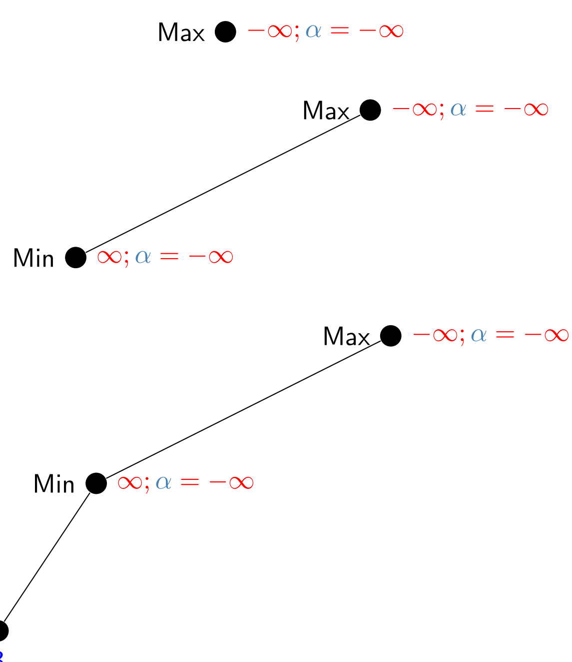
- **Answer:** Yes! We already know at this point that the middle action won't be taken by Max.

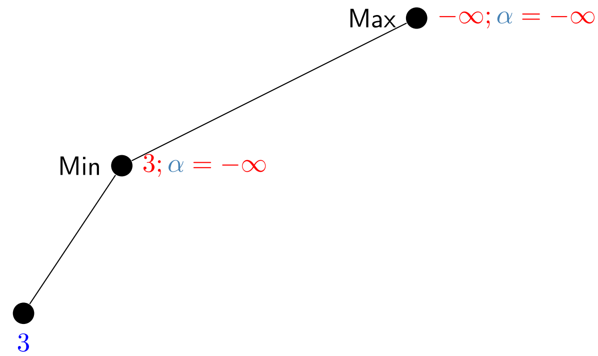


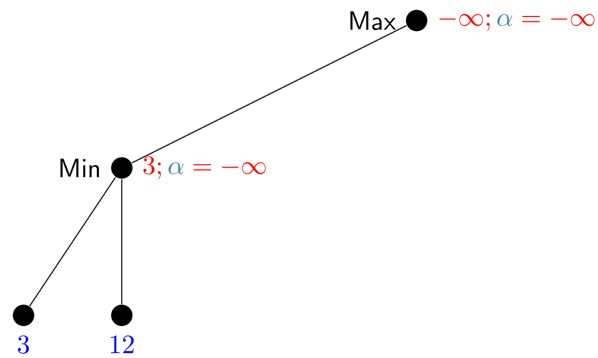
- **Idea:** We can use this to prune the search tree $\rightsquigarrow$ better algorithm

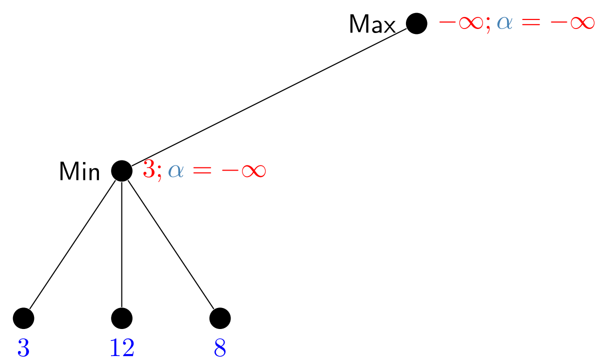
Alpha Pruning

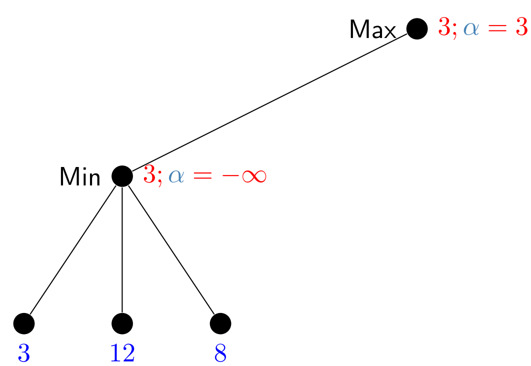
- **Definition 7.4.1.** For each node n in a minimax search tree, the alpha value $\alpha(n)$ is the highest Max-node utility that search has encountered on its path from the root to n.
- **Example 7.4.2 (Computing alpha values).**

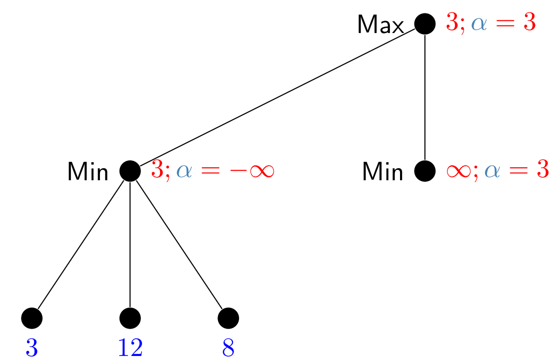


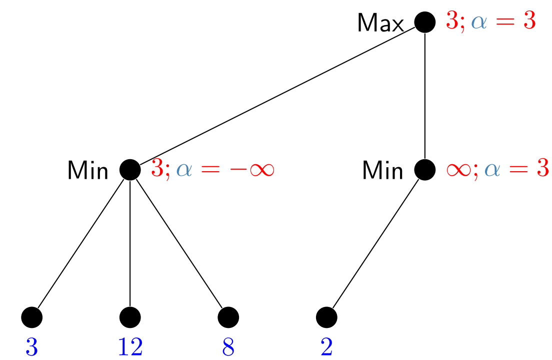

Max $-\infty; \alpha = -\infty$
Min $3; \alpha = -\infty$
3

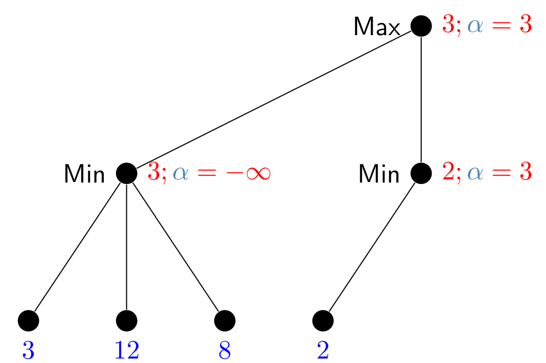

Max $-\infty; \alpha = -\infty$
Min $3; \alpha = -\infty$
3
12

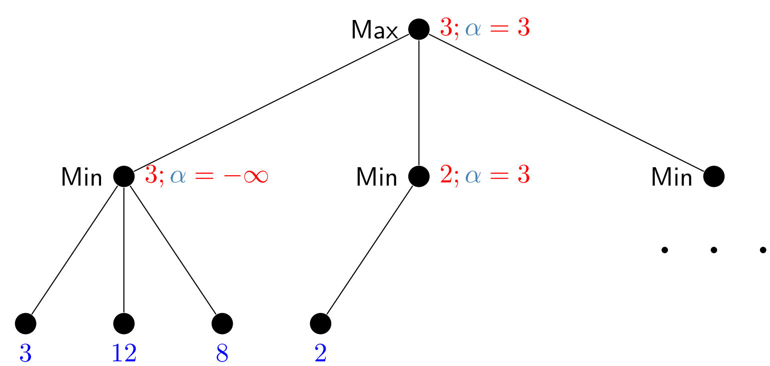

Max $-\infty; \alpha = -\infty$
Min $3; \alpha = -\infty$
3
12
8


Max $3; \alpha = 3$
Min $3; \alpha = -\infty$
3
12
8









▷ **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 7.4.3.** 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 7.4.4.** The alphabeta search algorithm is given by the following pseudocode

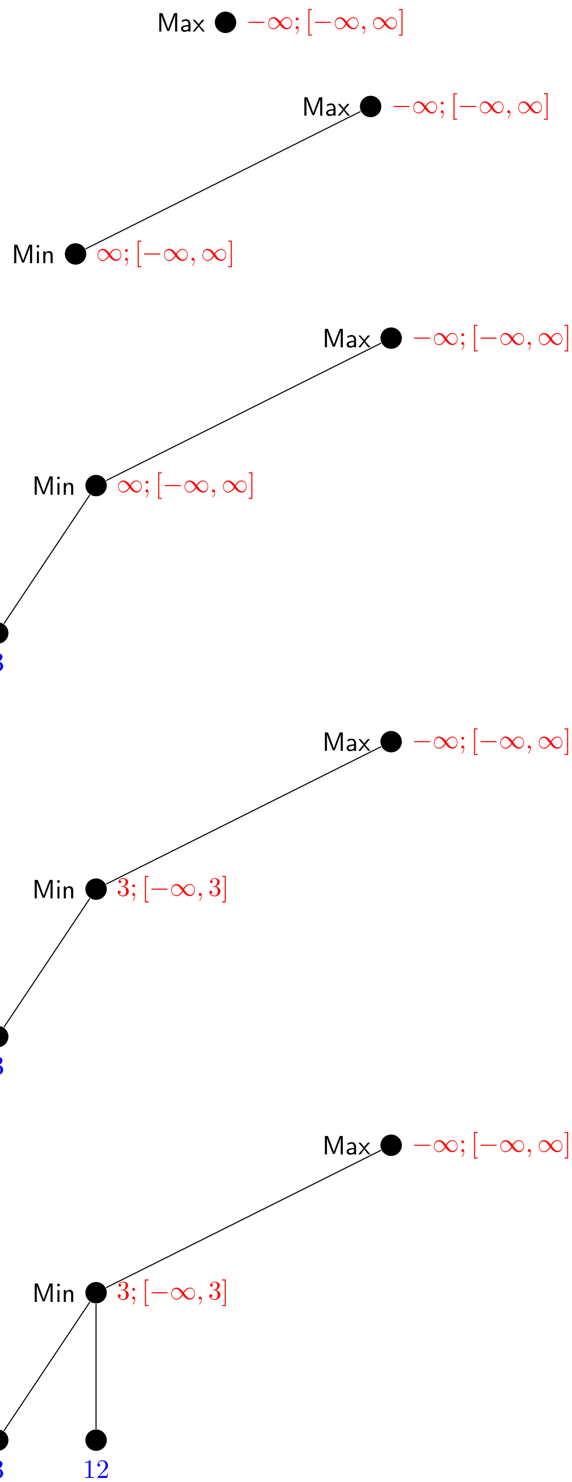
```
function Alpha—Beta—Search (s) returns an action
   v := Max—Value(s, −∞, +∞)
   return an action yielding value v in the previous function call

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
```

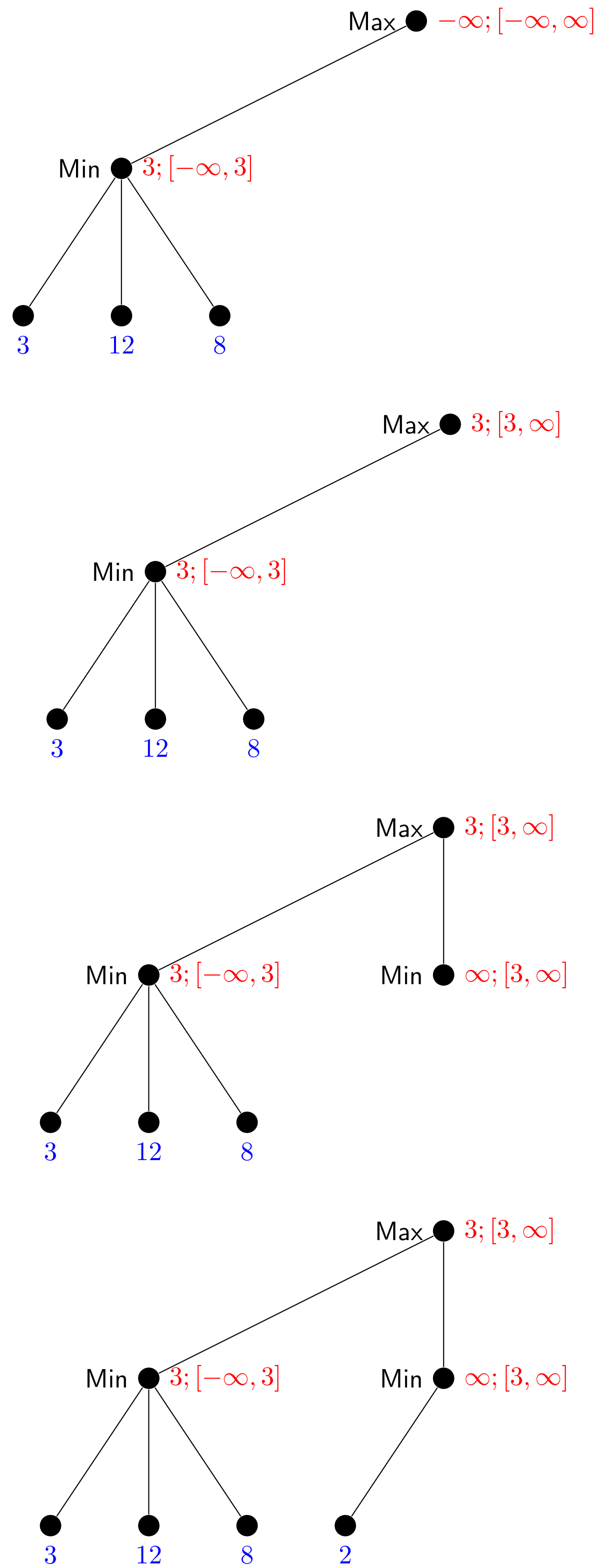
$\widehat{=}$ Minimax (slide 211) + α/β book-keeping and pruning.

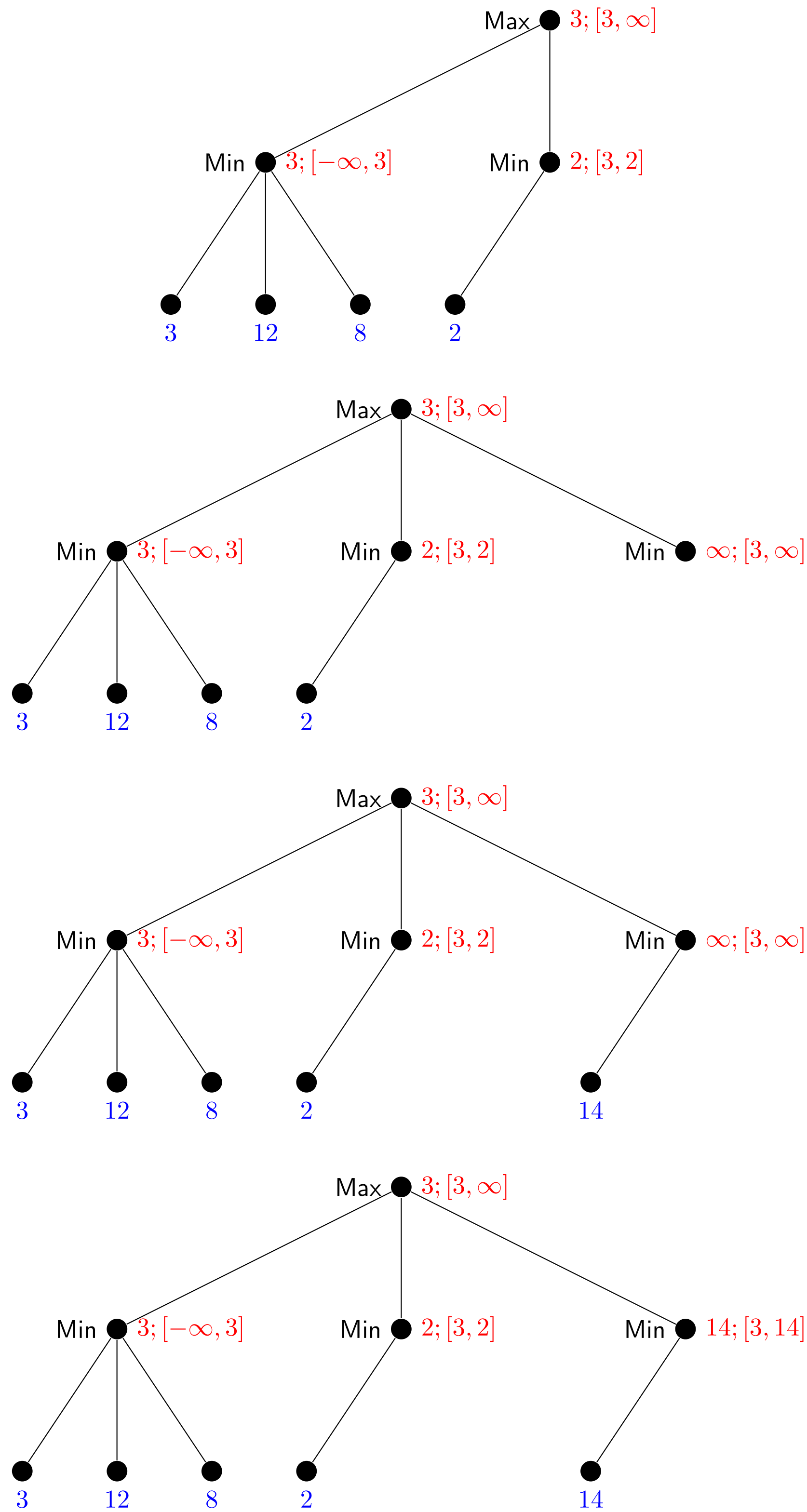
Note: Note that α only gets assigned a value in Max-nodes, and β only gets assigned a value in Min-nodes.

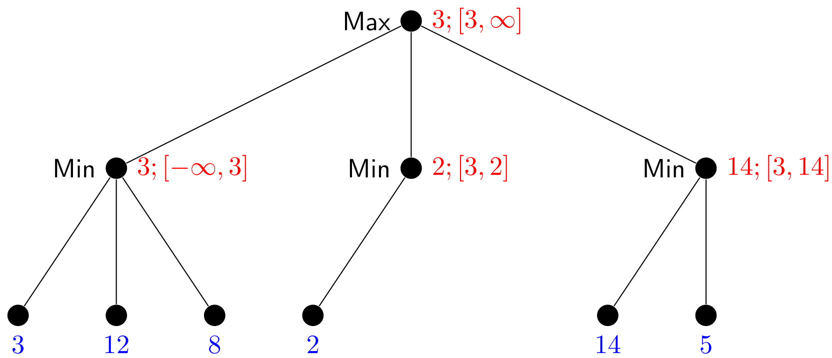
Alpha-Beta Search: Example

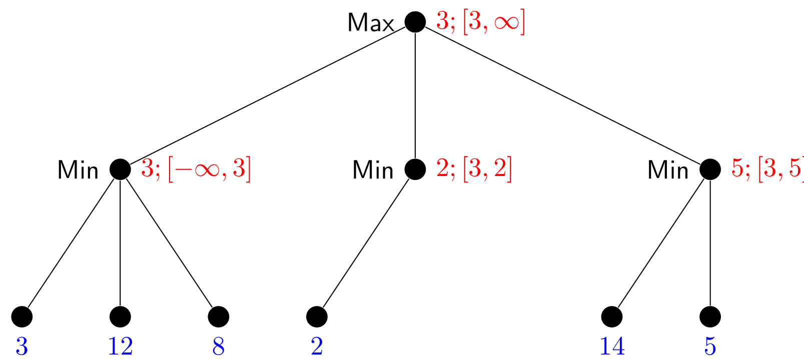
▷ **Notation:** $v; [\alpha, \beta]$

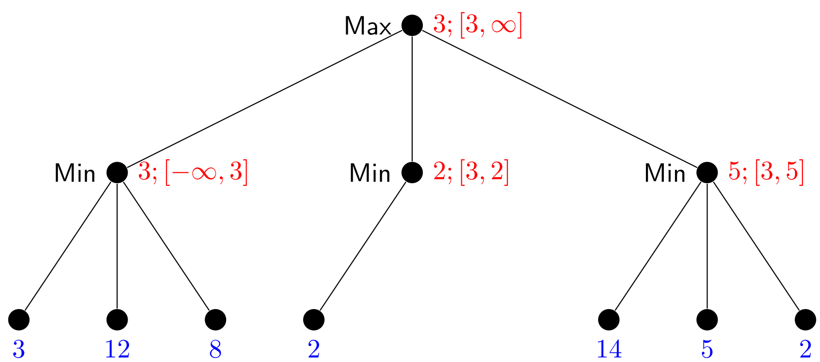


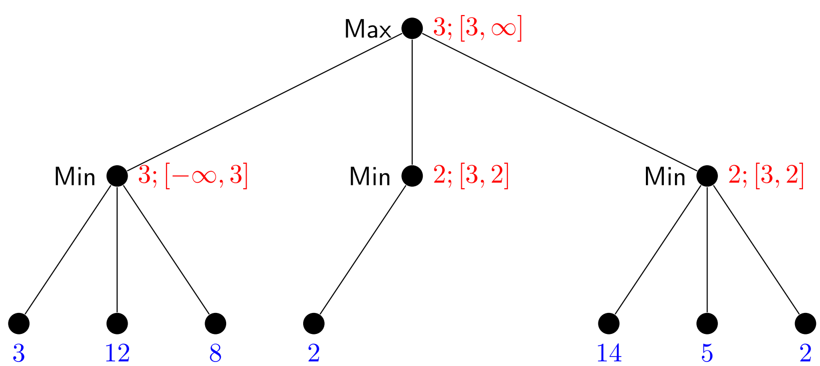

Max $-\infty; [-\infty, \infty]$
Min $3; [-\infty, 3]$
3 12 8
Max $3; [3, \infty]$
Min $3; [-\infty, 3]$
3 12 8
Max $3; [3, \infty]$
Min $3; [-\infty, 3]$
Min $\infty; [3, \infty]$
3 12 8
Max $3; [3, \infty]$
Min $3; [-\infty, 3]$
Min $\infty; [3, \infty]$
3 12 8 2


Max
$3; [3, \infty]$
Min
$3; [-\infty, 3]$
Min
$2; [3, 2]$
3
12
8
2
Max
$3; [3, \infty]$
Min
$3; [-\infty, 3]$
Min
$2; [3, 2]$
Min
$\infty; [3, \infty]$
3
12
8
2
Max
$3; [3, \infty]$
Min
$3; [-\infty, 3]$
Min
$2; [3, 2]$
Min
$\infty; [3, \infty]$
3
12
8
2
14
Max
$3; [3, \infty]$
Min
$3; [-\infty, 3]$
Min
$2; [3, 2]$
Min
$14; [3, 14]$
3
12
8
2
14







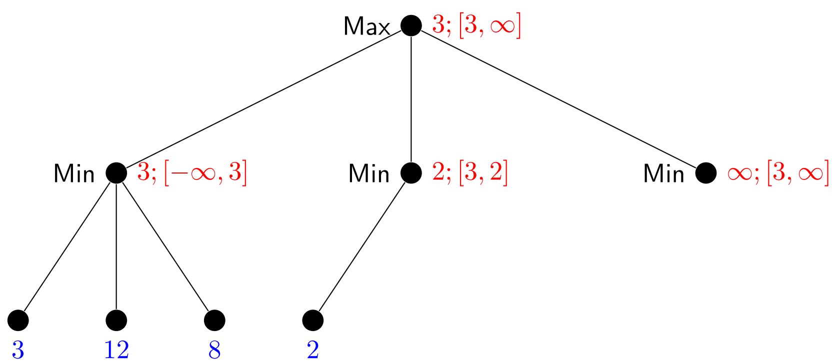


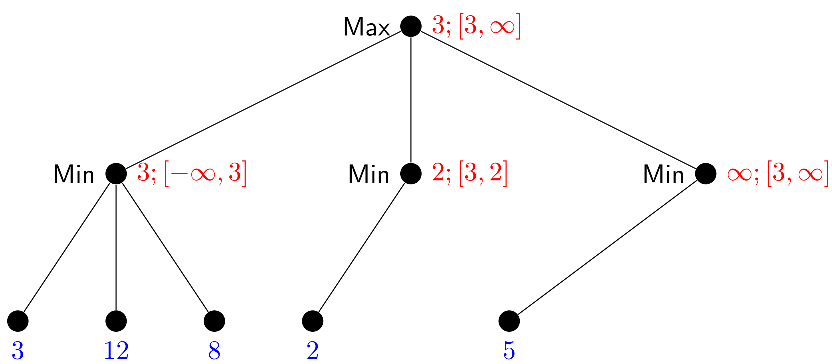
▷ **Note:** We could have saved work by choosing the opposite order for the successors

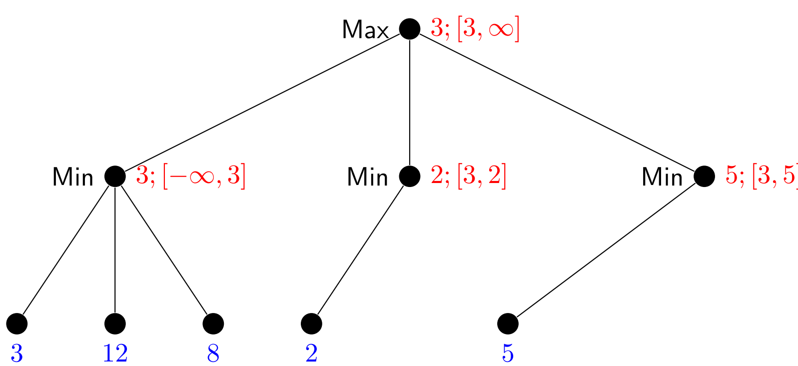
of the rightmost Min-node.
Choosing the best moves (for each of Max and Min) first yields more pruning!

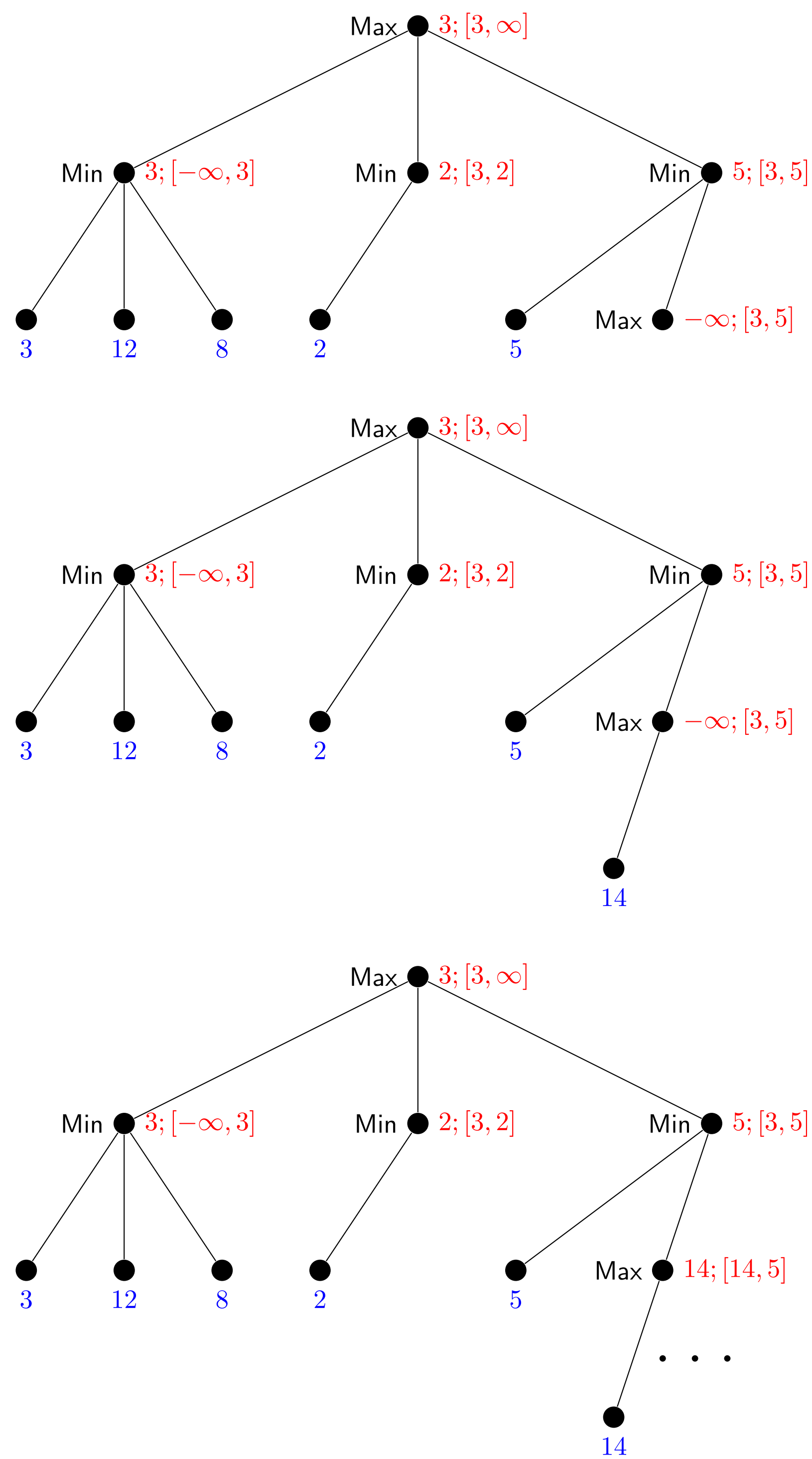
Alpha-Beta Search: Modified Example

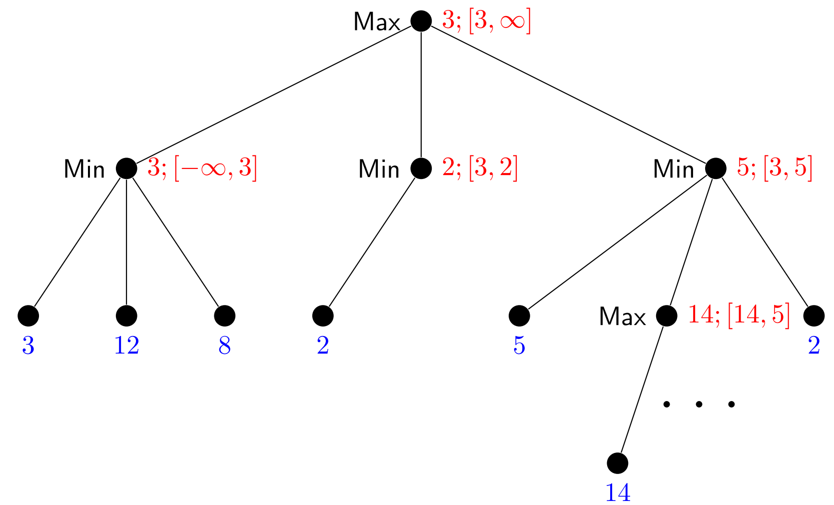
▷ **Showing off some actual β pruning:**

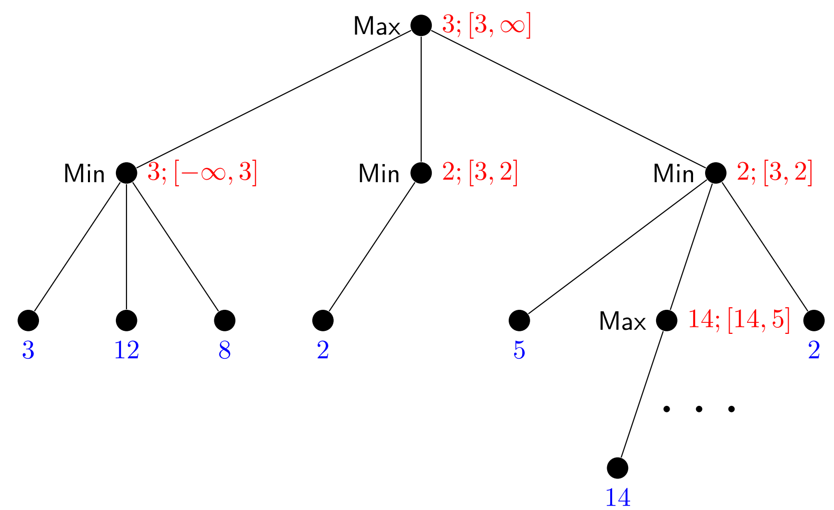







Max
3; [3, ∞]
Min
3; [−∞, 3]
Min
2; [3, 2]
Min
5; [3, 5]
Max
−∞; [3, 5]
3
12
8
2
5
Max
3; [3, ∞]
Min
3; [−∞, 3]
Min
2; [3, 2]
Min
5; [3, 5]
Max
−∞; [3, 5]
3
12
8
2
5
14
Max
3; [3, ∞]
Min
3; [−∞, 3]
Min
2; [3, 2]
Min
5; [3, 5]
Max
14; [14, 5]
3
12
8
2
5
14





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 7.4.5 (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.

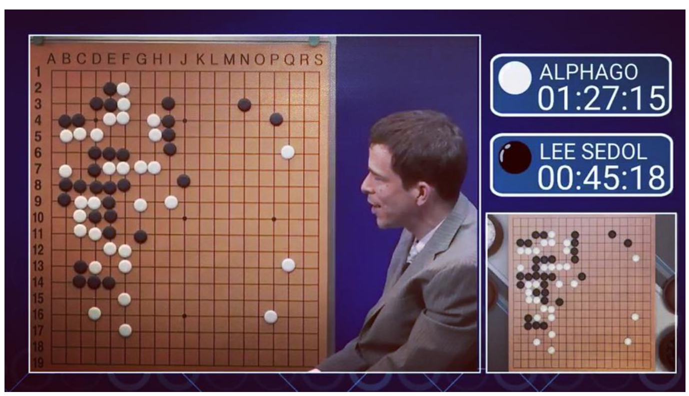
7.5 Monte-Carlo Tree Search (MCTS)

Video Nuggets covering this section can be found at https://fau.tv/clip/id/22259 and https://fau.tv/clip/id/22262.

We will now come to the most visible game-play program in recent times: The AlphaGo system for the game of go. This has been out of reach of the state of the art (and thus for alphabeta search) until 2016. This challenge was cracked by a different technique, which we will discuss in this section.

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 7.5.1.** 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 7.5.2.** For the Monte Carlo tree search algorithm (MCTS) we maintain a search tree T, the MCTS tree.

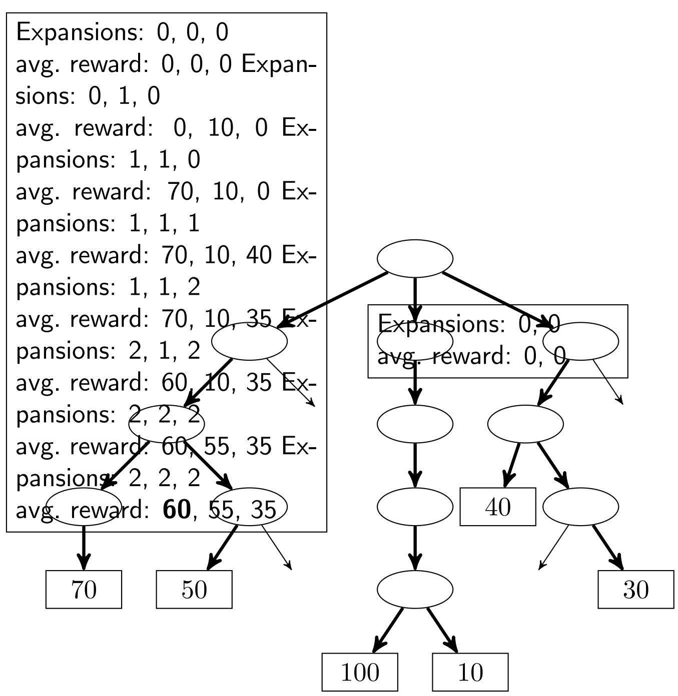
while time not up **do**
  apply actions within T **to** select a leaf state s'
  select action a' applicable **to** s', run random sample from a'
  add s' **to** T, update averages etc.
return an a **for** s with maximal average $u(t)$
When executing a, keep the part of T below a.

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

This looks only at a fraction of the search tree, so it is crucial to have good guidance *where to go*, i.e. which part of the search tree to look at.

Monte-Carlo Sampling: Illustration of Sampling

- **Idea:** Sample the search tree keeping track of the average utilities.
- **Example 7.5.3 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)

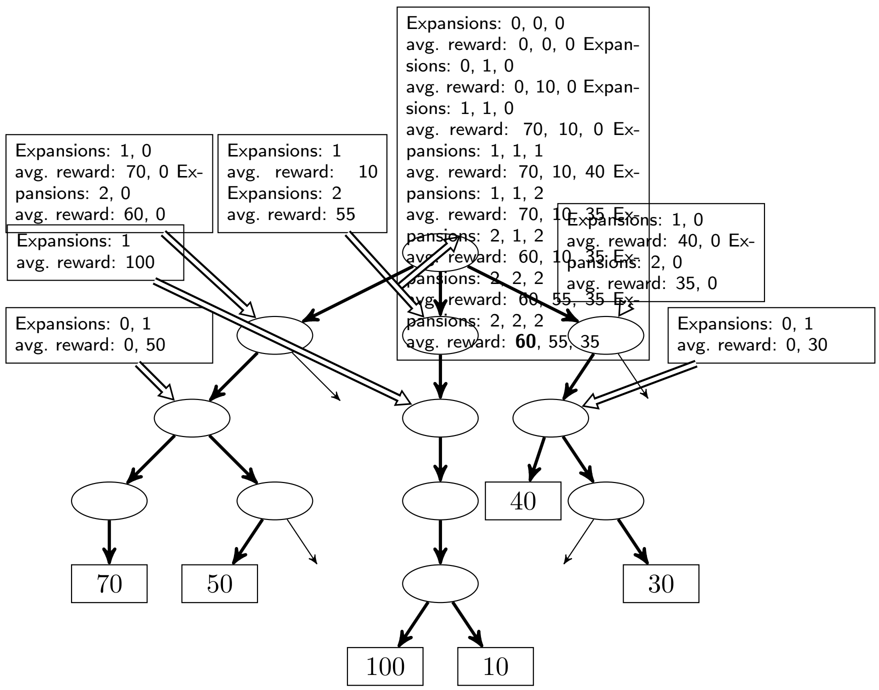


The sampling goes middle, left, right, right, left, middle. Then it stops and selects the highest-average action, 60, left. After first sample, when values in initial state are being updated, we have the following "expansions" and "avg. reward fields": small number of expansions favored for exploration: visit parts of the tree rarely visited before, what is out there? avg. reward: high values favored for exploitation: focus on promising parts of the search tree.

Monte-Carlo Tree Search: Building the Tree

▷ **Idea:** We can save work by building the tree as we go along.

▷ **Example 7.5.4 (Redoing the previous example).**



This is the exact same search as on previous slide, but incrementally building the search tree, by always keeping the first state of the sample. The first three iterations middle, left, right, go to show the tree extension; do point out here that, like the root node, the nodes added to the tree have expansions and avg reward counters for every applicable action. Then in next iteration right, after 30 leaf node was found, an important thing is that the averages get updated *along the entire path*, i.e., not only in the root as we did before, but also in the nodes along the way. After all six iterations have been done, as before we select the action left, value 60; but we keep the part of the tree below that action, "saving relevant work already done before".

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 7.5.5 (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:**

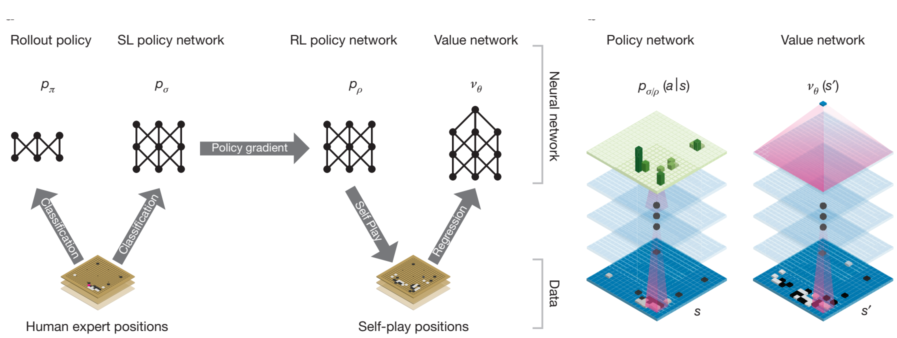


Illustration taken from [Sil+16] .

- ▷ 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").

Comments on the Figure:

a A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set.

b Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

Neural Networks + MCTS in AlphaGo

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

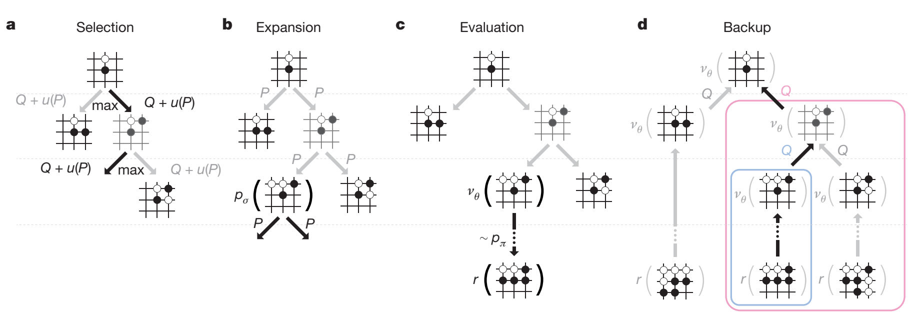


Illustration taken from [Sil+16]

- ▷ *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.

Comments on the Figure:

a Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge.

b The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action.

c At the end of a simulation, the leaf node is evaluated in two ways:

- using the value network v_θ,
- and by running a rollout to the end of the game

with the fast rollout policy $p\ \pi$, then computing the winner with function r.

d Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

AlphaGo, Conclusion?: This is definitely a great achievement!

- "Search + neural networks" looks like a great formula for general problem solving.
- expect to see lots of research on this in the coming decade(s).
- The AlphaGo design is quite intricate (architecture, learning workflow, training data design, neural network architectures, . . .).
- How much of this is reusable in/generalizes to other problems?
- Still lots of human expertise in here. Not as much, like in chess, about the game itself. But rather, in the design of the neural networks + learning architecture.

7.6 State of the Art

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22250`.

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.

FAU Michael Kohlhase: Artificial Intelligence 1 238 2025-02-06

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))

FAU Michael Kohlhase: Artificial Intelligence 1 239 2025-02-06

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))

FAU Michael Kohlhase: Artificial Intelligence 1 240 2025-02-06

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").

Suggested Reading:

- *Chapter 5: Adversarial Search*, Sections 5.1 – 5.4 [RN09].
  - Section 5.1 corresponds to my "Introduction", Section 5.2 corresponds to my "Minimax Search", Section 5.3 corresponds to my "Alpha-Beta Search". I have tried to add some additional clarifying illustrations. RN gives many complementary explanations, nice as additional background reading.
  - Section 5.4 corresponds to my "Evaluation Functions", but discusses additional aspects relating to narrowing the search and look-up from opening/termination databases. Nice as additional background reading.
  - I suppose a discussion of MCTS and AlphaGo will be added to the next edition ...

Chapter 8

Constraint Satisfaction Problems

In the last chapters we have studied methods for "general problem", i.e. such that are applicable to all problems that are expressible in terms of states and "actions". It is crucial to realize that these states were atomic, which makes the algorithms employed (search algorithms) relatively simple and generic, but does not let them exploit the any knowledge we might have about the *internal structure of states*.

In this chapter, we will look into algorithms that do just that by progressing to factored states representations. We will see that this allows for algorithms that are many orders of magnitude more efficient than search algorithms.

To give an intuition for factored states representations we, we present some motivational examples in **??** and go into detail of the Waltz algorithm, which gave rise to the main ideas of constraint satisfaction algorithms in **??**. **??** and **??** define constraint satisfaction problems formally and use that to develop a class of backtracking/search based algorithms. The main contribution of the factored states representations is that we can formulate advanced search heuristics that guide search based on the structure of the states.

8.1 Constraint Satisfaction Problems: Motivation

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22251`.

A (Constraint Satisfaction) Problem

▷ **Example 8.1.1 (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 8.1.2.** A constraint satisfaction problem (CSP) is a triple $\langle V, D, C\rangle$ where

1. V is a finite set V of variables,
2. an V-indexed family $(D_v)_{v\in V}$ of domains, and
3. for some subsets $\{v_1, \ldots, v_k\} \subseteq V$ a constraint $C_{\{v_1,\ldots,v_k\}} \subset D_{v_1} \times \ldots \times D_{v_k}$.

A variable assignment $\varphi \in (v\in V) \to D_v$ is a solution for C, iff $\langle \varphi(v_1), \ldots, \varphi(v_k)\rangle \in C_{\{v_1,\ldots,v_k\}}$ for all $\{v_1, \ldots, v_k\} \subseteq V$.

▷ **Definition 8.1.3.** A CSP γ is called satisfiable, iff it has a solution: a total variable assignment φ that satisfies all constraints.

▷ **Definition 8.1.4.** The process of finding solutions to CSPs is called constraint solving.

▷ *Remark 8.1.5.* We are using factored representation for world states now!

▷ Allows useful *general-purpose* algorithms with more power than standard tree search algorithm.

Another Constraint Satisfaction Problem

▷ **Example 8.1.6 (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

⇝

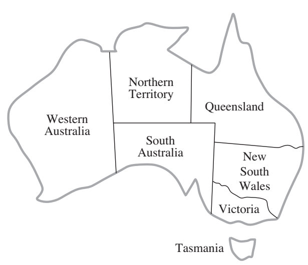
2	5	8	7	3	6	9	4	1
6	1	9	8	2	4	3	5	7
4	3	7	9	1	5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

- ▷ Variables: The 81 cells.
- ▷ Domains: Numbers $1, \ldots, 9$.
- ▷ Constraints: Each number only once in each row, column, block.

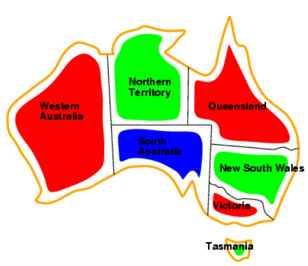
CSP Example: Map-Coloring

▷ **Definition 8.1.7.** 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 8.1.8 (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 $\langle \mathrm{WA}, \mathrm{NT} \rangle \in \{\langle \mathrm{red}, \mathrm{green} \rangle, \langle \mathrm{red}, \mathrm{blue} \rangle, \langle \mathrm{green}, \mathrm{red} \rangle, \ldots\}$



- ▷ **Intuition**: solutions map variables to domain values satisfying all constraints,
- ▷ e.g., $\{\mathrm{WA} = \mathrm{red}, \mathrm{NT} = \mathrm{green}, \ldots\}$

Bundesliga Constraints

▷ Variables: $v_{A\mathrm{vs}.B}$ 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]

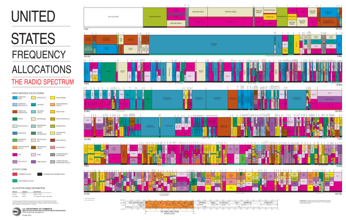
More Constraint Satisfaction Problems

Traveling Tournament Problem

Scheduling

Timetabling

Radio Frequency Assignment



1. U.S. Major League Baseball, 30 teams, each 162 games. There's one crucial additional difficulty, in comparison to Bundesliga. Which one? Travel is a major issue here!! Hence "Traveling Tournament Problem" in reference to the TSP.

2. This particular scheduling problem is called "car sequencing", how to most efficiently get cars through the available machines when making the final customer configuration (non-standard/flexible/custom extras).

3. Another common form of scheduling ...

4. The problem of assigning radio frequencies so that all can operate together without noticeable interference. Variable domains are available frequencies, constraints take form of $|x - y| > \delta_{xy}$, where delta depends on the position of x and y as well as the physical environment.

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.
- ▷ **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

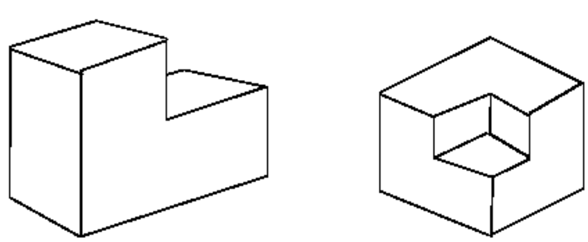
We will now have a detailed look at the problem (and innovative solution) that started the field of constraint satisfaction problems.

Background:

Adolfo Guzman worked on an algorithm to count the number of simple objects (like children's blocks) in a line drawing. David Huffman formalized the problem and limited it to objects in general position, such that the vertices are always adjacent to three faces and each vertex is formed from three planes at right angles (trihedral). Furthermore, the drawings could only have three kinds of lines: object boundary, concave, and convex. Huffman enumerated all possible configurations of lines around a vertex. This problem was too narrow for real-world situations, so Waltz generalized it to include cracks, shadows, non-trihedral vertices and light. This resulted in over 50 different line labels and thousands of different junctions. [ILD]

The Waltz Algorithm

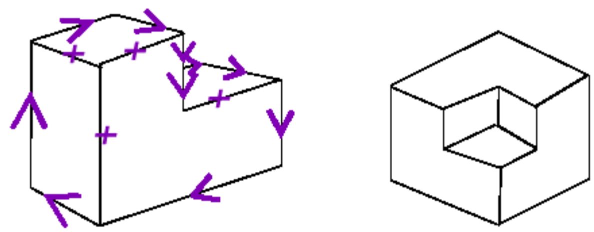
- ▷ **Remark:** One of the earliest examples of applied CSPs.
- ▷ **Motivation:** Interpret line drawings of polyhedra.



- ▷ **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.

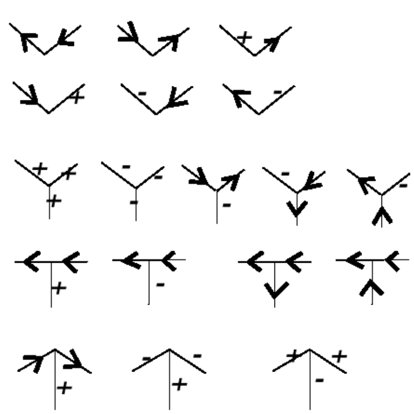
Waltz Algorithm on Simple Scenes

- **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 8.2.1.** *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 8.2.2.** *There are only 18 "legal" kinds of junctions:*

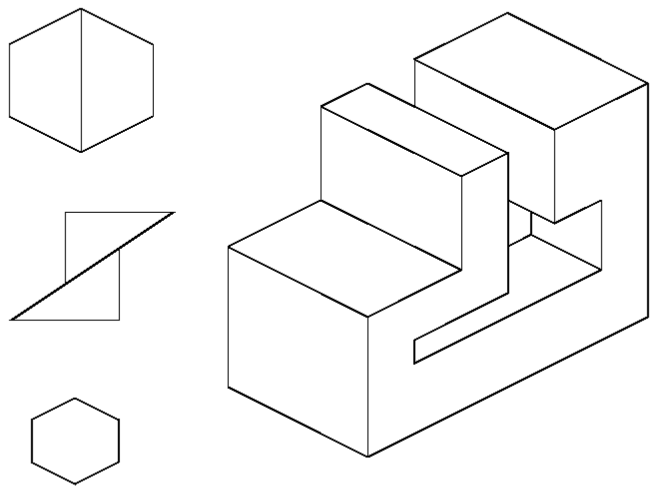


- **Idea:** given a representation of a diagram
  - label each junction in one of these manners (lots of possible ways)

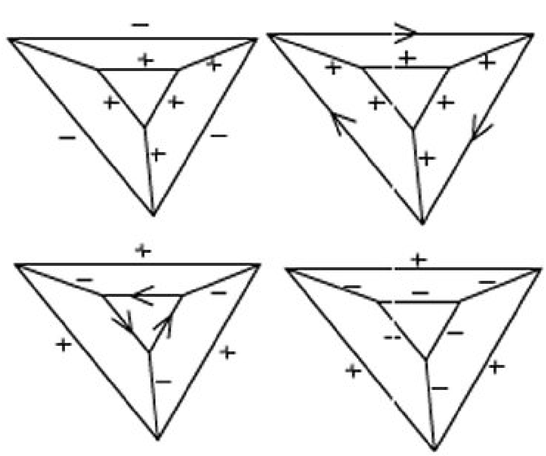
- ▷ junctions must be labeled, so that lines are labeled consistently
- ▷ **Fun Fact:** CSP always works perfectly! (early success story for CSP [Wal75])

Waltz's Examples

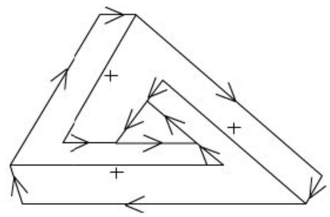
- ▷ In his dissertation 1972 [Wal75] David Waltz used the following examples



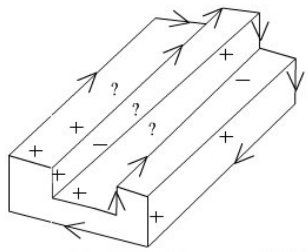
Waltz Algorithm (More Examples): Ambiguous Figures



Waltz Algorithm (More Examples): Impossible Figures



Consistent labelling for impossible figure



No consistent labelling possible

8.3 CSP: Towards a Formal Definition

We will now work our way towards a definition of CSPs that is formal enough so that we can define the concept of a solution. This gives use the necessary grounding to talk about algorithms later. **A Video Nugget** covering this section can be found at `https://fau.tv/clip/id/22277`.

Types of CSPs

- ▷ **Definition 8.3.1.** 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 8.3.2.** We call a CSP continuous, iff one domain is uncountable.
- ▷ **Example 8.3.3.** Start/end times for Hubble Telescope observations form a continuous CSP.
- ▷ **Theorem 8.3.4.** *Linear constraints solvable in poly time by linear programming methods.*

▷ **Theorem 8.3.5.** *There cannot be optimal algorithms for nonlinear constraint systems.*

Types of Constraints

▷ We classify the constraints by the number of variables they involve.

▷ **Definition 8.3.6.** Unary constraints involve a single variable, e.g., $\mathrm{SA} \neq green$.

▷ **Definition 8.3.7.** Binary constraints involve pairs of variables, e.g., $\mathrm{SA} \neq \mathrm{WA}$.

▷ **Definition 8.3.8.** 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 8.3.9.** Preferences (soft constraint) (e.g., red is better than green) are often representable by a cost for each variable assignment ⇝ constrained optimization problems.

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

▷ **Example 8.3.10 (Send More Money).** A student writes home:

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

Puzzle: letters stand for digits, addition should work out (parents send MONEY€)

▷ Variables: S, E, N, D, M, O, R, Y, each with domain $\{0, \ldots, 9\}$.

▷ Constraints:

1. all variables should have different values: $S \neq E$, $S \neq N$, ...
2. first digits are non-zero: $S \neq 0$, $M \neq 0$.
3. the addition scheme should work out: i.e. $1000 \cdot S + 100 \cdot E + 10 \cdot N + D + 1000 \cdot M + 100 \cdot O + 10 \cdot R + E = 10000 \cdot M + 1000 \cdot 0 + 100 \cdot N + 10 \cdot E + Y$.

BTW: The solution is $S \mapsto 9, E \mapsto 5, N \mapsto 6, D \mapsto 7, M \mapsto 1, O \mapsto 0, R \mapsto 8, Y \mapsto 2$ ⇝ parents send 10652€

▷ **Definition 8.3.11.** Problems like the one in ?? 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 8.3.12.** *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

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

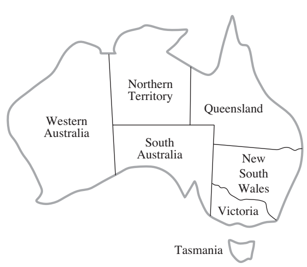
▷ **Problem:** The problem structure gets hidden. (search algorithms can get confused)

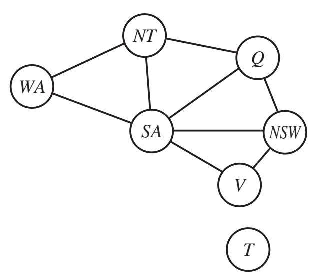
Constraint Graph

▷ **Definition 8.3.13.** A binary CSP is a CSP where each constraint is unary or binary.

▷ **Observation 8.3.14.** *A binary CSP forms a graph called the constraint graph whose nodes are variables, and whose edges represent the constraints.*

▷ **Example 8.3.15.** 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 8.3.16 (Assignment problems).** e.g., who teaches what class

▷ **Example 8.3.17 (Timetabling problems).** e.g., which class is offered when and where?

▷ **Example 8.3.18 (Hardware configuration).**

▷ **Example 8.3.19 (Spreadsheets).**

▷ **Example 8.3.20 (Transportation scheduling).**

▷ **Example 8.3.21 (Factory scheduling).**

▷ **Example 8.3.22 (Floorplanning).**

▷ **Note:** many real-world problems involve real-valued variables ⇝ continuous CSPs.

8.4 Constraint Networks: Formalizing Binary CSPs

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22279`.

Constraint Networks (Formalizing binary CSPs)

▷ **Definition 8.4.1.** A constraint network is a triple $\gamma := \langle V, D, C\rangle$, where

- ▷ V is a finite set of variables,
- ▷ $D := \{D_v \mid v \in V\}$ the set of their domains, and
- ▷ $C := \{C_{uv} \subseteq D_u \times D_v \mid u, v \in V \text{ and } u \neq v\}$ is a set of constraints with $C_{uv} = C_{vu}^{-1}$.

We call the undirected graph $\langle V, \{(u,v) \in V^2 \mid C_{uv} \neq D_u \times D_v\}\rangle$, the constraint graph of γ.

▷ We will talk of CSPs and mean constraint networks.

▷ **Remarks:** The mathematical formulation gives us a lot of leverage:

- ▷ $C_{uv} \subseteq D_u \times D_v \mathrel{\widehat{=}}$ possible assignments to variables u and v
- ▷ Relations are the most general formalization, generally we use symbolic formulations, e.g. "$u = v$" for the relation $C_{uv} = \{(a,b) \mid a = b\}$ or "$u \neq v$".
- ▷ We can express unary constraints C_u by restricting the domain of v: $D_v := C_v$.

Example: SuDoKu as a Constraint Network

▷ **Example 8.4.2 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as ??.

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

- ▷ Variables: $V = \{v_{ij} \mid 1 \leq i,j \leq 9\}$: v_{ij} =cell in row i column j.
- ▷ Domains For all $v \in V$: $D_v = D = \{1,\ldots,9\}$.
- ▷ Unary constraint: $C_{v_{ij}} = \{d\}$ if cell i,j is pre-filled with d.
- ▷ (Binary) constraint: $C_{v_{ij}v_{i'j'}} \mathrel{\widehat{=}}$ "$v_{ij} \neq v_{i'j'}$", i.e. $C_{v_{ij}v_{i'j'}} = \{(d,d') \in D \times D \mid d \neq d'\}$, for: $i = i'$ (same row), or $j = j'$ (same column), or $(\lceil \frac{i}{3} \rceil, \lceil \frac{j}{3} \rceil) = (\lceil \frac{i'}{3} \rceil, \lceil \frac{j'}{3} \rceil)$ (same block).

Note that the ideas are still the same as **??**, 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 8.4.3.** We call a partial function $a\colon V \rightharpoonup \bigcup_{u \in V} D_u$ a variable assignment if $a(u) \in D_u$ for all $u \in \mathbf{dom}(a)$.
- ▷ **Definition 8.4.4.** 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 $u,v \in \mathbf{dom}(a)$ 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 8.4.5.** The empty assignment ϵ is (trivially) consistent in any constraint network.
- ▷ **Definition 8.4.6.** Let f and g be variable assignments, then we say that f extends (or is an extension of) g, iff $\mathbf{dom}(g) \subset \mathbf{dom}(f)$ and $f|_{\mathbf{dom}(g)} = g$.
- ▷ **Definition 8.4.7.** We call a consistent (total) assignment a solution for γ and γ itself solvable or satisfiable.

How it all fits together

- ▷ **Lemma 8.4.8.** *Higher-order constraints can be transformed into equi-satisfiable* *binary constraints using auxiliary variables.*

▷ **Corollary 8.4.9.** *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 8.4.10.** *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

We now follow up on **??** to use search algorithms for solving constraint networks.

The key point of this section is that the factored states representations realized by constraint networks allow the formulation of very powerful heuristics. **A Video Nugget** covering this section can be found at `https://fau.tv/clip/id/22319`.

Standard search formulation (incremental)

▷ **Idea:** Every constraint network induces a single state problem.

▷ **Definition 8.5.1 (Let's do the math).** Given a constraint network $\gamma := \langle V, D, C\rangle$, then $\Pi_\gamma := \langle \mathcal{S}_\gamma, \mathcal{A}_\gamma, \mathcal{T}_\gamma, \mathcal{I}_\gamma, \mathcal{G}_\gamma\rangle$ is called the search problem induced by γ, iff

- ▷ State $\mathcal{S}_\gamma$ are variable assignments
- ▷ Action $\mathcal{A}_\gamma$: extend $\varphi \in \mathcal{S}_\gamma$ by a pair $x \mapsto v$ not conflicted with φ.
- ▷ Transition model $\mathcal{T}_\gamma(a, \varphi) = \varphi, x \mapsto v$ (extended assignment)
- ▷ Initial state $\mathcal{I}_\gamma$: the empty assignment ϵ.
- ▷ Goal states $\mathcal{G}_\gamma$: the total, consistent assignments

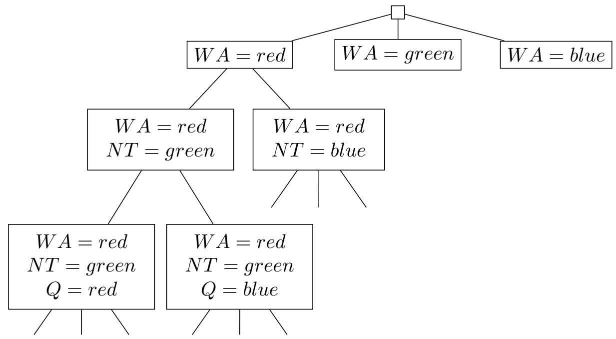
▷ **What has just happened?:** We interpret a constraint network γ as a search problem Π_γ. A solution to Π_γ induces a solution to γ.

▷ **Idea:** We have algorithms for that: e.g. tree search.

▷ **Remark:** This is the same for all CSPs! ☺
⇝ fail if no consistent assignments exist (not fixable!)

Standard search formulation (incremental)

▷ **Example 8.5.2.** A search tree for $\Pi_{Australia}$:



- **Observation:** Every solution appears at depth n with n variables.
- **Idea:** Use depth first search!
- **Observation:** Path is irrelevant ⇝ can use local search algorithms.
- 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 $\mathrm{WA} = \mathrm{red}$
  - ⇝ we only need to consider assignments to a single variable at each node
  - ⇝ $b = d$ and there are d^n leaves.
- **Definition 8.5.3.** Depth first search for CSPs with single-variable assignment extensions actions is called backtracking search.
- Backtracking search is the basic uninformed algorithm for CSPs.
- It can solve the n-queens problem for $\approx n, 25$.

Backtracking Search (Implementation)

- **Definition 8.5.4.** 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)
```

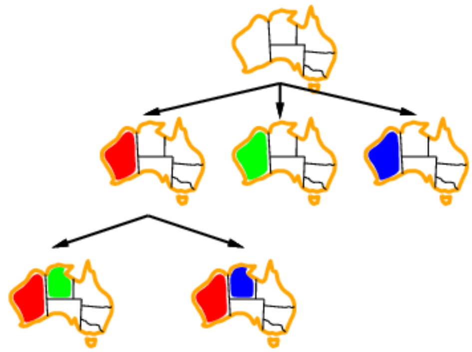
```
            if result ≠ failure then return result
            remove {var= value} from assignment
    return failure
```


Backtracking in Australia

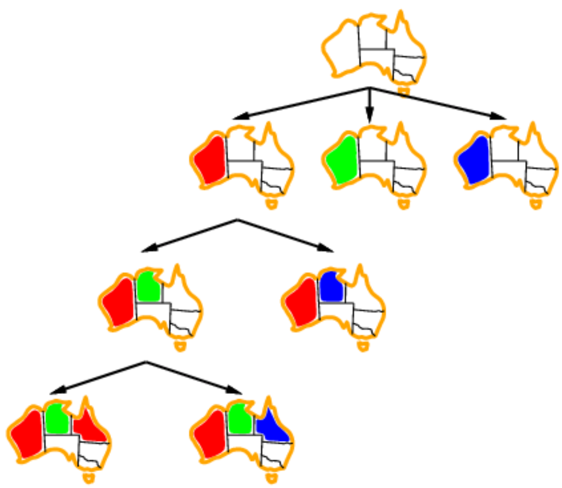
▷ **Example 8.5.5.** We apply backtracking search for a map coloring problem:

Step 1:

Step 2:



Step 3:



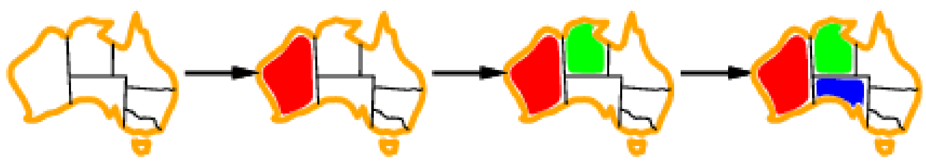
Step 4:

Improving Backtracking Efficiency

- ▷ General-purpose methods can give huge gains in speed for backtracking search.
- ▷ Answering the following questions well helps find powerful heuristics:
  1. Which variable should be assigned next? (i.e. a variable ordering heuristic)
  2. In what order should its values be tried? (i.e. a value ordering heuristic)
  3. Can we detect inevitable failure early? (for pruning strategies)
  4. Can we take advantage of problem structure? (⇝ inference)
- ▷ **Observation:** Questions 1/2 correspond to the missing subroutines Select—Unassigned—Variable and Order—Domain—Values from ??.

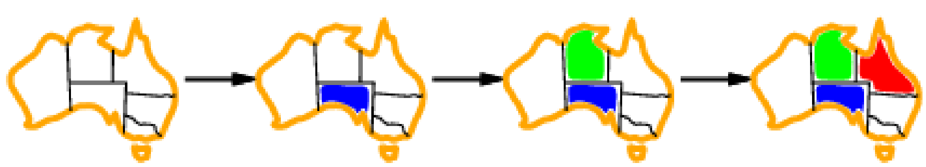
Heuristic: Minimum Remaining Values (Which Variable)

- **Definition 8.5.6.** 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 $\#(\{d \in D_v \,|\, a \cup \{v \mapsto d\}$ 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 $\#(\{d \in D_v \,|\, a \cup \{v \mapsto d\}$ is consistent$\}) = 1$, then the value assignment to v is forced by our previous choices.
- **Example 8.5.7.** In step 3 of **??**, 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 8.5.8.** 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 $\#(\{v \in (V \backslash \mathbf{dom}(a)) \,|\, C_{uv} \in C\})$ 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 8.5.9.**



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

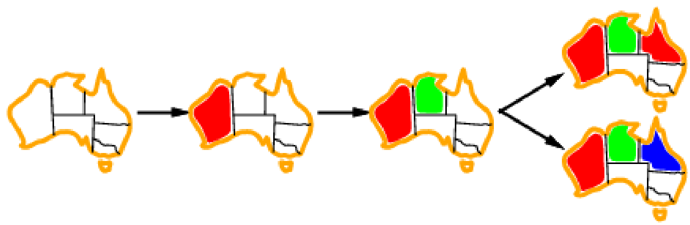
Where in **??** does the most constraining variable play a role in the choice? SA (only possible choice), NT (all choices possible except WA, V, T). Where in the illustration does most constrained variable play a role in the choice? NT (all choices possible except T), Q (only Q and WA possible).

Least Constraining Value Heuristic (Value Ordering)

▷ **Definition 8.5.10.** 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 $\#(\{e \in D_u \,|\, u \notin \mathbf{dom}(a),\ C_{uv} \in C, \text{ and } (e,d) \notin C_{uv}\})$

▷ By choosing the least constraining value first, we increase the chances to not rule out the solutions below the current node.

▷ **Example 8.5.11.**



▷ 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.

Suggested Reading: p

- *Chapter 6: Constraint Satisfaction Problems*, Sections 6.1 and 6.3, in [RN09].

– Compared to our treatment of the topic "Constraint Satisfaction Problems" (**??** and **??**), RN covers much more material, but less formally and in much less detail (in particular, my slides contain many additional in-depth examples). Nice background/additional reading, can't replace the lectures.

– Section 6.1: Similar to our "Introduction" and "Constraint Networks", less/different examples, much less detail, more discussion of extensions/variations.

– Section 6.3: Similar to my "Naïve Backtracking" and "Variable- and Value Ordering", with less examples and details; contains part of what we cover in **??** (RN does inference first, then backtracking). Additional discussion of *backjumping*.

Chapter 9

Constraint Propagation

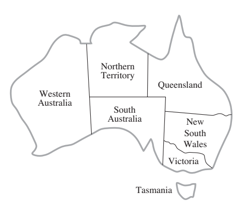
In this chapter we discuss another idea that is central to symbolic AI as a whole. The first component is that with the factored states representations, we need to use a representation language for (sets of) states. The second component is that instead of state-level search, we can graduate to representation-level search (inference), which can be much more efficient that state level search as the respective representation language actions correspond to groups of state-level actions.

9.1 Introduction

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22321`.

Illustration: Constraint Propagation

▷ **Example 9.1.1.** A constraint network γ:



▷ **Question:** Can we add a constraint without losing any solutions?

▷ **Example 9.1.2.** C_{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"
⇝ a smaller number of consistent (partial) variable assignments
⇝ more efficient search!

FAU Michael Kohlhase: Artificial Intelligence 1 277 2025-02-06

Illustration: Decomposition

▷ **Example 9.1.3.** 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 9.1.4 (Extreme case).** No inter-dependencies at all, as for Tasmania above.

FAU Michael Kohlhase: Artificial Intelligence 1 278 2025-02-06

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.

FAU Michael Kohlhase: Artificial Intelligence 1 279 2025-02-06

9.2 Constraint Propagation/Inference

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22326`.

Constraint Propagation/Inference: Basic Facts

▷ **Definition 9.2.1.** 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 9.2.2.** It's what you do all the time when playing SuDoKu:

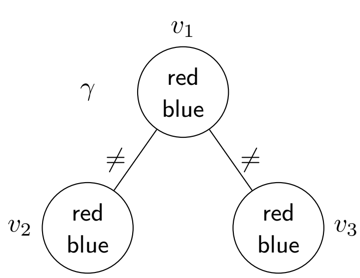
	5	8	7		6	9	4	1
		9	8		4	3	5	7
4		7	9		5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

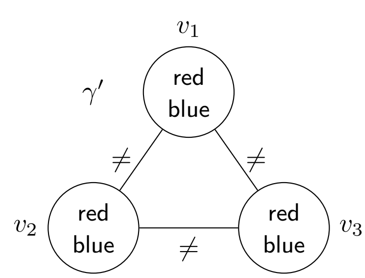
▷ **Formally:** Replace γ by an equivalent and strictly tighter constraint network γ'.

Equivalent Constraint Networks

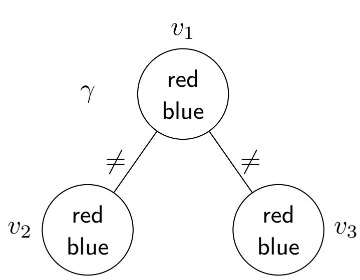
▷ **Definition 9.2.3.** We say that two constraint networks $\gamma := \langle V, D, C\rangle$ and $\gamma' := \langle V, D', C'\rangle$ sharing the same set of variables are equivalent, (write $\gamma' \equiv \gamma$), if they have the same solutions.

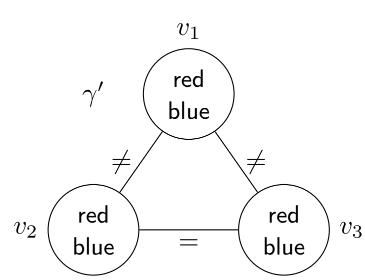
▷ **Example 9.2.4.**





Are these constraint networks equivalent? No.





Are these constraint networks equivalent? Yes.

Tightness

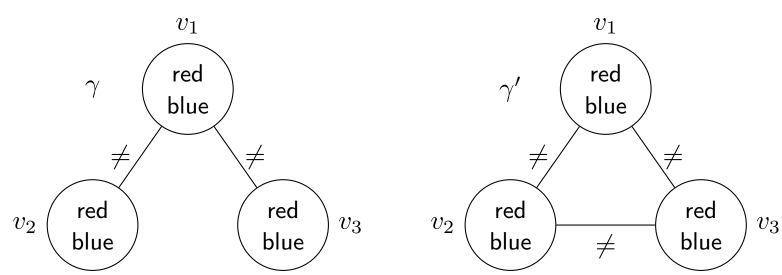
▷ **Definition 9.2.5 (Tightness).** Let $\gamma := \langle V, D, C \rangle$ and $\gamma' = \langle V, D', C' \rangle$ be constraint networks sharing the same set of variables, then γ' is tighter than γ, (write $\gamma' \sqsubseteq \gamma$), if:

(i) For all $v \in V$: ${D'}_v \subseteq D_v$.

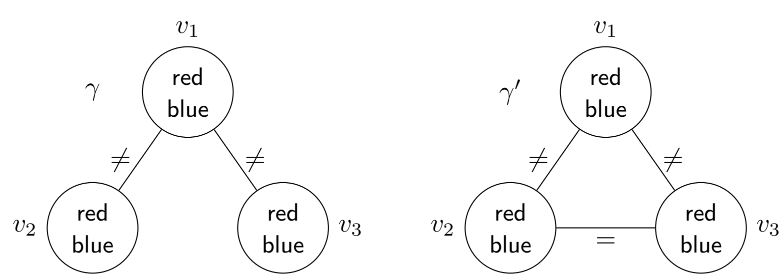
(ii) For all $u \neq v \in V$ and ${C'}_{uv} \in C'$: either ${C'}_{uv} \notin C$ or ${C'}_{uv} \subseteq C_{uv}$.

γ' is strictly tighter than γ, (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is proper.

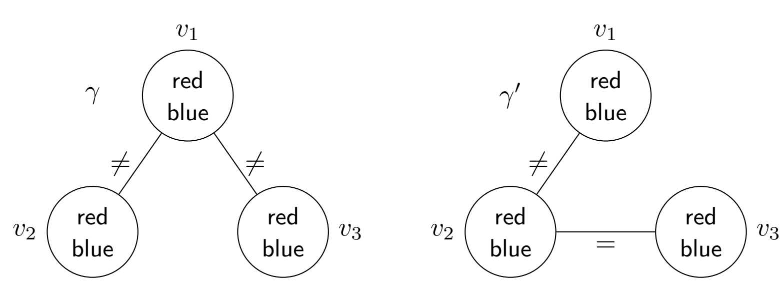
▷ **Example 9.2.6.**



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



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



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

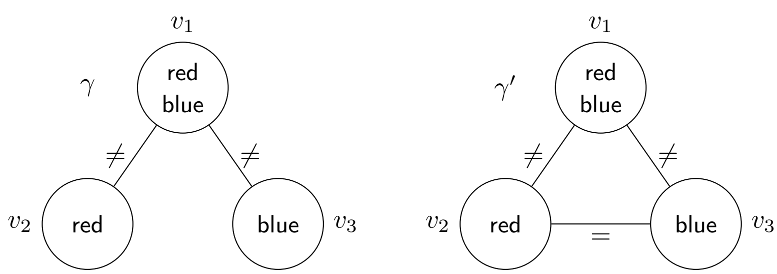
▷ **Intuition:** Strict tightness $\widehat{=}$ γ' *has the same constraints as* γ*, plus some.*

Equivalence + Tightness = Inference

▷ **Theorem 9.2.7.** *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*.*

▷ $\rightsquigarrow$ γ' is a better encoding of the underlying problem.

▷ **Example 9.2.8.** 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 9.2.9.** 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 /* γ' = (V_γ', D_γ', C_γ') */
5    γ' := Inference(γ')
6    if exists v with D_γ'v = ∅ then return ‘‘inconsistent’’
7    select some variable v for which a is not defined
8    for each d ∈ copy of D_γ'v in some order do
9      a' := a ∪ {v = d}; D_γ'v := {d} /* makes a explicit as a constraint */
10     a'' := BacktrackingWithInference(γ',a')
11     if a'' ≠ “inconsistent” then return a''
12   return ‘‘inconsistent’’
```

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

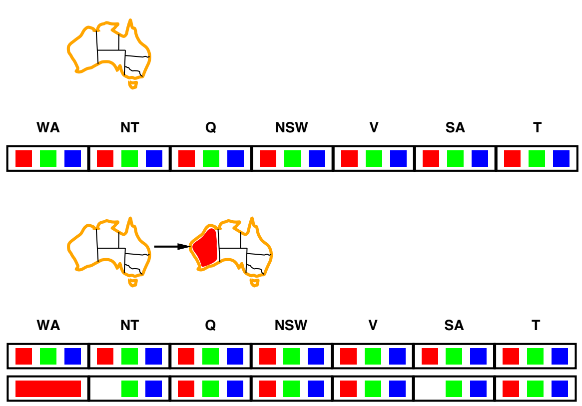
9.3 Forward Checking

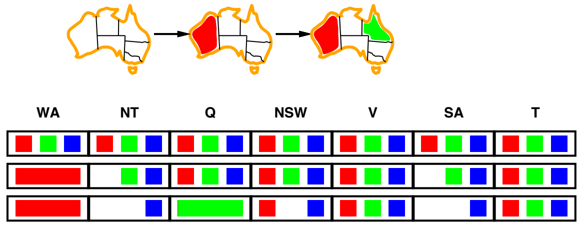
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22326`.

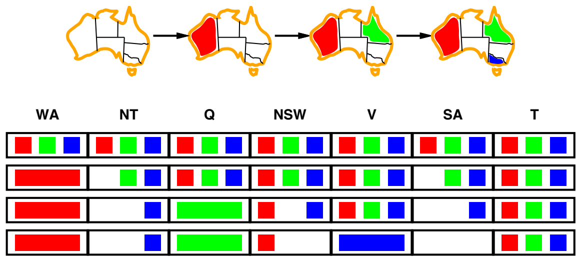
Forward Checking

▷ **Definition 9.3.1.** 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.

▷ **Example 9.3.2.** Forward checking in Australia







▷ **Definition 9.3.3 (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

Note: It's a bit strange that we start with d' here; this is to make link to arc consistency – coming up next – as obvious as possible (same notations u, and d vs. v and d').

Forward Checking: Discussion

▷ **Definition 9.3.4.** An inference procedure is called sound, iff for any input γ the output γ' have the same solutions.

▷ **Lemma 9.3.5.** *Forward checking is sound.*

Proof sketch: Recall here that the assignment a is represented as unary constraints inside γ.

▷ **Corollary 9.3.6.** *γ and γ' are equivalent.*

▷ Incremental computation: Instead of the first for-loop in ??, 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 9.3.7.** Let p and q be inference procedures, then p subsumes q, if $p(\gamma)\sqsubseteq q(\gamma)$ for any input γ.

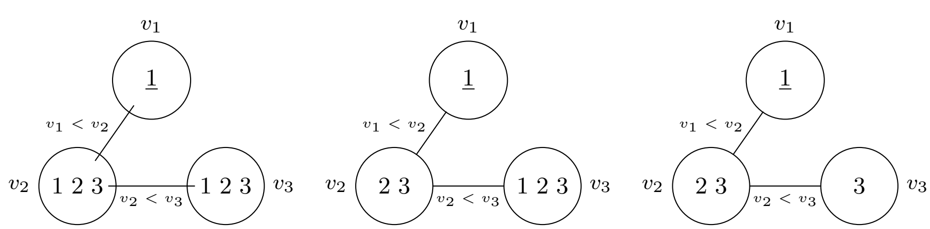
9.4 Arc Consistency

Video Nuggets covering this section can be found at `https://fau.tv/clip/id/22350` and `https://fau.tv/clip/id/22351`.

When Forward Checking is Not Good Enough

▷ **Problem:** Forward checking makes inferences only from assigned to unassigned variables.

▷ **Example 9.4.1.**



We could do better here: value 3 for v_2 is not consistent with any remaining value for v_3 $\leadsto$ it can be removed!

But forward checking does not catch this.

Arc Consistency: Definition

▷ **Definition 9.4.2 (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.

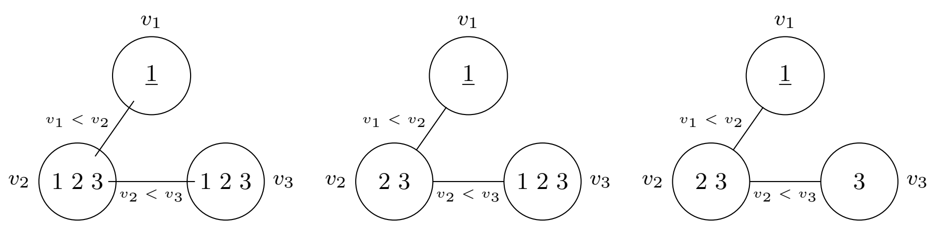
Arc Consistency: Example

▷ **Definition 9.4.3 (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 9.4.4 (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)

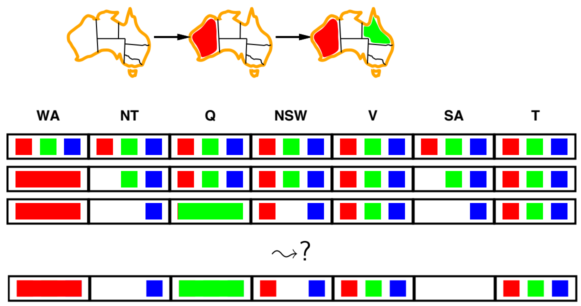
Arc Consistency: Example

▷ **Definition 9.4.5 (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 9.4.6.**



▷ **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 9.4.7.** $\mathrm{AC}(\gamma)$ *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 9.4.8.** $\mathrm{AC}(\gamma)$ *subsumes forward checking:* $\mathrm{AC}(\gamma) \sqsubseteq \mathrm{ForwardChecking}(\gamma)$.

▷ *Proof:* Recall from slide 282 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.

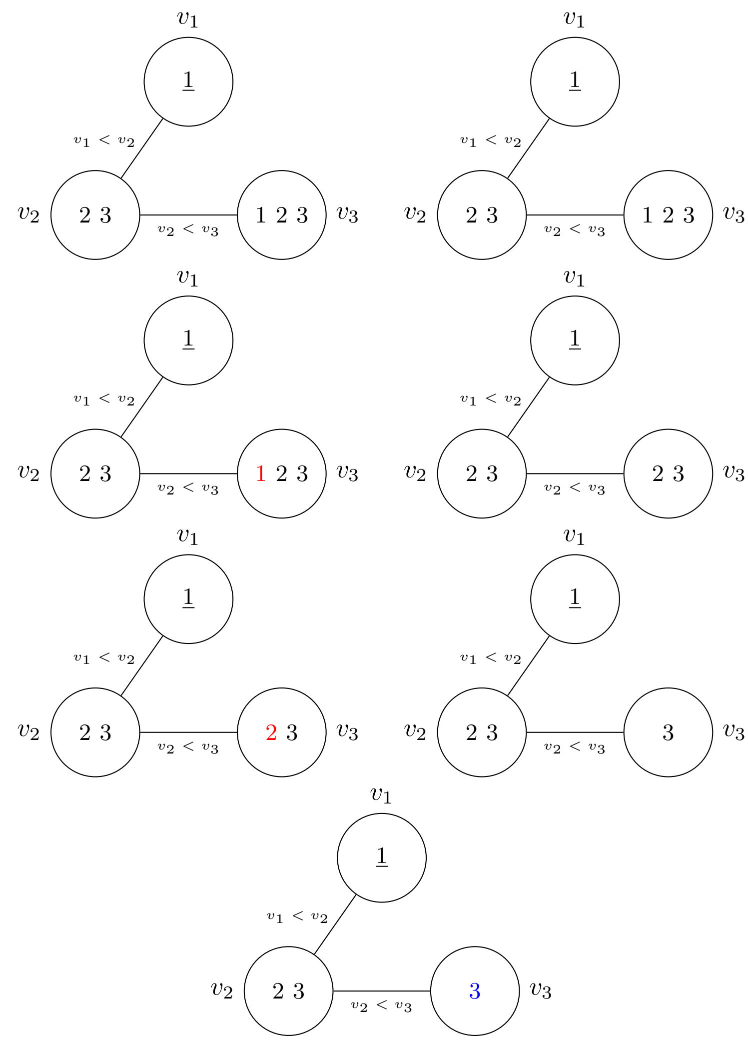
Enforcing Arc Consistency for *One* Pair of Variables

▷ **Definition 9.4.9 (Revise).** Revise is an algorithm enforcing arc consistency of u relative to v

```
function Revise(γ,u,v) returns modified γ
  for each d ∈ D_u do
    if there is no d' ∈ D_v with (d,d') ∈ C_uv then D_u := D_u\{d}
  return γ
```

▷ **Lemma 9.4.10.** *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 9.4.11.** $\mathrm{Revise}(\gamma, v_3, v_2)$



AC-1: Enforcing Arc Consistency (Version 1)

▷ **Idea:** Apply Revise pairwise up to a fixed point.

▷ **Definition 9.4.12.** AC-1 enforces arc consistency in constraint networks:

```
function AC−1(γ) returns modified γ
  repeat
    changesMade := False
    for each constraint C_uv 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 9.4.13.** *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 9.4.14.** 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?

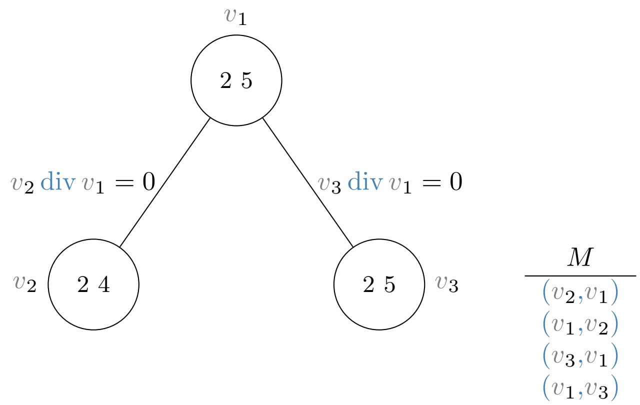
▷ **Answer:** At any time during the while-loop, if $(u,v) \notin M$ then u is arc consistent relative to v.

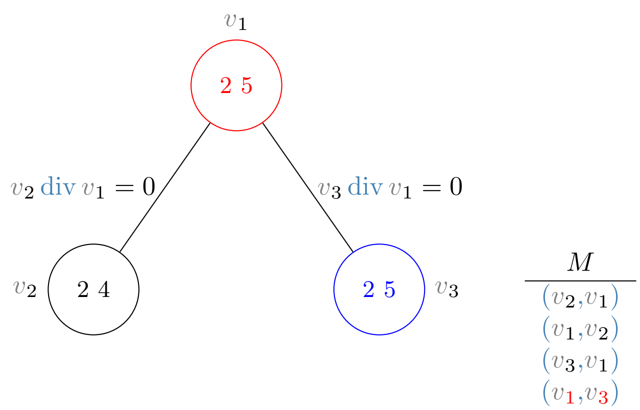
▷ **Question:** Why only "where $w \neq v$"?

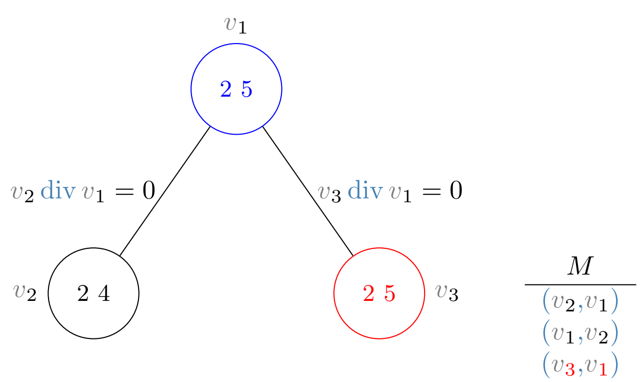
▷ **Answer:** If $w = v$ is the reason why D_u changed, then w is still arc consistent relative to u: the values just removed from D_u did not match any values from D_w anyway.

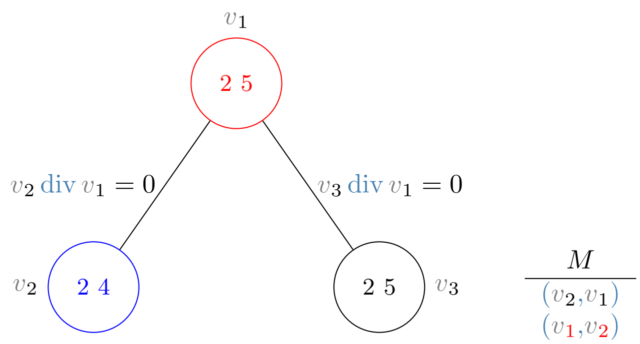
AC-3: Example

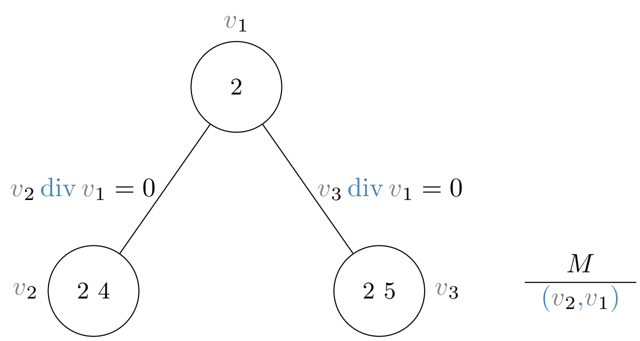
▷ **Example 9.4.15.** $y \operatorname{div} x = 0$: y modulo x is 0, i.e., y is divisible by x

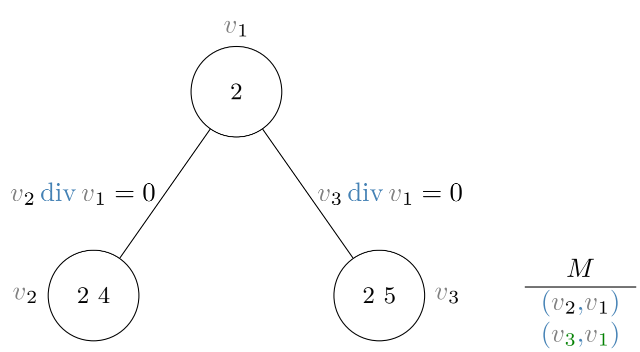

v1
2 5
v2 div v1 = 0
v3 div v1 = 0
v2
2 4
2 5
v3
M
(v2,v1)
(v1,v2)
(v3,v1)
(v1,v3)

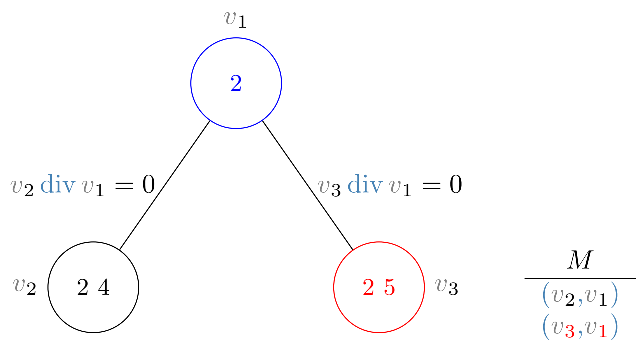

v1
2 5
v2 div v1 = 0
v3 div v1 = 0
v2
2 4
2 5
v3
M
(v2,v1)
(v1,v2)
(v3,v1)
(v1,v3)

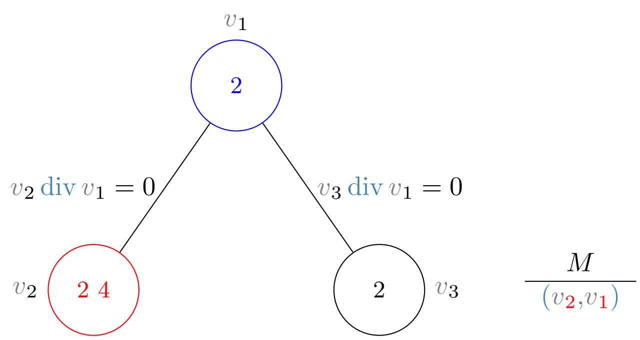

v1
2 5
v2 div v1 = 0
v3 div v1 = 0
v2
2 4
2 5
v3
M
(v2,v1)
(v1,v2)
(v3,v1)

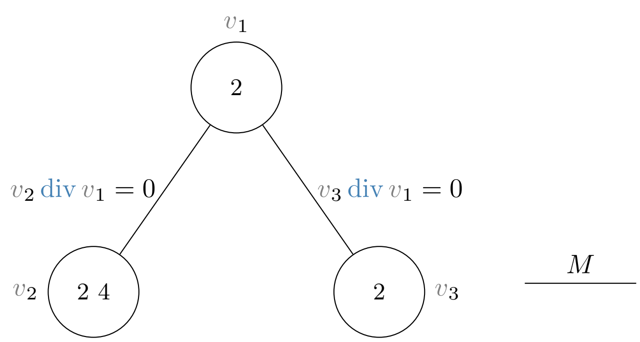

v_1
2 5
v_2 div $v_1 = 0$
v_3 div $v_1 = 0$
v_2
2 4
2 5
v_3
M
(v_2, v_1)
(v_1, v_2)


v_1
2
v_2 div $v_1 = 0$
v_3 div $v_1 = 0$
v_2
2 4
2 5
v_3
M
(v_2, v_1)


v_1
2
v_2 div $v_1 = 0$
v_3 div $v_1 = 0$
v_2
2 4
2 5
v_3
M
(v_2, v_1)
(v_3, v_1)







AC-3: Runtime

- ▷ **Theorem 9.4.16 (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 Revise(γ, 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 Revise(γ, 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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22353`.

Reminder: The Big Picture

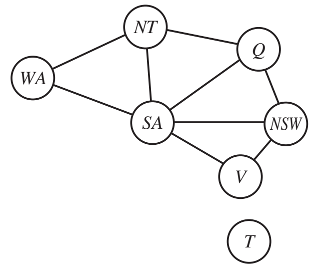
▷ Say γ is a constraint network with n variables and maximal domain size d.
  ▹ d^n total assignments must be tested in the worst case to solve γ.

▷ **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"?

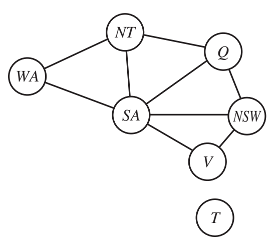
Problem Structure

- **Idea:** Tasmania and mainland are "independent subproblems"
- **Definition 9.5.1.** 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



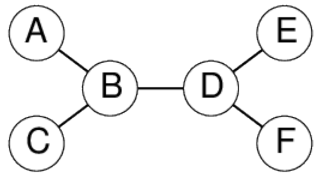
"Decomposition" 1.0: Disconnected Constraint Graphs

- **Theorem 9.5.2 (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.*
- *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 9.5.3.** Color Tasmania separately in Australia



- **Example 9.5.4 (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: $2^{28} \approx 280.000.000$.
- **Definition 9.5.5.** The process of decomposing a constraint network into components is called decomposition. There are various decomposition algorithms.

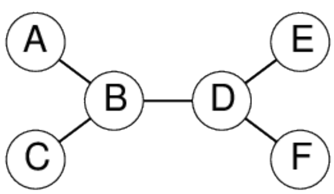
Tree-structured CSPs



- **Definition 9.5.6.** We call a CSP tree-structured, iff its constraint graph is acyclic
- **Theorem 9.5.7.** *Tree-structured CSP can be solved in $\mathcal{O}(nd^2)$ time.*
- Compare to general CSPs, where worst case time is $\mathcal{O}(d^n)$.
- This property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and the complexity of reasoning.

Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



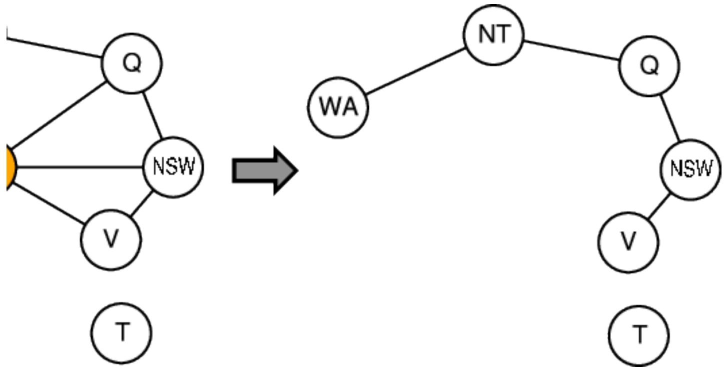
2. For j from n down to 2, apply

    RemoveInconsistent(Parent(X_j,X_j))

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

Nearly tree-structured CSPs

- **Definition 9.5.8.** Conditioning: instantiate a variable, prune its neighbors' domains.
- **Example 9.5.9.**



▷ **Definition 9.5.10.** 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.

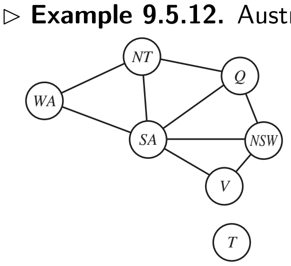
"Decomposition" 2.0: Acyclic Constraint Graphs

▷ **Theorem 9.5.11 (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 9.5.12.** Australia is not acyclic. (But see next section)



▷ **Example 9.5.13 (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 9.5.14.** Algorithm AcyclicCG(γ):

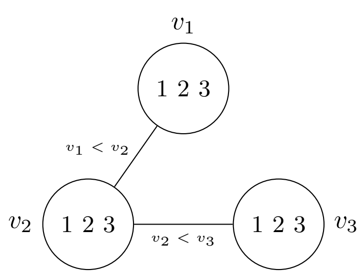
1. Obtain a (directed) tree from γ's constraint graph, picking an arbitrary variable v as the root, and directing edges outwards.[a]
2. Order the variables topologically, i.e., such that each node is ordered before its children; denote that order by $v_1, \ldots, v_n$.
3. **for** $i := n, n-1, \ldots, 2$ **do**:
   (a) Revise($\gamma, v_{parent(i)}, v_i$).
   (b) **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 $v_1, \ldots, v_n$.

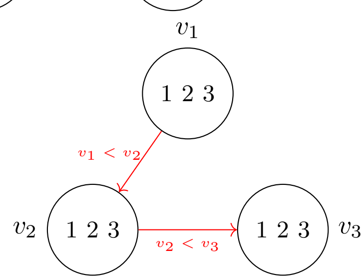
▷ **Lemma 9.5.15.** *This algorithm will find a solution without ever having to backtrack!*

[a]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

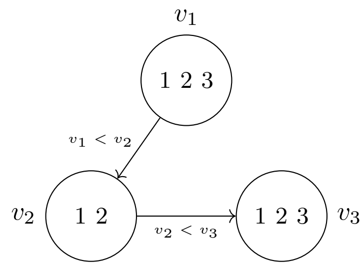
▷ **Example 9.5.16 (AcyclicCG() execution).**



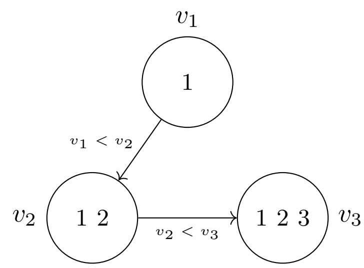


Input network γ.

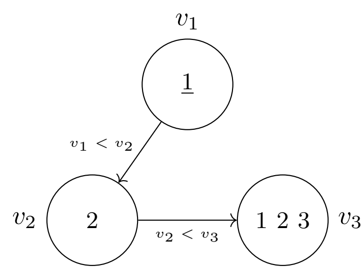
Step 1: Directed tree for root v_1.



Step 2: Order v_1, v_2, v_3.

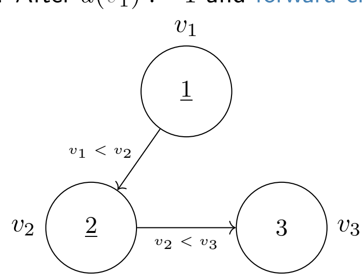


Step 3: After Revise(γ, v_2, v_3).

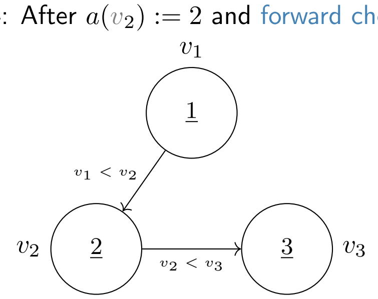


Step 3: After Revise(γ, v_1, v_2).

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



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



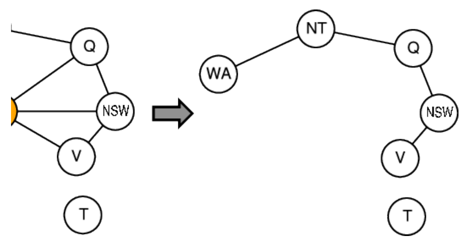
Step 4: After $a(v_3) := 3$ (and forward checking).

9.6 Cutset Conditioning

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22354`.

"Almost" Acyclic Constraint Graphs

- **Example 9.6.1 (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 9.6.2 (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 9.6.3.** The cutset conditioning algorithm, computes an optimal cutset, from γ and an existing cutset V_0.

function CutsetConditioning(γ,V_0,a) **returns** a solution, or "inconsistent"
  γ' := a copy of γ; γ' := ForwardChecking(γ',a)
  if ex. v with $D_{\gamma'_v} = \emptyset$ **then return** "inconsistent"
  if ex. $v \in V_0$ s.t. $a(v)$ is undefined **then** select such v
  else a' := AcyclicCG(γ');
  if $a' \neq$ "inconsistent" **then return** $a \cup a'$ **else return** "inconsistent"
  for each $d \in$ copy of $D_{\gamma'_v}$ **in** some order **do**
    $a' := a \cup \{v = d\}$; $D_{\gamma'_v} := \{d\}$;
    a'' := CutsetConditioning(γ',V_0,a')
  if $a'' \neq$ "inconsistent" **then return** a'' **else return** "inconsistent"

- Forward checking is required so that "$a \cup \mathrm{AcyclicCG}(\gamma')$" is consistent in γ.
- **Observation 9.6.4.** *Running time is exponential only in $\#(V_0)$, not in $\#(V)$!*
- *Remark 9.6.5.* Finding optimal cutsets is $\mathbf{NP}$ hard, but good approximations exist.

9.7 Constraint Propagation with Local Search

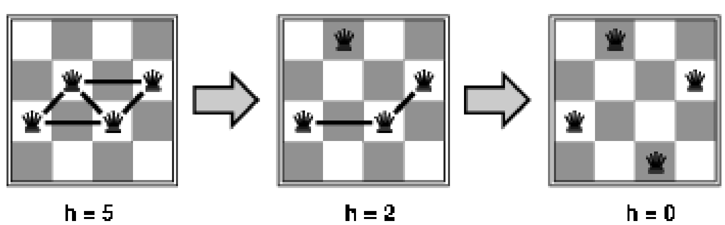
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22355`.

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

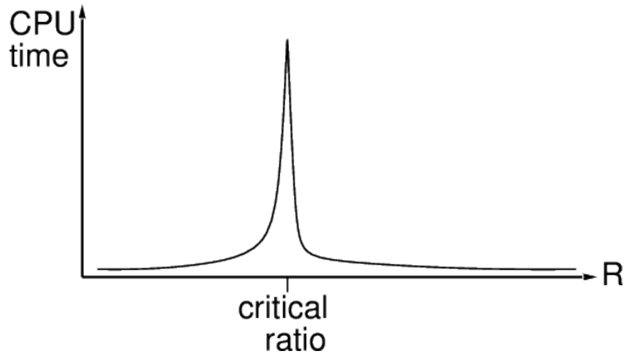
- ▷ States: 4 queens in 4 columns ($4^4 = 256$ states)
- ▷ Actions: Move queen in column
- ▷ Goal state: No conflicts
- ▷ Heuristic: $h(n) \widehat{=}$ number of conflict



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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22356`.

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.

Suggested Reading:

- *Chapter 6: Constraint Satisfaction Problems* in [RN09], in particular Sections 6.2, 6.3.2, and 6.5.
  - Compared to our treatment of the topic "constraint satisfaction problems" (**??** and **??**), RN covers much more material, but less formally and in much less detail (in particular, our slides contain many additional in-depth examples). Nice background/additional reading, can't replace the lectures.
  - Section 6.3.2: Somewhat comparable to our "inference" (except that equivalence and tightness are not made explicit in RN) together with "forward checking".
  - Section 6.2: Similar to our "arc consistency", less/different examples, much less detail, additional discussion of path consistency and global constraints.
  - Section 6.5: Similar to our "decomposition" and "cutset conditioning", less/different examples, much less detail, additional discussion of tree decomposition.

Part III

Knowledge and Inference

A Video Nugget covering this part can be found at `https://fau.tv/clip/id/22466`.

This part of the course introduces representation languages and inference methods for structured state representations for agents: In contrast to the atomic and factored state representations from **??**, we look at state representations where the relations between objects are not determined by the problem statement, but can be determined by inference-based methods, where the knowledge about the environment is represented in a formal langauge and new knowledge is derived by transforming expressions of this language.

We look at propositional logic – a rather weak representation langauge – and first-order logic – a much stronger one – and study the respective inference procedures. In the end we show that computation in Prolog is just an inference process as well.

Chapter 10

Propositional Logic & Reasoning, Part I: Principles

10.1 Introduction: Inference with Structured State Representations

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22455`.

State Representations in Agents and Algorithms

- **Recall:** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attribute and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.
- **Recall:** We have used atomic representations in search problems and tree search algorithms.
- **But:** We already allowed peeking into state in
  - informed search to compute heuristics
  - adversarial search ⇜ too many state!
- **Recall:** We have used factored representations in
  - backtracking search for CSPs ⇝ universally useful heuristics
  - constraint propagation: inference ⇝ lifting search to the CSP description level.
- **Up Next:** Inference for structured state representations.

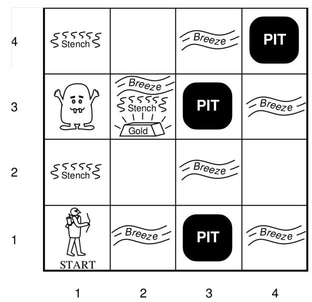
FAU Michael Kohlhase: Artificial Intelligence 1 314 2025-02-06

10.1.1 A Running Example: The Wumpus World

To clarify the concepts and methods for inference with structured state representations, we now introduce an extended example (the Wumpus world) and the agent model (logic-based agents) that use them. We will refer back to both from time to time below.

The Wumpus world is a very simple game modeled after the early text adventure games of the 1960 and 70ies, where the player entered a world and was provided with textual information about percepts and could explore the world via actions. The main difference is that we use it as an agent environment in this course.

The Wumpus World



Definition 10.1.1. The Wumpus world is a simple game where an agent explores a cave with 16 cells that can contain pits, gold, and the Wumpus with the goal of getting back out alive with the gold.
The agent cannot observe the locations of pits, gold, and the Wumpus, but some of their effects in the cell it currently visits.

▷ **Definition 10.1.2 (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 10.1.3 (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.

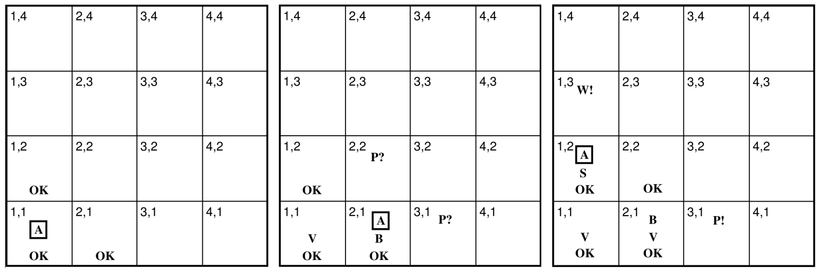
▷ **Definition 10.1.4 (Percepts).** The agent can experience the following percepts: stench, breeze, glitter, bump, scream, none.

- ▷ 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).

The game is complex enough to warrant structured state representations and can easily be extended to include uncertainty and non-determinism later.
As our focus is on inference processes here, let us see how a human player would reason when entering the Wumpus world. This can serve as a model for designing our artificial agents.

Reasoning in the Wumpus World

▷ **Example 10.1.5 (Reasoning in the Wumpus World).**
As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



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

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

▷ **Note:** The agent has more knowledge than just the percepts ⇝ inference!

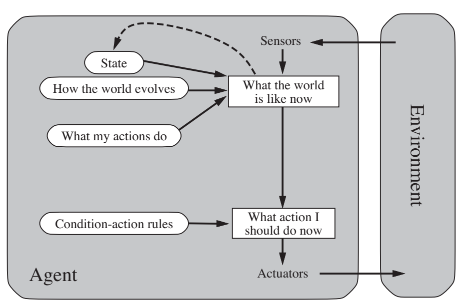
Let us now look into what kind of agent we would need to be successful in the Wumpus world: it seems reasonable that we should build on a model-based agent and specialize it to structured state representations and inference.

Agents that Think Rationally

▷ **Problem:** But how can we build an agent that can do the necessary inferences?

▷ **Idea:** Think Before You Act!
"Thinking" = Inference about knowledge represented using logic.

▷ **Definition 10.1.6.** 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. Agent schema:



The formal language of the logical system acts as a world description language.
Agent function:

function KB−AGENT ($percept$) **returns** an action
  persistent: KB, a knowledge base
    t, a counter, initially 0, indicating time
  TELL(KB, MAKE−PERCEPT−SENTENCE($percept$,t))
  $action$:= ASK(KB, MAKE−ACTION−QUERY(t))

```
TELL(KB, MAKE−ACTION−SENTENCE(action,t))
 t := t+1
return action
```

Its agent function maintains a knowledge base about the environment, which is updated with percept descriptions (formalizations of the percepts) and action descriptions. The next action is the result of a suitable inference-based query to the knowledge base.

10.1.2 Propositional Logic: Preview

We will now give a preview of the concepts and methods in propositional logic based on the Wumpus world before we formally define them below. The focus here is on the use of PL^0 as a world description language and understanding how inference might work.
We will start off with our preview by looking into the use of PL^0 as a world description language for the Wumpus world. For that we need to fix the language itself (its syntax) and the meaning of expressions in PL^0 (its semantics).

Logic: Basic Concepts (Representing Knowledge)

- ▷ We now preview some of the concepts involved in logic so that you have an intuition for the formal definitions below.
- ▷ **Definition 10.1.7.** Syntax: What are legal formulae $\mathbf{A}$ in the logic?
- ▷ **Example 10.1.8.** "W" and "$W \Rightarrow S$".
  ($W \widehat{=}$ *Wumpus is here*, $S \widehat{=}$ *it stinks*, $W \Rightarrow S \widehat{=}$ If W, then S)
- ▷ **Definition 10.1.9.** Semantics: Which formulae $\mathbf{A}$ are true?
- ▷ **Observation:** Whether $W \Rightarrow S$ is true depends on whether W and S are!
- ▷ **Idea:** Capture the state of W and S... in a variable assignment.
- ▷ **Definition 10.1.10.** For a variable assignment φ, write $\varphi \models \mathbf{A}$ if φ is true in the Wumpus world described by φ.
- ▷ **Example 10.1.11.** If $\varphi := \{W \mapsto \mathsf{T}, S \mapsto \mathsf{F}\}$, then $\varphi \models W$ but $\varphi \not\models (W \Rightarrow S)$.
- ▷ **Intuition:** Knowledge about the state of the world is described by formulae, interpretations evaluate them in the current world (they should turn out true!)
- ▷ **Definition 10.1.12.** The process of representing a natural language text in the formal language of a logical system is called formalization.
- ▷ **Observation:** Formalizing a NL text or utterance makes it machine-actionable. (the ultimate purpose of AI)
- ▷ **Observation:** Formalization is an art/skill, not a science!

It is critical to understand that while PL^0 as a logical system is given once and for all, the agent designer still has to formalize the situation (here the Wumpus world) in the world description language (here PL^0; but we will look at more expressive logical systems below). This formalization is the seed of the knowledge base, the logic-based agent can then add to via its percepts and action descriptions, and that also forms the basis of its inferences. We will look at this aspect now.

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 10.1.13.** Entailment: Which $\mathbf{B}$ follow from $\mathbf{A}$, written $\mathbf{A} \models \mathbf{B}$, meaning that, for all φ with $\varphi \models \mathbf{A}$, we have $\varphi \models \mathbf{B}$? 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 10.1.14.** Deduction: Which formulas $\mathbf{B}$ can be derived from $\mathbf{A}$ using a set $\mathcal{C}$ of inference rules (a calculus), written $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$?
- **Example 10.1.15.** If $\mathcal{C}$ contains $\dfrac{\mathbf{A} \quad \mathbf{A} \Rightarrow \mathbf{B}}{\mathbf{B}}$ then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$
- **Intuition:** Deduction $\widehat{=}$ process in an actual computer trying to reason about entailment. E.g. a mechanical process attempting to determine Wumpus position.
- **Critical Insight:** Entailment is purely semantical and gives a mathematical foundation of reasoning in PL^0, while Deduction is purely syntactic and can be implemented well. (but this only helps if they are related)
- **Definition 10.1.16.** Soundness: whenever $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$, we also have $\mathbf{A} \models \mathbf{B}$.
- **Definition 10.1.17.** Completeness: whenever $\mathbf{A} \models \mathbf{B}$, we also have $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$.

General Problem Solving using Logic

- **Idea:** Any problem that can be formulated as reasoning about logic. $\rightsquigarrow$ 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.

10.1.3 Propositional Logic: Agenda

Our Agenda for This Topic

▷ **This subsection**: Basic definitions and concepts; tableaux, resolution.

- ▹ Sets up the framework. Resolution is the quintessential reasoning procedure underlying most successful SAT solvers.

▷ **Next Section (??)**: The Davis Putnam procedure and clause learning; practical problem structure.

- ▹ State-of-the-art algorithms for reasoning about propositional logic, and an important observation about how they behave.

Our Agenda for This Chapter

▷ Propositional logic: What's the syntax and semantics? How can we capture deduction?

- ▹ We study this logic formally.

▷ **Tableaux, Resolution**: How can we make deduction mechanizable? What are its properties?

- ▹ Formally introduces the most basic machine-oriented reasoning algorithm.

▷ **Killing a Wumpus**: How can we use all this to figure out where the Wumpus is?

- ▹ Coming back to our introductory example.

10.2 Propositional Logic (Syntax/Semantics)

Video Nuggets covering this section can be found at https://fau.tv/clip/id/22457 and https://fau.tv/clip/id/22458.

We will now develop the formal theory behind the ideas previewed in the last section and use that as a prototype for the theory of the more expressive logical systems still to come in AI-1. As PL^0 is a very simple logical system, we could cut some corners in the exposition but we will stick closer to a generalizable theory.

Propositional Logic (Syntax)

▷ **Definition 10.2.1 (Syntax).** The formulae of propositional logic (write PL^0) are made up from

- ▷ propositional variables: $\mathcal{V}_0 := \{P, Q, R, P^1, P^2, \ldots\}$ (countably infinite)
- ▷ A propositional signature: constants/constructors called connectives: $\Sigma_0 := \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots\}$

We define the set $wff_0(\mathcal{V}_0)$ of well-formed propositional formula (wffs) as

- ▷ propositional variables,
- ▷ the logical constants T and F,
- ▷ negations $\neg\mathbf{A}$,
- ▷ conjunctions $\mathbf{A} \wedge \mathbf{B}$($\mathbf{A}$ and $\mathbf{B}$ are called conjuncts),
- ▷ disjunctions $\mathbf{A} \vee \mathbf{B}$ ($\mathbf{A}$ and $\mathbf{B}$ are called disjuncts),
- ▷ implications $\mathbf{A} \Rightarrow \mathbf{B}$, and
- ▷ equivalences (or biimplication). $\mathbf{A} \Leftrightarrow \mathbf{B}$,

where $\mathbf{A}, \mathbf{B} \in wff_0(\mathcal{V}_0)$ themselves.

▷ **Example 10.2.2.** $P \wedge Q, P \vee Q, \neg P \vee Q \Leftrightarrow P \Rightarrow Q \in wff_0(\mathcal{V}_0)$

▷ **Definition 10.2.3.** Propositional formulae without connectives are called atomic (or an atom) and complex otherwise.

We can also express the formal language introduced by ?? as a context-free grammar.

Propositional Logic Grammar Overview

▷ **Grammar for Propositional Logic:**

propositional variables	X	::=	$\mathcal{V}_0 = \{P, Q, R, \ldots, \ldots\}$	variables
propositional formulae	$\mathbf{A}$	::=	X	variable
		|	$T \| F$	truth values
		|	$\neg\mathbf{A}$	negation
		|	$\mathbf{A}_1 \wedge \mathbf{A}_2$	conjunction
		|	$\mathbf{A}_1 \vee \mathbf{A}_2$	disjunction
		|	$\mathbf{A}_1 \Rightarrow \mathbf{A}_2$	implication
		|	$\mathbf{A}_1 \Leftrightarrow \mathbf{A}_2$	equivalence

Propositional logic is a very old and widely used logical system. So it should not be surprising that there are other notations for the connectives than the ones we are using in AI-1. We list the most important ones here for completeness.

Alternative Notations for Connectives

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

These notations will not be used in AI-1, but sometimes appear in the literature.
The semantics of PL^0 is defined relative to a model, which consists of a universe of discourse and an interpretation function that we specify now.

Semantics of PL^0 (Models)

- **Warning:** For the official semantics of PL^0 we will separate the tasks of giving meaning to connectives and propositional variables to different mappings.
- This will generalize better to other logical systems. (and thus applications)
- **Definition 10.2.4.** 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 constant a logical constant, iff its value is fixed by the interpretation.
- Treat the other connectives as abbreviations, e.g. $\mathbf{A}\vee\mathbf{B}\widehat{=}\neg(\neg\mathbf{A}\wedge\neg\mathbf{B})$ and $\mathbf{A}\Rightarrow\mathbf{B}\widehat{=}\neg\mathbf{A}\vee\mathbf{B}$, and $T\widehat{=}P\vee\neg P$ (only need to treat $\neg,\wedge$ directly)
- **Note:** PL^0 is a single-model logical system with canonical model $\langle\mathcal{D}_o,\mathcal{I}\rangle$.

We have a problem in the exposition of the theory here: As PL^0 semantics only has a single, canonical model, we could simplify the exposition by just not mentioning the universe and interpretation function. But we choose to expose both of them in the construction, since other versions of propositional logic – in particular the system $\mathrm{PL}^{\mathrm{nq}}$ below – that have a choice of models as they use a different distribution of the representation among constants and variables.

Semantics of PL^0 (Evaluation)

- **Problem:** The interpretation function $\mathcal{I}$ only assigns meaning to connectives.
- **Definition 10.2.5.** A variable assignment $\varphi\colon \mathcal{V}_0 \to \mathcal{D}_o$ assigns values to propositional variables.
- **Definition 10.2.6.** 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)
  - $\mathcal{I}_\varphi(\neg\mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathbf{A}))$.
  - $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathbf{A}), \mathcal{I}_\varphi(\mathbf{B}))$.
- **Note:** $\mathcal{I}_\varphi(\mathbf{A} \vee \mathbf{B}) = \mathcal{I}_\varphi(\neg(\neg\mathbf{A} \wedge \neg\mathbf{B}))$ is only determined by $\mathcal{I}_\varphi(\mathbf{A})$ and $\mathcal{I}_\varphi(\mathbf{B})$, so we think of the defined connectives as logical constants as well.
- **Alternative Notation:** Write $[\![\mathbf{A}]\!]_\varphi$ for $\mathcal{I}_\varphi(\mathbf{A})$. (and $[\![\mathbf{A}]\!]$, if $\mathbf{A}$ is ground)
- **Definition 10.2.7.** Two formulae $\mathbf{A}$ and $\mathbf{B}$ are called equivalent, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}_\varphi(\mathbf{B})$ for all variable assignments φ.

In particular in a interpretation-less exposition of propositional logic would have elided the homomorphic construction of the value function and could have simplified the recursive cases in **??** to $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T} = \mathcal{I}_\varphi(\mathbf{B})$.

But the homomorphic construction via $\mathcal{I}(\wedge)$ is standard to definitions in other logical systems and thus generalizes better.

Computing Semantics

- **Example 10.2.8.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

- What a mess!

Now we will also review some propositional identities that will be useful later on. Some of them we have already seen, and some are new. All of them can be proven by simple truth table arguments.

Propositional Identities

▷ **Definition 10.2.9.** We have the following identities in propositional logic:

Name	for $\wedge$	for $\vee$
Idempotence	$\varphi \wedge \varphi = \varphi$	$\varphi \vee \varphi = \varphi$
Identity	$\varphi \wedge T = \varphi$	$\varphi \vee F = \varphi$
Absorption 1	$\varphi \wedge F = F$	$\varphi \vee T = T$
Commutativity	$\varphi \wedge \psi = \psi \wedge \varphi$	$\varphi \vee \psi = \psi \vee \varphi$
Associativity	$\varphi \wedge (\psi \wedge \theta) = (\varphi \wedge \psi) \wedge \theta$	$\varphi \vee (\psi \vee \theta) = (\varphi \vee \psi) \vee \theta$
Distributivity	$\varphi \wedge (\psi \vee \theta) = \varphi \wedge \psi \vee \varphi \wedge \theta$	$\varphi \vee \psi \wedge \theta = (\varphi \vee \psi) \wedge (\varphi \vee \theta)$
Absorption 2	$\varphi \wedge (\varphi \vee \theta) = \varphi$	$\varphi \vee \varphi \wedge \theta = \varphi$
De Morgan rule	$\neg(\varphi \wedge \psi) = \neg\varphi \vee \neg\psi$	$\neg(\varphi \vee \psi) = \neg\varphi \wedge \neg\psi$
double negation	$\neg\neg\varphi = \varphi$	
Definitions	$\varphi \Rightarrow \psi = \neg\varphi \vee \psi$	$\varphi \Leftrightarrow \psi = (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$

▷ **Idea:** How about using these as inference component (simplification) to simplify calculations like the one in ??. (see below)

We will now use the distribution of values of a propositional formula under all variable assignments to characterize them semantically. The intuition here is that we want to understand theorems, examples, counterexamples, and inconsistencies in mathematics and everyday reasoning[1].

The idea is to use the formal language of propositional formulae as a model for mathematical language. Of course, we cannot express all of mathematics as propositional formulae, but we can at least study the interplay of mathematical statements (which can be true or false) with the copula "and", "or" and "not".

Semantic Properties of Propositional Formulae

▷ **Definition 10.2.10.** Let $\mathcal{M} := \langle\mathcal{U}, \mathcal{I}\rangle$ be our model, then we call $\mathbf{A}$

- ▷ true under φ (φ satisfies $\mathbf{A}$) in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, (write $\mathcal{M} \models^\varphi \mathbf{A}$)
- ▷ false under φ (φ falsifies $\mathbf{A}$) in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{F}$, (write $\mathcal{M} \not\models^\varphi \mathbf{A}$)
- ▷ satisfiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$ for some assignment φ,
- ▷ valid in $\mathcal{M}$, iff $\mathcal{M} \models^\varphi \mathbf{A}$ for all variable assignments φ,
- ▷ falsifiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{F}$ for some assignments φ, and
- ▷ unsatisfiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{F}$ for all assignments φ.

▷ **Example 10.2.11.** $x \vee x$ is satisfiable and falsifiable.

▷ **Example 10.2.12.** $x \vee \neg x$ is valid and $x \wedge \neg x$ is unsatisfiable.

▷ **Note:** As PL^0 is a single-model logical system, we can elide the reference to the model and regain the notation $\varphi \models \mathbf{A}$ from the preview for $\mathcal{M} \models^\varphi \mathbf{A}$.

▷ **Definition 10.2.13 (Entailment).** (aka. logical consequence)
We say that $\mathbf{A}$ entails $\mathbf{B}$ (write $\mathbf{A} \models \mathbf{B}$), iff $\mathcal{I}_\varphi(\mathbf{B}) = \mathsf{T}$ for all φ with $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$ (i.e. all assignments that make $\mathbf{A}$ true also make $\mathbf{B}$ true)

[1]Here (and elsewhere) we will use mathematics (and the language of mathematics) as a test tube for understanding reasoning, since mathematics has a long history of studying its own reasoning processes and assumptions.

Let us now see how these semantic properties model mathematical practice.

In mathematics we are interested in assertions that are true in all circumstances. In our model of mathematics, we use variable assignments to stand for "circumstances". So we are interested in propositional formulae which are true under all variable assignments; we call them valid. We often give examples (or show situations) which make a conjectured formula false; we call such examples counterexamples, and such assertions falsifiable. We also often give examples for certain formulae to show that they can indeed be made true (which is not the same as being valid yet); such assertions we call satisfiable. Finally, if a formula cannot be made true in any circumstances we call it unsatisfiable; such assertions naturally arise in mathematical practice in the form of refutation proofs, where we show that an assertion (usually the negation of the theorem we want to prove) leads to an obviously unsatisfiable conclusion, showing that the negation of the theorem is unsatisfiable, and thus the theorem valid.

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

Let us finally test our intuitions about propositional logic with a "real-world example": a logic puzzle, as you could find it in a Sunday edition of the local newspaper.

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.
   (a) Persons with red or green hair do not study AI.
   (b) Neither the Physics nor the Chinese students have black hair.
   (c) 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 $ai(S) \vee ai(N) \vee ai(J)$
   (a) $ai(x) \Rightarrow \neg red(x) \wedge \neg gre(x)$.
   (b) $phy(x) \Rightarrow \neg bla(x)$ and $chi(x) \Rightarrow \neg bla(x)$.
   (c) $phy(S) \wedge chi(J) \vee phy(J) \wedge chi(S)$.

Now, we obtain new knowledge via entailment steps:

3. 1. together with 2.2a entails that $ai(x) \Rightarrow bla(x)$ for every $x \in \{S, N, J\}$,
4. thus $\neg bla(S) \wedge \neg bla(J)$ by 2.2c and 2.2b and
5. so $\neg ai(S) \wedge \neg ai(J)$ by 3. and 4.
6. With 2. the latter entails $ai(N)$.

The example shows that puzzles like that are a bit difficult to solve without writing things down. But if we formalize the situation in PL^0, then we can solve the puzzle quite handily with inference. Note that we have been a bit generous with the names of propositional variables; e.g. $bla(x)$, where $x \in \{S, N, J\}$, to keep the representation small enough to fit on the slide. This does not hinder the method in any way.

10.3 Inference in Propositional Logics

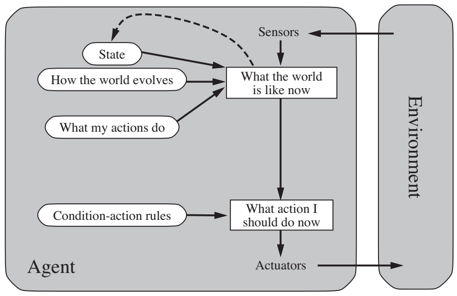
We have now defined syntax (the language agents can use to represent knowledge) and its semantics (how expressions of this language relate to agent's environment). Theoretically, an agent could use the entailment relation to derive new knowledge from percepts and the existing state representation – in the MAKE−PERCEPT−SENTENCE and MAKE−ACTION−SENTENCE subroutines below. But as we have seen in above, this is very tedious. A much better way would be to have a set of rules that directly act on the state representations.
Let us now look into what kind of agent we would need to be successful in the Wumpus world: it seems reasonable that we should build on a model-based agent and specialize it to structured state representations and inference.

Agents that Think Rationally

▷ **Problem:** But how can we build an agent that can do the necessary inferences?

▷ **Idea:** Think Before You Act!
"Thinking" = Inference about knowledge represented using logic.

▷ **Definition 10.3.1.** 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. Agent schema:



The formal language of the logical system acts as a world description language. Agent function:

function KB−AGENT ($percept$) **returns** an action
  persistent: KB, a knowledge base
    t, a counter, initially 0, indicating time
  TELL(KB, MAKE−PERCEPT−SENTENCE($percept$,t))
  $action$:= ASK(KB, MAKE−ACTION−QUERY(t))
  TELL(KB, MAKE−ACTION−SENTENCE($action$,t))
  t := t+1
  return $action$

Its agent function maintains a knowledge base about the environment, which is updated with percept descriptions (formalizations of the percepts) and action descriptions. The next action is the result of a suitable inference-based query to the knowledge base.

A Simple Formal System: Prop. Logic with Hilbert-Calculus

- **Formulae:** Built from propositional variables: $P, Q, R...$ and implication: $\Rightarrow$
- **Semantics:** $\mathcal{I}_\varphi(P) = \varphi(P)$ and $\mathcal{I}_\varphi(\mathbf{A} \Rightarrow \mathbf{B}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{F}$ or $\mathcal{I}_\varphi(\mathbf{B}) = \mathsf{T}$.
- **Definition 10.3.2.** The Hilbert calculus $\mathcal{H}^0$ consists of the inference rules:

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

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

- **Example 10.3.3.** A $\mathcal{H}^0$ theorem $\mathbf{C} \Rightarrow \mathbf{C}$ and its proof

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

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

This is indeed a very simple formal system, but it has all the required parts:

- A formal language: expressions built up from variables and implications.
- A semantics: given by the obvious interpretation function
- A calculus: given by the two axioms and the two inference rules.

The calculus gives us a set of rules with which we can derive new formulae from old ones. The axioms are very simple rules, they allow us to derive these two formulae in any situation. The proper inference rules are slightly more complicated: we read the formulae above the horizontal line as assumptions and the (single) formula below as the conclusion. An inference rule allows us to derive the conclusion, if we have already derived the assumptions.
Now, we can use these inference rules to perform a proof – a sequence of formulae that can be derived from each other. The representation of the proof in the slide is slightly compactified to fit onto the slide: We will make it more explicit here. We first start out by deriving the formula

$$(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R \tag{10.1}$$

which we can always do, since we have an axiom for this formula, then we apply the rule Subst, where $\mathbf{A}$ is this result, $\mathbf{B}$ is $\mathbf{C}$, and X is the variable P to obtain

$$(\mathbf{C} \Rightarrow Q \Rightarrow R) \Rightarrow (\mathbf{C} \Rightarrow Q) \Rightarrow \mathbf{C} \Rightarrow R \tag{10.2}$$

Next we apply the rule Subst to this where $\mathbf{B}$ is $\mathbf{C} \Rightarrow \mathbf{C}$ and X is the variable Q this time to obtain

$$(\mathbf{C} \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow R) \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C} \Rightarrow R \tag{10.3}$$

And again, we apply the rule Subst this time, $\mathbf{B}$ is $\mathbf{C}$ and X is the variable R yielding the first formula in our proof on the slide. To conserve space, we have combined these three steps into one in the slide. The next steps are done in exactly the same way.
In general, formulae can be used to represent facts about the world as propositions; they have a semantics that is a mapping of formulae into the real world (propositions are mapped to truth values.) We have seen two relations on formulae: the entailment relation and the derivation relation. The first one is defined purely in terms of the semantics, the second one is given by a calculus, i.e. purely syntactically. Is there any relation between these relations?

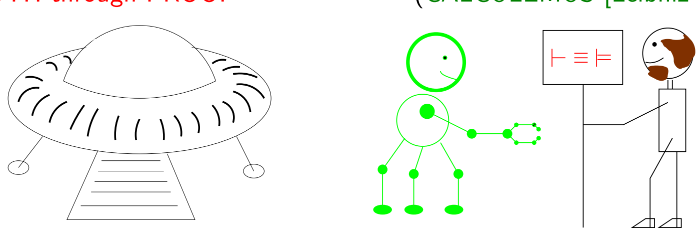
Soundness and Completeness

▷ **Definition 10.3.4.** 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 $\mathcal{H} \models \mathbf{A}$, whenever $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$, and
- ▷ complete, iff $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$, whenever $\mathcal{H} \models \mathbf{A}$.

▷ **Goal:** Find calculi C, such that $\vdash_C \mathbf{A}$ iff $\models \mathbf{A}$ (provability and validity coincide)

- ▷ To TRUTH through PROOF (CALCULEMUS [Leibniz ~1680])



▷

Michael Kohlhase: Artificial Intelligence 1 336 2025-02-06

Ideally, both relations would be the same, then the calculus would allow us to infer all facts that can be represented in the given formal language and that are true in the real world, and only those. In other words, our representation and inference is faithful to the world.

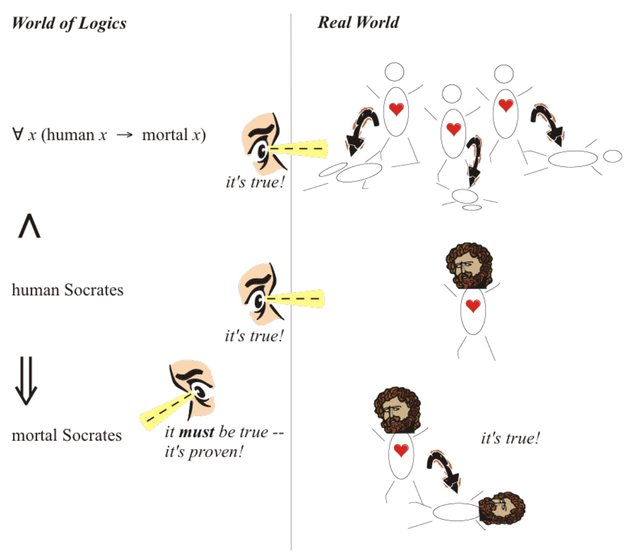
A consequence of this is that we can rely on purely syntactical means to make predictions about the world. Computers rely on formal representations of the world; if we want to solve a problem on our computer, we first represent it in the computer (as data structures, which can be seen as a formal language) and do syntactic manipulations on these structures (a form of calculus). Now, if the provability relation induced by the calculus and the validity relation coincide (this will be quite difficult to establish in general), then the solutions of the program will be correct, and we will find all possible ones. Of course, the logics we have studied so far are very simple, and not able to express interesting facts about the world, but we will study them as a simple example of the fundamental problem of computer science: How do the formal representations correlate with the real world.

Within the world of logics, one can derive new propositions (the *conclusions*, here: *Socrates is mortal*) from given ones (the *premises*, here: *Every human is mortal* and *Sokrates is human*). Such derivations are *proofs*.

In particular, logics can describe the internal structure of real-life facts; e.g. individual things, actions, properties. A famous example, which is in fact as old as it appears, is illustrated in the slide below.

The Miracle of Logic

▷ Purely formal derivations are true in the real world!



Michael Kohlhase: Artificial Intelligence 1 337 2025-02-06

If a formal system is correct, the conclusions one can prove are true (= hold in the real world) whenever the premises are true. This is a miraculous fact (think about it!)

10.4 Propositional Natural Deduction Calculus

Video Nuggets covering this section can be found at `https://fau.tv/clip/id/22520` and `https://fau.tv/clip/id/22525`.

We will now introduce the "natural deduction" calculus for propositional logic. The calculus was created to model the natural mode of reasoning e.g. in everyday mathematical practice. In particular, it was intended as a counter-approach to the well-known Hilbert style calculi, which were mainly used as theoretical devices for studying reasoning in principle, not for modeling particular reasoning styles. We will introduce natural deduction in two styles/notations, both were invented by Gerhard Gentzen in the 1930's and are very much related. The Natural Deduction style (ND) uses local hypotheses in proofs for hypothetical reasoning, while the "sequent style" is a rationalized version and extension of the ND calculus that makes certain meta-proofs simpler to push through by making the context of local hypotheses explicit in the notation. The sequent notation also constitutes a more adequate data struture for implementations, and user interfaces.

Rather than using a minimal set of inference rules, we introduce a natural deduction calculus that provides two/three inference rules for every logical constant, one "introduction rule" (an inference rule that derives a formula with that logical constant at the head) and one "elimination rule" (an inference rule that acts on a formula with this head and derives a set of subformulae).

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

▷ **Idea:** $\mathcal{ND}_0$ tries to mimic human argumentation for theorem proving.

▷ **Definition 10.4.1.** The propositional natural deduction calculus $\mathcal{ND}_0$ has inference rules for the introduction and elimination of connectives:

Introduction	Elimination	Axiom
$\frac{\mathbf{A} \quad \mathbf{B}}{\mathbf{A} \wedge \mathbf{B}} \mathcal{ND}_0{\wedge}I$	$\frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{A}} \mathcal{ND}_0{\wedge}E_l$ $\quad \frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{B}} \mathcal{ND}_0{\wedge}E_r$	$\frac{}{\mathbf{A} \vee \neg\mathbf{A}} \mathcal{ND}_0\mathrm{TND}$
$\frac{\begin{array}{c}[\mathbf{A}]^1 \\ = \\ \mathbf{B}\end{array}}{\mathbf{A} \Rightarrow \mathbf{B}} \mathcal{ND}_0 \Rightarrow I^1$	$\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}} \mathcal{ND}_0 \Rightarrow E$	

$\mathcal{ND}_0 \Rightarrow I^a$ proves $\mathbf{A} \Rightarrow \mathbf{B}$ by exhibiting a $\mathcal{ND}_0$ derivation $\mathcal{D}$ (depicted by the double horizontal lines) of $\mathbf{B}$ from the local hypothesis $\mathbf{A}$; $\mathcal{ND}_0 \Rightarrow I^a$ then discharges (get rid of $\mathbf{A}$, which can only be used in $\mathcal{D}$) the local hypothesis and concludes $\mathbf{A} \Rightarrow \mathbf{B}$. This mode of reasoning is called hypothetical reasoning.

▷ **Definition 10.4.2.** Given a set $\mathcal{H} \subseteq wf\!f_0(\mathcal{V}_0)$ of assumptions and a conclusion $\mathbf{C}$, we write $\mathcal{H} \vdash_{\mathcal{ND}_0} \mathbf{C}$, 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)

The most characteristic rule in the natural deduction calculus is the $\mathcal{ND}_0 \Rightarrow I^a$ rule and the hypothetical reasoning it introduce. $\mathcal{ND}_0 \Rightarrow I^a$ corresponds to the mathematical way of proving an implication $\mathbf{A} \Rightarrow \mathbf{B}$: We assume that $\mathbf{A}$ is true and show $\mathbf{B}$ from this local hypothesis. When we can do this we discharge the assumption and conclude $\mathbf{A} \Rightarrow \mathbf{B}$.

Note that the local hypothesis is discharged by the rule $\mathcal{ND}_0 \Rightarrow I^a$, i.e. it cannot be used in any other part of the proof. As the $\mathcal{ND}_0 \Rightarrow I^a$ rules may be nested, we decorate both the rule and the corresponding local hypothesis with a marker (here the number 1).

Let us now consider an example of hypothetical reasoning in action.

Natural Deduction: Examples

▷ **Example 10.4.3 (Inference with Local Hypotheses).**

$$\dfrac{\dfrac{\dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{B}}\mathcal{ND}_0{\wedge}E_r \quad \dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{A}}\mathcal{ND}_0{\wedge}E_l}{\mathbf{B}\wedge\mathbf{A}}\mathcal{ND}_0{\wedge}I}{\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\mathcal{ND}_0{\Rightarrow}I^1 \qquad \dfrac{\dfrac{\begin{array}{c}[A]^1\\ [B]^2\\ A\end{array}}{B\Rightarrow A}\mathcal{ND}_0{\Rightarrow}I^2}{A\Rightarrow B\Rightarrow A}\mathcal{ND}_0{\Rightarrow}I^1$$

Here we see hypothetical reasoning with local local hypotheses at work. In the left example, we assume the formula $\mathbf{A}\wedge\mathbf{B}$ and can use it in the proof until it is discharged by the rule $\mathcal{ND}_0{\wedge}E_l$ on the bottom – therefore we decorate the hypothesis and the rule by corresponding numbers (here the label "1"). Note the local assumption $\mathbf{A}\wedge\mathbf{B}$ is *local to the proof fragment* delineated by the corresponding (local) hypothesis and the discharging rule, i.e. even if this derivation is only a fragment of a larger proof, then we cannot use its (local) hypothesis anywhere else.

Note also that we can use as many copies of the local hypothesis as we need; they are all discharged at the same time.

In the right example we see that local hypotheses can be nested as long as they are kept local. In particular, we may not use the hypothesis $\mathbf{B}$ after the $\mathcal{ND}_0{\Rightarrow}I^2$, e.g. to continue with a $\mathcal{ND}_0{\Rightarrow}E$.

One of the nice things about the natural deduction calculus is that the deduction theorem is almost trivial to prove. In a sense, the triviality of the deduction theorem is the central idea of the calculus and the feature that makes it so natural.

A Deduction Theorem for $\mathcal{ND}_0$

▷ **Theorem 10.4.4.** $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}_0}\mathbf{B}$, *iff* $\mathcal{H}\vdash_{\mathcal{ND}_0}\mathbf{A}\Rightarrow\mathbf{B}$.

▷ *Proof:* We show the two directions separately

1. If $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}_0}\mathbf{B}$, then $\mathcal{H}\vdash_{\mathcal{ND}_0}\mathbf{A}\Rightarrow\mathbf{B}$ by $\mathcal{ND}_0{\Rightarrow}I$, and
2. If $\mathcal{H}\vdash_{\mathcal{ND}_0}\mathbf{A}\Rightarrow\mathbf{B}$, then $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}_0}\mathbf{A}\Rightarrow\mathbf{B}$ by weakening and $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}_0}\mathbf{B}$ by $\mathcal{ND}_0{\Rightarrow}E$.

Another characteristic of the natural deduction calculus is that it has inference rules (introduction and elimination rules) for all connectives. So we extend the set of rules from **??** for disjunction, negation and falsity.

More Rules for Natural Deduction

▷ **Note:** $\mathcal{ND}_0$ does not try to be minimal, but comfortable to work in!

▷ **Definition 10.4.5.** $\mathcal{ND}_0$ has the following additional inference rules for the remaining connectives.

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

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

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

▷ **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 10.4.6.** A judgment is a meta-statement about the provability of propositions.

▷ **Definition 10.4.7.** A sequent is a judgment of the form $\mathcal{H}\vdash\mathbf{A}$ about the provability of the formula $\mathbf{A}$ from the set $\mathcal{H}$ of hypotheses. We write $\vdash\mathbf{A}$ for $\emptyset\vdash\mathbf{A}$.

▷ **Idea:** Reformulate $\mathcal{ND}_0$ inference rules so that they act on sequents.

▷ **Example 10.4.8.** We give the sequent style version of ??:

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

▷ **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 10.4.9.** The following inference rules make up the propositional sequent style natural deduction calculus $\mathcal{ND}^0_\vdash$:

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

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

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

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

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

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

FAU Michael Kohlhase: Artificial Intelligence 1 343 2025-02-06

Linearized Notation for (Sequent-Style) ND Proofs

▷ **Definition 10.4.10.** Linearized notation for sequent-style ND proofs

1.	$\mathcal{H}_1$	$\vdash$	$\mathbf{A}_1$	$(\mathcal{J}_1)$
2.	$\mathcal{H}_2$	$\vdash$	$\mathbf{A}_2$	$(\mathcal{J}_2)$
3.	$\mathcal{H}_3$	$\vdash$	$\mathbf{A}_3$	$(\mathcal{J}_3 1,2)$

corresponds to

$$\frac{\mathcal{H}_1\vdash\mathbf{A}_1\quad\mathcal{H}_2\vdash\mathbf{A}_2}{\mathcal{H}_3\vdash\mathbf{A}_3}\,\mathcal{R}$$

▷ **Example 10.4.11.** We show a linearized version of the $\mathcal{ND}_0$ examples ??

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

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

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

FAU Michael Kohlhase: Artificial Intelligence 1 344 2025-02-06

Each row in the table represents one inference step in the proof. It consists of line number (for referencing), a formula for the statement, a justification via a ND inference rule (and the rows this one is derived from), and finally a sequence of row numbers of proof steps that are local hypotheses in effect for the current row.

10.5 Predicate Logic Without Quantifiers

In the hair-color example we have seen that we are able to model complex situations in PL^0. The trick of using variables with fancy names like $bla(N)$ is a bit dubious, and we can already imagine that it will be difficult to support programmatically unless we make names like $bla(N)$ into first-class citizens i.e. expressions of the logic language themselves.

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)

Michael Kohlhase: Artificial Intelligence 1 345 2025-02-06

Individuals and their Properties/Relationships

- **Observation:** We want to talk about individuals like Stefan, Nicole, and Jochen and their properties, e.g. being blond, or studying AI and relationships, e.g. that *Stefan loves Nicole*.
- **Idea:** Re-use PL^0, but replace propositional variables with something more expressive! (instead of fancy variable name trick)
- **Definition 10.5.1.** A first-order signature $\langle \Sigma^f, \Sigma^p \rangle$ consists of
  - $\Sigma^f := \bigcup_{k\in\mathbb{N}} \Sigma^f_k$ of function constants, where members of Σ^f_k denote k-ary functions on individuals,
  - $\Sigma^p := \bigcup_{k\in\mathbb{N}} {\Sigma^p}_k$ of predicate constants, where members of ${\Sigma^p}_k$ denote k-ary relations among individuals,

  where Σ^f_k and ${\Sigma^p}_k$ are pairwise disjoint, countable sets of symbols for each $k \in \mathbb{N}$.

  A 0-ary function constant refers to a single individual, therefore we call it a individual constant.

A Grammar for PL^{nq}

▷ **Definition 10.5.2.** The formulae of PL^{nq} are given by the following grammar

function constants	f^k	$\in$	Σ^f_k	
predicate constants	p^k	$\in$	$\Sigma^p{}_k$	
terms	t	$::=$	f^0	individualconstant
		$\mid$	$f^k(t_1, \ldots, t_k)$	application
formulae	$\mathbf{A}$	$::=$	$p^k(t_1, \ldots, t_k)$	atomic
		$\mid$	$\neg\mathbf{A}$	negation
		$\mid$	$\mathbf{A}_1 \wedge \mathbf{A}_2$	conjunction

PL^{nq} Semantics

▷ **Definition 10.5.3.** Domains $\mathcal{D}_0 = \{\mathsf{T}, \mathsf{F}\}$ of truth values and $\mathcal{D}_\iota \neq \emptyset$ of individuals.

▷ **Definition 10.5.4.** 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)
- ▷ $\mathcal{I}\colon \Sigma^f_0 \to \mathcal{D}_\iota$ (interpret individual constants as individuals)
- ▷ $\mathcal{I}\colon \Sigma^f_k \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (interpret function constants as functions)
- ▷ $\mathcal{I}\colon \Sigma^p{}_k \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicate constants as relations)

▷ **Definition 10.5.5.** The value function $\mathcal{I}$ assigns values to formulae: (recursively)

- ▷ $\mathcal{I}(f(\mathbf{A}^1, \ldots, \mathbf{A}^k)) := \mathcal{I}(f)(\mathcal{I}(\mathbf{A}^1), \ldots, \mathcal{I}(\mathbf{A}^k))$
- ▷ $\mathcal{I}(p(\mathbf{A}^1, \ldots, \mathbf{A}^k)) := \mathsf{T}$, iff $\langle \mathcal{I}(\mathbf{A}^1), \ldots, \mathcal{I}(\mathbf{A}^k)\rangle \in \mathcal{I}(p)$
- ▷ $\mathcal{I}(\neg\mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}(\mathbf{A}))$ and $\mathcal{I}(\mathbf{A} \wedge \mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}(\mathbf{A}), \mathcal{I}(\mathbf{G}))$ (just as in PL^0)

▷ **Definition 10.5.6.** Model: $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I}\rangle$ varies in $\mathcal{D}_\iota$ and $\mathcal{I}$.

▷ **Theorem 10.5.7.** *PL^{nq} is isomorphic to PL^0* (*interpret atoms as prop. variables*)

All of the definitions above are quite abstract, we now look at them again using a very concrete – if somewhat contrived – example: The relevant parts are a universe $\mathcal{D}$ with four elements, and an interpretation that maps the signature into individuals, functions, and predicates over $\mathcal{D}$, which are given as concrete sets.

A Model for PL^{nq}

▷ **Example 10.5.8.** 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,

- ▷ $P \mapsto \{♣, ♠\}$ and $Q \mapsto \{♠, ♢\}$, for unary predicate constants.
- ▷ $R \mapsto \{\langle ♡, ♢\rangle, \langle ♢, ♡\rangle\}$, and $S \mapsto \{\langle ♢, ♠\rangle, \langle ♠, ♣\rangle\}$ for binary predicate constants.

▷ **Example 10.5.9 (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
- ▷ $\langle ♣, ♠\rangle \in \{\langle ♡, ♢\rangle, \langle ♢, ♡\rangle\}$ and $♡ \in \{♣, ♠\}$

So, $\mathcal{I}(R(a,b) \wedge P(c)) = \mathsf{F}$.

FAU Michael Kohlhase: Artificial Intelligence 1 349 2025-02-06

The example above also shows how we can compute of meaning by in a concrete model: we just follow the evaluation rules to the letter.

We now come to the central technical result about $\mathrm{PL}^{\mathrm{nq}}$: it is essentially the same as propositional logic (PL^0). We say that the two logic are isomorphic. Technically, this means that the formulae of $\mathrm{PL}^{\mathrm{nq}}$ can be translated to PL^0 and there is a corresponding model translation from the models of PL^0 to those of $\mathrm{PL}^{\mathrm{nq}}$ such that the respective notions of evaluation are assignped to each other.

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

▷ **Observation:** For every choice of Σ of signature, the set $\mathcal{A}_\Sigma$ of atomic $\mathrm{PL}^{\mathrm{nq}}$ formulae is countable, so there is a $\mathcal{V}_\Sigma \subseteq \mathcal{V}_0$ and a bijection $\theta_\Sigma \colon \mathcal{A}_\Sigma \to \mathcal{V}_\Sigma$.

θ_Σ can be extended to formulae as $\mathrm{PL}^{\mathrm{nq}}$ and PL^0 share connectives.

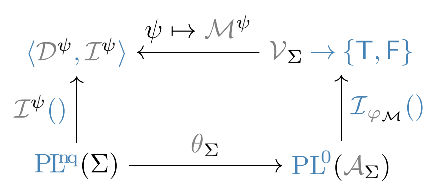
▷ **Lemma 10.5.10.** *For every model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I}\rangle$, there is a variable assignment $\varphi_\mathcal{M}$, such that $\mathcal{I}_{\varphi_\mathcal{M}}(\mathbf{A}) = \mathcal{I}(\mathbf{A})$.*

▷ *Proof sketch:* We just define $\varphi_\mathcal{M}(X) := \mathcal{I}(\theta_\Sigma^{-1}(X))$

▷ **Lemma 10.5.11.** *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 $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}^\psi(\mathbf{A})$.*

▷ *Proof sketch:* see next slide

▷ **Corollary 10.5.12.** *$\mathrm{PL}^{\mathrm{nq}}$ is isomorphic to PL^0, i.e. the following diagram commutes:*



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

FAU Michael Kohlhase: Artificial Intelligence 1 350 2025-02-06

The practical upshot of the commutative diagram from ?? is that if we have a way of computing evaluation (or entailment for that matter) in PL^0, then we can "borrow" it for $\mathrm{PL}^{\mathrm{nq}}$ by composing it with the language and model translations. In other words, we can reuse calculi and automated

theorem provers from PL^0 for $\mathrm{PL}^{\mathrm{nq}}$.
But we still have to provide the proof for ??, which we do now.

Valuation and Satisfiability

▷ **Lemma 10.5.13.** *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 $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}^\psi(\mathbf{A})$.*

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

10.6 Conclusion

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25027`.

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. $\mathbf{A}$ entails $\mathbf{B}$ if all interpretations that satisfy $\mathbf{A}$ also satisfy $\mathbf{B}$. Deduction is the process of deriving new entailed formulae.

▷ Propositional logic formulae 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 $\leadsto$ further expansion of the rules.

▷ **Can we design a more human-like logic?:** Yep

- ▷ Predicate logic: quantification of variables ranging over individuals. (cf. ?? and ??)
- ▷ ... and a whole zoo of logics much more powerful still.
- ▷ Note: In applications, propositional CNF encodings are generated by computer programs. This mitigates (but does not remove!) the inconveniences of propositional modeling.

Suggested Reading:

- *Chapter 7: Logical Agents*, Sections 7.1 – 7.5 [RN09].
  - Sections 7.1 and 7.2 roughly correspond to my "Introduction", Section 7.3 roughly corresponds to my "Logic (in AI)", Section 7.4 roughly corresponds to my "Propositional Logic", Section 7.5 roughly corresponds to my "Resolution" and "Killing a Wumpus".
  - Overall, the content is quite similar. I have tried to add some additional clarifying illustrations. RN gives many complementary explanations, nice as additional background reading.
  - I would note that RN's presentation of resolution seems a bit awkward, and Section 7.5 contains some additional material that is imho not interesting (alternate inference rules, forward and backward chaining). Horn clauses and unit resolution (also in Section 7.5), on the other hand, are quite relevant.

Chapter 11

Formal Systems: Syntax, Semantics, Entailment, and Derivation in General

We will now take a more abstract view and introduce the necessary prerequisites of abstract rule systems. We will also take the opportunity to discuss the quality criteria for calculi.

Recap: General Aspects of Propositional Logic

- **There are many ways to define Propositional Logic:**
  - We chose $\wedge$ and $\neg$ as primitive, and many others as defined.
  - We could have used $\vee$ and $\neg$ just as well.
  - We could even have used only one connective e.g. negated conjunction $\uparrow$ or disjunction $\downarrow$ and defined $\wedge$, $\vee$, and $\neg$ via $\uparrow$ and $\downarrow$ respectively.

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

$\downarrow$	$\top$	$\bot$
$\top$	F	F
$\bot$	F	T

$\neg a$	$a \uparrow a$	$a \downarrow a$
ab	$a \uparrow b \uparrow a \uparrow b$	$a \downarrow ab \downarrow b$
ab	$a \uparrow a \uparrow b \uparrow b$	$a \downarrow b \downarrow a \downarrow b$

- **Observation:** The set $wff_0(\mathcal{V}_0)$ 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 $\mathcal{M} \models \mathbf{A}$, and
  - entailment i.e whether $\mathbf{A} \models \mathbf{B}$.
- **Observation:** In particular, the inductive/compositional nature of $wff_0(\mathcal{V}_0)$ and $\mathcal{I}_\varphi \colon wff_0(\mathcal{V}_0) \to \mathcal{D}_0$ are secondary.
- **Idea:** Concentrate on language, models $(\mathcal{M}, \varphi)$, and satisfiability.

FAU Michael Kohlhase: Artificial Intelligence 1 354 2025-02-06

The notion of a logical system is at the basis of the field of logic. In its most abstract form, a logical system consists of a formal language, a class of models, and a satisfaction relation between models and expressions of the formal language. The satisfaction relation tells us when an expression is deemed true in this model.

Logical Systems

- **Definition 11.0.1.** A logical system (or simply a logic) is a triple $\mathcal{L} := \langle\mathcal{L}, \mathcal{K}, \models\rangle$, where the language $\mathcal{L}$ is a formal language, the model class $\mathcal{K}$ 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 11.0.2 (Propositional Logic).** $\langle wff(\Sigma_{PL^0}, \mathcal{V}_{PL^0}), \mathcal{K}_o, \models\rangle$ is a logical system, if we define $\mathcal{K}_o := \mathcal{V}_0 \rightharpoonup \mathcal{D}_0$ (the set of variable assignments) and $\varphi \models \mathbf{A}$ iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$.
- **Definition 11.0.3.** Let $\langle\mathcal{L}, \mathcal{K}, \models\rangle$ be a logical system, $\mathbf{M} \in \mathcal{K}$ a model and $\mathbf{A} \in \mathcal{L}$ a formula. Then we say that $\mathbf{A}$ is
  - satisfied by $\mathbf{M}$ iff $\mathbf{M} \models \mathbf{A}$.
  - satisfiable iff $\mathbf{A}$ is satisfied by some model.
  - unsatisfiable iff $\mathbf{A}$ is not satisfiable.
  - falsified by $\mathbf{M}$ iff $\mathbf{M} \not\models \mathbf{A}$.
  - valid or unfalsifiable (write $\models \mathbf{A}$) iff $\mathbf{A}$ is satisfied by every model.
  - invalid or falsifiable (write $\not\models \mathbf{A}$) iff $\mathbf{A}$ is not valid.

Let us now turn to the syntactical counterpart of the entailment relation: derivability in a calculus. Again, we take care to define the concepts at the general level of logical systems.
The intuition of a calculus is that it provides a set of syntactic rules that allow to reason by considering the form of propositions alone. Such rules are called inference rules, and they can be strung together to derivations — which can alternatively be viewed either as sequences of formulae where all formulae are justified by prior formulae or as trees of inference rule applications. But we can also define a calculus in the more general setting of logical systems as an arbitrary relation on formulae with some general properties. That allows us to abstract away from the homomorphic setup of logics and calculi and concentrate on the basics.

Derivation Relations and Inference Rules

- **Definition 11.0.4.** Let $\mathcal{L}$ be a formal language, then we call a relation $\vdash \subseteq \mathcal{P}(\mathcal{L}) \times \mathcal{L}$ a derivation relation for $\mathcal{L}$, if
  - $\mathcal{H} \vdash \mathbf{A}$, if $\mathbf{A} \in \mathcal{H}$ ($\vdash$ is proof reflexive),
  - $\mathcal{H} \vdash \mathbf{A}$ and $(\mathcal{H}' \cup \{\mathbf{A}\}) \vdash \mathbf{B}$ imply $(\mathcal{H} \cup \mathcal{H}') \vdash \mathbf{B}$ ($\vdash$ is proof transitive),
  - $\mathcal{H} \vdash \mathbf{A}$ and $\mathcal{H} \subseteq \mathcal{H}'$ imply $\mathcal{H}' \vdash \mathbf{A}$ ($\vdash$ is monotonic or admits weakening).
- **Definition 11.0.5.** Let $\mathcal{L}$ be a formal language, then an inference rule over $\mathcal{L}$ is a decidable $n+1$ ary relation on $\mathcal{L}$. Inference rules are traditionally written as

$$\frac{\mathbf{A}_1 \quad \ldots \quad \mathbf{A}_n}{\mathbf{C}}\,\mathcal{N}$$

  where $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and $\mathbf{C}$ are schemata for words in $\mathcal{L}$ and $\mathcal{N}$ is a name. The $\mathbf{A}_i$ are called assumptions of $\mathcal{N}$, and $\mathbf{C}$ is called its conclusion.

Any $n+1$-tuple

$$\frac{\mathbf{a}_1 \quad \ldots \quad \mathbf{a}_n}{\mathbf{c}}$$

in $\mathcal{N}$ is called an application of $\mathcal{N}$ and we say that we apply $\mathcal{N}$ to a set M of words with $\mathbf{a}_1, \ldots, \mathbf{a}_n \in M$ to obtain $\mathbf{c}$.

- ▷ **Definition 11.0.6.** An inference rule without assumptions is called an axiom.
- ▷ **Definition 11.0.7.** A calculus (or inference system) is a formal language $\mathcal{L}$ equipped with a set $\mathcal{C}$ of inference rules over $\mathcal{L}$.

With formula schemata we mean representations of sets of formulae, we use boldface uppercase letters as (meta)-variables for formulae, for instance the formula schema $\mathbf{A} \Rightarrow \mathbf{B}$ represents the set of formulae whose head is $\Rightarrow$.

Derivations

- ▷ **Definition 11.0.8.**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 $\mathbf{C} \in \mathcal{L}$ from a set $\mathcal{H} \subseteq \mathcal{L}$ of hypotheses (write $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{C}$) is a sequence $\mathbf{A}_1, \ldots, \mathbf{A}_m$ of $\mathcal{L}$-formulae, such that
  - ▷ $\mathbf{A}_m = \mathbf{C}$, (derivation culminates in $\mathbf{C}$)
  - ▷ for all $1 \leq i \leq m$, either $\mathbf{A}_i \in \mathcal{H}$, or (hypothesis)
  - ▷ there is an inference rule $\frac{\mathbf{A}_{l_1} \quad \ldots \quad \mathbf{A}_{l_k}}{\mathbf{A}_i}$ 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 $\mathbf{A}_{l_j}$ are the children of the node $\mathbf{A}_i$.
- ▷ **Example 11.0.9.**

  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.

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

Inference rules are relations on formulae represented by formula schemata (where boldface, uppercase letters are used as metavariables for formulae). For instance, in **??** the inference rule $\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}$ was applied in a situation, where the metavariables $\mathbf{A}$ and $\mathbf{B}$ were instantiated by the formulae P and $Q \Rightarrow P$.

As axioms do not have assumptions, they can be added to a derivation at any time. This is just what we did with the axioms in **??**.

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 $\langle \mathcal{L}, \mathcal{C}, \mathcal{K}, \models \rangle$ 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 11.0.10.** Let $\mathcal{C}$ be a calculus, then a $\mathcal{C}$-derivation $\emptyset \vdash_{\mathcal{C}} \mathbf{A}$ is called a proof of $\mathbf{A}$ and if one exists (write $\vdash_{\mathcal{C}} \mathbf{A}$) then $\mathbf{A}$ is called a $\mathcal{C}$-theorem.

Definition 11.0.11. The act of finding a proof for $\mathbf{A}$ is called proving $\mathbf{A}$.

▷ **Definition 11.0.12.** 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 11.0.13.** An inference rule

$$\frac{\mathbf{A}_1 \quad \ldots \quad \mathbf{A}_n}{\mathbf{C}}$$

is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $\mathbf{A}_1, \ldots, \mathbf{A}_n \vdash_{\mathcal{C}} \mathbf{C}$.

▷ **Observation 11.0.14.** *Derivable inference rules are admissible, but not the other way around.*

The notion of a formal system encapsulates the most general way we can conceptualize a logical system with a calculus, i.e. a system in which we can do "formal reasoning".

Chapter 12

Machine-Oriented Calculi for Propositional Logic

A Video Nugget covering this chapter can be found at `https://fau.tv/clip/id/22531`.

12.1 Test Calculi

Automated Deduction as an Agent Inference Procedure

- ▷ **Recall:** Our knowledge of the cave entails a definite Wumpus position!(slide 316)
- ▷ **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.

The following theorem is simple, but will be crucial later on.

Unsatisfiability Theorem

- ▷ **Theorem 12.1.1 (Unsatisfiability Theorem).** *$\mathcal{H} \models \mathbf{A}$ iff $\mathcal{H} \cup \{\neg\mathbf{A}\}$ is unsatisfiable.*
- ▷ *Proof:* We prove both directions separately
  1. "⇒": Say $\mathcal{H} \models \mathbf{A}$
     1.1. For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \models \mathbf{A}$ and thus $\varphi \not\models (\neg\mathbf{A})$.
  2. "⇐": Say $\mathcal{H} \cup \{\neg\mathbf{A}\}$ is unsatisfiable.
     2.1. For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \not\models (\neg\mathbf{A})$ and thus $\varphi \models \mathbf{A}$.
- ▷ **Observation 12.1.2.** *Entailment can be tested via satisfiability.*

Test Calculi: A Paradigm for Automating Inference

- **Definition 12.1.3.** Given a formal system $\langle\mathcal{L},\mathcal{C},\mathcal{K},\models\rangle$, the task of theorem proving consists in determining whether $\mathcal{H}\vdash_{\mathcal{C}}C$ for a conjecture $C\in\mathcal{L}$ and hypotheses $\mathcal{H}\subseteq\mathcal{L}$.
- **Definition 12.1.4.** Automated theorem proving (ATP) is the automation of theorem proving
- **Idea:** A set $\mathcal{H}$ of hypotheses and a conjecture $\mathbf{A}$ induce a search problem $\Pi_{\mathcal{C}}^{\mathcal{H}\models\mathbf{A}}:=\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, where the states $\mathcal{S}$ are sets of formulae, the actions $\mathcal{A}$ are the inference rules from $\mathcal{C}$, the initial state $\mathcal{I}=\mathcal{H}$, and the goal states are those with $\mathbf{A}\in\mathcal{S}$.
- **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 (**??**).
- **Definition 12.1.5.** For a given conjecture A and hypotheses $\mathcal{H}$ a test calculus $\mathcal{T}$ tries to derive a refutation $\mathcal{H},\overline{A}\vdash_{\mathcal{T}}\bot$ instead of $\mathcal{H}\vdash A$, where $\overline{A}$ is unsatisfiable iff A is valid and $\bot$, an "obviously" unsatisfiable formula.
- **Observation:** A test calculus $\mathcal{C}$ induces a search problem where the initial state is $\mathcal{H}\cup\{\neg\mathbf{A}\}$ and $S\in\mathcal{S}$ is a goal state iff $\bot\in S$.(proximity of $\bot$ easier for heuristics)
- Searching for $\bot$ admits simple heuristics, e.g. size reduction. ($\bot$ minimal)

12.1.1 Normal Forms

Before we can start, we will need to recap some nomenclature on formulae.

Recap: Atoms and Literals

- **Definition 12.1.6.** A formula is called atomic (or an atom) if it does not contain logical constants, else it is called complex.
- **Definition 12.1.7.** Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system and $\mathbf{A}\in\mathcal{L}$, then we call a pair $\mathbf{A}^\alpha$ of a formula and a truth value $\alpha\in\{\mathsf{T},\mathsf{F}\}$ a labeled formula. For a set Φ of formulae we use $\Phi^\alpha:=\{\mathbf{A}^\alpha\,|\,\mathbf{A}\in\Phi\}$.

  We call a labeled formula $\mathbf{A}^{\mathsf{T}}$ positive and $\mathbf{A}^{\mathsf{F}}$ negative.

  Definition 12.1.8. Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system and $\mathbf{A}^\alpha$ a labeled formula. Then we say that $\mathcal{M}\in\mathcal{K}$ satisfies $\mathbf{A}^\alpha$ (written $\mathcal{M}\models\mathbf{A}$), iff $\alpha=\mathsf{T}$ and $\mathcal{M}\models\mathbf{A}$ or $\alpha=\mathsf{F}$ and $\mathcal{M}\not\models\mathbf{A}$.
- **Definition 12.1.9.** Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system, $\mathbf{A}\in\mathcal{L}$ atomic, and $\alpha\in\{\mathsf{T},\mathsf{F}\}$, then we call a $\mathbf{A}^\alpha$ a literal.
- **Intuition:** To satisfy a formula, we make it "true". To satisfy a labeled formula $\mathbf{A}^\alpha$, it must have the truth value α.

▷ **Definition 12.1.10.** For a literal $\mathbf{A}^\alpha$, we call the literal $\mathbf{A}^\beta$ with $\alpha \neq \beta$ the opposite literal (or partner literal).

The idea about literals is that they are atoms (the simplest formulae) that carry around their intended truth value.

Alternative Definition: Literals

▷ **Note:** Literals are often defined without recurring to labeled formulae:

▷ **Definition 12.1.11.** A literal is an atom $\mathbf{A}$ (positive literal) or negated atom $\neg\mathbf{A}$ (negative literal). $\mathbf{A}$ and $\neg\mathbf{A}$ 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 12.1.12.** A formula is in conjunctive normal form (CNF) if it is T or a conjunction of disjunctions of literals: i.e. if it is of the form $\bigwedge_{i=1}^{n}\bigvee_{j=1}^{m_i} l_{ij}$

▷ **Definition 12.1.13.** A formula is in disjunctive normal form (DNF) if it is F or a disjunction of conjunctions of literals: i.e. if it is of the form $\bigvee_{i=1}^{n}\bigwedge_{j=1}^{m_i} l_{ij}$

▷ **Observation 12.1.14.** *Every formula has equivalent formulae in CNF and DNF.*

Video Nuggets covering this chapter can be found at `https://fau.tv/clip/id/23705` and `https://fau.tv/clip/id/23708`.

12.2 Analytical Tableaux

Video Nuggets covering this section can be found at `https://fau.tv/clip/id/23705` and `https://fau.tv/clip/id/23708`.

12.2.1 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 (⇝ models).

▷ **Example 12.2.1.** Tableau calculi try to construct models for labeled formulae:

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

▷ **Idea:** Open branches in saturated tableaux yield models.

▷ **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)

▷ Satisfiable, iff there are open branches (correspond to models)

Tableau calculi develop a formula in a tree-shaped arrangement that represents a case analysis on when a formula can be made true (or false). Therefore the formulae are decorated with upper indices that hold the intended truth value.
On the left we have a refutation tableau that analyzes a negated formula (it is decorated with the intended truth value F). Both branches contain an elementary contradiction $\bot$.

On the right we have a model generation tableau, which analyzes a positive formula (it is decorated with the intended truth value T). This tableau uses the same rules as the refutation tableau, but makes a case analysis of when this formula can be satisfied. In this case we have a closed branch and an open one. The latter corresponds a model.

Now that we have seen the examples, we can write down the tableau rules formally.

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 12.2.2.** The propositional tableau calculus $\mathcal{T}_0$ has two inference rules per connective (one for each possible label)

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

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

▷ **Definition 12.2.3.** 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 12.2.4.** Call a tableau saturated, iff no rule adds new material and a branch closed, iff it ends in $\bot$, else open. A tableau is closed, iff all of its branches are.

In analogy to the $\bot$ at the end of closed branches, we sometimes decorate open branches with a $\Box$ symbol.

These inference rules act on tableaux have to be read as follows: if the formulae over the line appear in a tableau branch, then the branch can be extended by the formulae or branches below the line. There are two rules for each primary connective, and a branch closing rule that adds the special symbol $\bot$ (for unsatisfiability) to a branch.

We use the tableau rules with the convention that they are only applied, if they contribute new material to the branch. This ensures termination of the tableau procedure for propositional logic (every rule eliminates one primary connective).

Definition 12.2.5. We will call a closed tableau with the labeled formula $\mathbf{A}^\alpha$ at the root a tableau refutation for $\mathcal{A}^\alpha$.

The saturated tableau represents a full case analysis of what is necessary to give $\mathbf{A}$ the truth value α; since all branches are closed (contain contradictions) this is impossible.

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

▷ **Definition 12.2.6 ($\mathcal{T}_0$-Theorem/Derivability).** $\mathbf{A}$ is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau with $\mathbf{A}^\mathsf{F}$ at the root.

$\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ derives $\mathbf{A}$ in $\mathcal{T}_0$ ($\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau starting with $\mathbf{A}^\mathsf{F}$ and Φ^T. The tableau with only a branch of $\mathbf{A}^\mathsf{F}$ and Φ^T is called initial for $\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$.

Definition 12.2.7. We will call a tableau refutation for $\mathbf{A}^\mathsf{F}$ a tableau proof for $\mathbf{A}$, since it refutes the possibility of finding a model where $\mathbf{A}$ evaluates to F. Thus $\mathbf{A}$ must evaluate to T in all models, which is just our definition of validity.

Thus the tableau procedure can be used as a calculus for propositional logic. In contrast to the propositional Hilbert calculus it does not prove a theorem $\mathbf{A}$ by deriving it from a set of axioms, but it proves it by refuting its negation. Such calculi are called negative or test calculi. Generally negative calculi have computational advantages over positive ones, since they have a built-in sense of direction.

We have rules for all the necessary connectives (we restrict ourselves to $\wedge$ and $\neg$, since the others can be expressed in terms of these two via the propositional identities above. For instance, we can write $\mathbf{A} \vee \mathbf{B}$ as $\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$, and $\mathbf{A} \Rightarrow \mathbf{B}$ as $\neg\mathbf{A} \vee \mathbf{B}$,. . . .)

We now look at a formulation of propositional logic with fancy variable names. Note that loves(mary, bill) is just a variable name like P or X, which we have used earlier.

A Valid Real-World Example

▷ **Example 12.2.8.** *If Mary loves Bill and John loves Mary, then John loves Mary*

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

This is a closed tableau, so the $\text{loves}(\text{mary},\text{bill})\land\text{loves}(\text{john},\text{mary})\Rightarrow\text{loves}(\text{john},\text{mary})$ is a $\mathcal{T}_0$-theorem.

As we will see, $\mathcal{T}_0$ is sound and complete, so

$$\text{loves}(\text{mary},\text{bill})\land\text{loves}(\text{john},\text{mary})\Rightarrow\text{loves}(\text{john},\text{mary})$$

is valid.

We could have used the unsatisfiability theorem (**??**) here to show that *If Mary loves Bill and John loves Mary* entails *John loves Mary*. But there is a better way to show entailment: we directly use derivability in $\mathcal{T}_0$.

Deriving Entailment in $\mathcal{T}_0$

▷ **Example 12.2.9.** *Mary loves Bill* and *John loves Mary* together entail that *John loves Mary*

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

This is a closed tableau, so $\{\text{loves}(\text{mary},\text{bill}),\text{loves}(\text{john},\text{mary})\}\vdash_{\mathcal{T}_0}\text{loves}(\text{john},\text{mary})$.

Again, as $\mathcal{T}_0$ is sound and complete we have

$$\{\text{loves}(\text{mary},\text{bill}),\text{loves}(\text{john},\text{mary})\}\models\text{loves}(\text{john},\text{mary})$$

Note: We can also use the tableau calculus to try and show entailment (and fail). The nice thing is that the failed proof, we can see what went wrong.

A Falsifiable Real-World Example

▷ **Example 12.2.10.** * *If Mary loves Bill or John loves Mary, then John loves Mary*

Try proving the implication (this fails)

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

Indeed we can make $\mathcal{I}_\varphi(\text{loves}(\text{mary},\text{bill})) = \mathsf{T}$ but $\mathcal{I}_\varphi(\text{loves}(\text{john},\text{mary})) = \mathsf{F}$.

Obviously, the tableau above is saturated, but not closed, so it is not a tableau proof for our initial entailment conjecture. We have marked the literal on the open branch green, since they allow us to read of the conditions of the situation, in which the entailment fails to hold. As we intuitively argued above, this is the situation, where *Mary loves Bill*. In particular, the open branch gives us a variable assignment (marked in green) that satisfies the initial formula. In this case, *Mary loves Bill*, which is a situation, where the entailment fails.

Again, the derivability version is much simpler:

Testing for Entailment in $\mathcal{T}_0$

▷ **Example 12.2.11.** Does *Mary loves Bill or John loves Mary* entail that *John loves Mary*?

$$
\begin{array}{c}
(\text{loves}(\text{mary},\text{bill}) \lor \text{loves}(\text{john},\text{mary}))^{\mathsf{T}}\\
\text{loves}(\text{john},\text{mary})^{\mathsf{F}}\\
\begin{array}{c|c}
\text{loves}(\text{mary},\text{bill})^{\mathsf{T}} & \text{loves}(\text{john},\text{mary})^{\mathsf{T}}\\
 & \bot
\end{array}
\end{array}
$$

This saturated tableau has an open branch that shows that the interpretation with $\mathcal{I}_\varphi(\text{loves}(\text{mary},\text{bill})) = \mathsf{T}$ but $\mathcal{I}_\varphi(\text{loves}(\text{john},\text{mary})) = \mathsf{F}$ falsifies the derivability/entailment conjecture.

We have seen in the examples above that while it is possible to get by with only the connectives $\lor$ and $\neg$, it is a bit unnatural and tedious, since we need to eliminate the other connectives first. In this section, we will make the calculus less frugal by adding rules for the other connectives, without losing the advantage of dealing with a small calculus, which is good making statements about the calculus itself.

12.2.2 Practical Enhancements for Tableaux

The main idea here is to add the new rules as derivable inference rules, i.e. rules that only abbreviate derivations in the original calculus. Generally, adding derivable inference rules does not change the derivation relation of the calculus, and is therefore a safe thing to do. In particular, we will add the following rules to our tableau calculus.

We will convince ourselves that the first rule is derivable, and leave the other ones as an exercise.

Derived Rules of Inference

▷ **Definition 12.2.12.** An inference rule

$$\frac{\mathbf{A}_1 \quad \ldots \quad \mathbf{A}_n}{\mathbf{C}}$$

is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $\mathbf{A}_1, \ldots, \mathbf{A}_n \vdash_{\mathcal{C}} \mathbf{C}$.

▷ **Definition 12.2.13.** We have the following derivable inference rules in $\mathcal{T}_0$:

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

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

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

Michael Kohlhase: Artificial Intelligence 1 372 2025-02-06

With these derived rules, theorem proving becomes quite efficient. With these rules, the tableau (**??**) would have the following simpler form:

Tableaux with derived Rules (example)

Example 12.2.14.

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

Michael Kohlhase: Artificial Intelligence 1 373 2025-02-06

12.2.3 Soundness and Termination of Tableaux

As always we need to convince ourselves that the calculus is sound, otherwise, tableau proofs do not guarantee validity, which we are after. Since we are now in a refutation setting we cannot just show that the inference rules preserve validity: we care about unsatisfiability (which is the dual notion to validity), as we want to show the initial labeled formula to be unsatisfiable. Before we can do this, we have to ask ourselves, what it means to be (un)-satisfiable for a labeled formula or a tableau.

Soundness (Tableau)

▷ **Idea:** A test calculus is refutation sound, iff its inference rules preserve satisfiability and the goal formulae are unsatisfiable.

▷ **Definition 12.2.15.** A labeled formula $\mathbf{A}^\alpha$ is valid under φ, iff $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$.

▷ **Definition 12.2.16.** A tableau $\mathcal{T}$ is satisfiable, iff there is a satisfiable branch $\mathcal{P}$ in $\mathcal{T}$, i.e. if the set of formulae on $\mathcal{P}$ is satisfiable.

▷ **Lemma 12.2.17.** *$\mathcal{T}_0$ rules transform satisfiable tableaux into satisfiable ones.*

▷ **Theorem 12.2.18 (Soundness).** *$\mathcal{T}_0$ is sound, i.e. $\Phi \subseteq wff_0(\mathcal{V}_0)$ valid, if there is a closed tableau $\mathcal{T}$ for Φ^F.*

▷ *Proof:* by contradiction

1. Suppose Φ isfalsifiable $\widehat{=}$ not valid.
2. Then the initial tableau is satisfiable, (Φ^F satisfiable)
3. so $\mathcal{T}$ is satisfiable, by ??.
4. Thus there is a satisfiable branch (by definition)
5. but all branches are closed ($\mathcal{T}$ closed)

▷ **Theorem 12.2.19 (Completeness).** *$\mathcal{T}_0$ is complete, i.e. if $\Phi \subseteq wff_0(\mathcal{V}_0)$ is valid, then there is a closed tableau $\mathcal{T}$ for Φ^F.*

Proof sketch: Proof difficult/interesting; see ??

FAU Michael Kohlhase: Artificial Intelligence 1 374 2025-02-06

Thus we only have to prove ??, this is relatively easy to do. For instance for the first rule: if we have a tableau that contains $(\mathbf{A} \wedge \mathbf{B})^\mathsf{T}$ and is satisfiable, then it must have a satisfiable branch. If $(\mathbf{A} \wedge \mathbf{B})^\mathsf{T}$ is not on this branch, the tableau extension will not change satisfiability, so we can assume that it is on the satisfiable branch and thus $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$ for some variable assignment φ. Thus $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$ and $\mathcal{I}_\varphi(\mathbf{B}) = \mathsf{T}$, so after the extension (which adds the formulae $\mathbf{A}^\mathsf{T}$ and $\mathbf{B}^\mathsf{T}$ to the branch), the branch is still satisfiable. The cases for the other rules are similar.

The next result is a very important one, it shows that there is a procedure (the tableau procedure) that will always terminate and answer the question whether a given propositional formula is valid or not. This is very important, since other logics (like the often-studied first-order logic) does not enjoy this property.

▷ Termination for Tableaux

▷ **Lemma 12.2.20.** *$\mathcal{T}_0$ 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 $\mathbf{A}^\alpha$ 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 12.2.21.** *$\mathcal{T}_0$ induces a decision procedure for validity in PL^0.*

Proof: We combine the results so far

▷ 1. By **??** it is decidable whether $\vdash_{\mathcal{T}_0} \mathbf{A}$
2. By soundness (**??**) and completeness (**??**), $\vdash_{\mathcal{T}_0} \mathbf{A}$ iff $\mathbf{A}$ is valid.

Note: The proof above only works for the "base $\mathcal{T}_0$" because (only) there the rules do not "copy". A rule like

$$\frac{(\mathbf{A} \Leftrightarrow \mathbf{B})^{\mathsf{T}}}{\begin{array}{c|c} \mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{B}^{\mathsf{F}} \end{array}}$$

does, and in particular the number of non-worked-off variables below the line is larger than above the line. For such rules, we would have a more intricate version of μ which – instead of returning a natural number – returns a more complex object; a multiset of numbers. would work here. In our proof we are just assuming that the defined connectives have already eliminated. The tableau calculus basically computes the disjunctive normal form: every branch is a disjunct that is a conjunction of literals. The method relies on the fact that a DNF is unsatisfiable, iff each literal is, i.e. iff each branch contains a contradiction in form of a pair of opposite literals.

12.3 Resolution for Propositional Logic

12.3.1 Resolution for Propositional Logic

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/23712`. The next calculus is a test calculus based on the conjunctive normal form: the resolution calculus. In contrast to the tableau method, it does not compute the normal form as it goes along, but has a pre-processing step that does this and a single inference rule that maintains the normal form. The goal of this calculus is to derive the empty clause, which is unsatisfiable.

Another Test Calculus: Resolution

▷ **Definition 12.3.1.** A clause is a disjunction $l_1^{\alpha_1} \vee \ldots \vee l_n^{\alpha_n}$ of literals. We will use $\Box$ for the "empty" disjunction (no disjuncts) and call it the empty clause. A clause with exactly one literal is called a unit clause.

▷ **Definition 12.3.2 (Resolution Calculus).** The resolution calculus $\mathcal{R}_0$ operates a clause sets via a single inference rule:

$$\frac{P^{\mathsf{T}} \vee \mathbf{A} \quad P^{\mathsf{F}} \vee \mathbf{B}}{\mathbf{A} \vee \mathbf{B}} \mathcal{R}$$

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 12.3.3 (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 12.3.4.** 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 12.3.5 (Transformation into Clause Normal Form).** The CNF transformation calculus CNF_0 consists of the following four inference rules on sets of labeled formulae.

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

▷ **Definition 12.3.6.** We write $CNF_0(\mathbf{A}^\alpha)$ for the set of all clauses derivable from $\mathbf{A}^\alpha$ via the rules above.

FAU Michael Kohlhase: Artificial Intelligence 1 377 2025-02-06

that the **C**-terms in the definition of the inference rules are necessary, since we assumed that the assumptions of the inference rule must match full clauses. The **C** terms are used with the convention that they are optional. So that we can also simplify $(\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}$ to $\mathbf{A}^{\mathsf{T}} \vee \mathbf{B}^{\mathsf{T}}$.
Background: The background behind this notation is that $\mathbf{A}$ and $T \vee \mathbf{A}$ are equivalent for any $\mathbf{A}$. That allows us to interpret the **C**-terms in the assumptions as T and thus leave them out.
The clause normal form translation as we have formulated it here is quite frugal; we have left out rules for the connectives $\vee$, $\Rightarrow$, and $\Leftrightarrow$, relying on the fact that formulae containing these connectives can be translated into ones without before CNF transformation. The advantage of having a calculus with few inference rules is that we can prove meta properties like soundness and completeness with less effort (these proofs usually require one case per inference rule). On the other hand, adding specialized inference rules makes proofs shorter and more readable.
Fortunately, there is a way to have your cake and eat it. Derivable inference rules have the property that they are formally redundant, since they do not change the expressive power of the calculus. Therefore we can leave them out when proving meta-properties, but include them when actually using the calculus.

Derived Rules of Inference

▷ **Definition 12.3.7.** An inference rule

$$\frac{\mathbf{A}_1 \quad \ldots \quad \mathbf{A}_n}{\mathbf{C}}$$

is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $\mathbf{A}_1, \ldots, \mathbf{A}_n \vdash_{\mathcal{C}} \mathbf{C}$.

▷ **Idea:** Derived rules make derivations shorter.

▷ **Example 12.3.8.**

$$\cfrac{\cfrac{\cfrac{\mathbf{C} \vee (\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}}{\mathbf{C} \vee (\neg\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}}}{\mathbf{C} \vee \neg\mathbf{A}^{\mathsf{T}} \vee \mathbf{B}^{\mathsf{T}}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}} \vee \mathbf{B}^{\mathsf{T}}} \quad \rightsquigarrow \quad \frac{\mathbf{C} \vee (\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}} \vee \mathbf{B}^{\mathsf{T}}}$$

▷ **Other Derived CNF Rules:**

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

With these derivable rules, theorem proving becomes quite efficient. To get a better understanding of the calculus, we look at an example: we prove an axiom of the Hilbert Calculus we have studied above.

Example: Proving Axiom S with Resolution

▷ **Example 12.3.9.** 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 12.3.10.** 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	$\Box$	resolve 7.1 with 3.1

Clause Set Simplification

▷ **Observation:** Let Δ be a clause set, l a literal with $l \in \Delta$ (unit clause), and Δ' be Δ where

- ▷ all clauses $l \vee C$ have been removed and
- ▷ and all clauses $\bar{l} \vee C$ have been shortened to C.

Then Δ is satisfiable, iff Δ' is. We call Δ' the clause set simplification of Δ wrt. l.

▷ **Corollary 12.3.11.** *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)

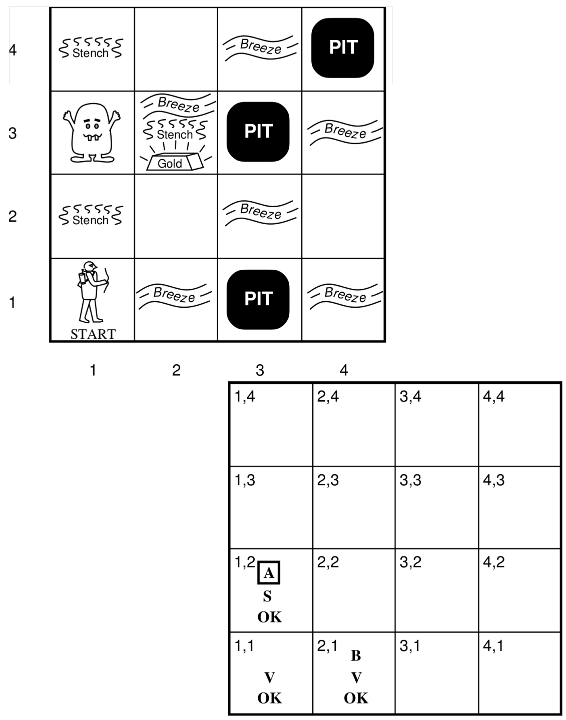
12.3.2 Killing a Wumpus with Propositional Inference

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/23713`.

Let us now consider an extended example, where we also address the question how inference in PL^0 – here resolution is embedded into the rational agent metaphor we use in AI-1: we come back to the Wumpus world.

Applying Propositional Inference: Where is the Wumpus?

▷ **Example 12.3.12 (Finding the Wumpus).** The situation and what the agent knows



- ▷ 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)

Before we come to the general mechanism, we will go into how we would "convince ourselves that the Wumpus is in $[1,3]$.

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)

The first in is to compute the clause normal form of the relevant knowledge.

And Now Using Resolution Conventions

▷ We obtain the clause set Δ composed of the following clauses:

- ▷ **Propositions whose value we know**: ${S_{1,1}}^{\mathsf{F}}$, ${W_{1,1}}^{\mathsf{F}}$, ${S_{2,1}}^{\mathsf{F}}$, ${W_{2,1}}^{\mathsf{F}}$, ${S_{1,2}}^{\mathsf{T}}$, ${W_{1,2}}^{\mathsf{F}}$
- ▷ **Knowledge about the Wumpus and smell**:

from	clauses
R_1	${S_{1,1}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{F}}$, ${S_{1,1}}^{\mathsf{T}} \vee {W_{1,2}}^{\mathsf{F}}$, ${S_{1,1}}^{\mathsf{T}} \vee {W_{2,1}}^{\mathsf{F}}$
R_2	${S_{2,1}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{F}}$, ${S_{2,1}}^{\mathsf{T}} \vee {W_{2,1}}^{\mathsf{F}}$, ${S_{2,1}}^{\mathsf{T}} \vee {W_{2,2}}^{\mathsf{F}}$, ${S_{2,1}}^{\mathsf{T}} \vee {W_{3,1}}^{\mathsf{F}}$
R_3	${S_{1,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{F}}$, ${S_{1,2}}^{\mathsf{T}} \vee {W_{1,2}}^{\mathsf{F}}$, ${S_{1,2}}^{\mathsf{T}} \vee {W_{2,2}}^{\mathsf{F}}$, ${S_{1,2}}^{\mathsf{T}} \vee {W_{1,3}}^{\mathsf{F}}$
R_4	${S_{1,2}}^{\mathsf{F}} \vee {W_{1,3}}^{\mathsf{T}} \vee {W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$

- ▷ **Negated goal formula**: ${W_{1,3}}^{\mathsf{F}}$

Given this clause normal form, we only need to find generate empty clause via repeated applications of the resolution rule.

Resolution Proof Killing the Wumpus!

▷ **Example 12.3.13 (Where is the Wumpus).** We show a derivation that proves that he is in $(1,3)$.

- ▷ *Assume the Wumpus is not in $(1,3)$. Then either there's no stench in $(1,2)$, or the Wumpus is in some other neigbor cell of $(1,2)$.*
  - ▷ Parents: ${W_{1,3}}^{\mathsf{F}}$ and ${S_{1,2}}^{\mathsf{F}} \vee {W_{1,3}}^{\mathsf{T}} \vee {W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.
  - ▷ Resolvent: ${S_{1,2}}^{\mathsf{F}} \vee {W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.
- ▷ *There's a stench in $(1,2)$, so it must be another neighbor.*
  - ▷ Parents: ${S_{1,2}}^{\mathsf{T}}$ and ${S_{1,2}}^{\mathsf{F}} \vee {W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.
  - ▷ Resolvent: ${W_{2,2}}^{\mathsf{T}} \vee {W_{1,1}}^{\mathsf{T}}$.

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

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

Now that we have seen how we can use propositional inference to derive consequences of the percepts and world knowledge, let us come back to the question of a general mechanism for agent functions with propositional inference.

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

- **Question:** Where did the ${W_{1,3}}^{\mathsf{F}}$ come from?
- **Observation 12.3.14.** *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 12.3.15 (Back to the Wumpus).** In ??, the agent is in $[1,2]$, it has perceived stench, and the possible actions include shoot, and 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.

Admittedly, the search framework from **??** does not quite cover the agent function we have here, since that assumes that the world is fully observable, which the Wumpus world is emphatically not. But it already gives us a good impression of what would be needed for the "general mechanism".

12.4 Conclusion

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 $\mathrm{KB} \models \mathbf{A}$ by showing that $\mathrm{KB} \cup \{\neg\mathbf{A}\}$ is unsatisfiable.

Excursion: A full analysis of any calculus needs a completeness proof. We will not cover this in AI-1, but provide one for the calculi introduced so far in**??**.

Chapter 13

Propositional Reasoning: SAT Solvers

13.1 Introduction

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25019`.

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 13.1.1.** The SAT problem (SAT): Given a propositional formula $\mathbf{A}$, decide whether or not $\mathbf{A}$ is satisfiable. We denote the class of all SAT problems with SAT
- ▷ The SAT problem was the first problem proved to be $\mathbf{NP}$-complete!
- ▷ $\mathbf{A}$ is commonly assumed to be in CNF. This is without loss of generality, because any $\mathbf{A}$ 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 13.1.2.** Tools addressing SAT are commonly referred to as SAT solvers.

▷ **Recall:** To decide whether $\mathrm{KB} \models \mathbf{A}$, decide satisfiability of $\theta := \mathrm{KB} \cup \{\neg\mathbf{A}\}$: θ is unsatisfiable iff $\mathrm{KB} \models \mathbf{A}$.

▷ **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, and v specifying the permissible combined assignments to u and v. One extension is to allow constraints of higher arity.

▷ **Observation 13.1.3 (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 13.1.4 (Encoding CSP as SAT).** *Given any constraint network $\mathcal{C}$, we can in low order polynomial time construct a CNF formula $\mathbf{A}(\mathcal{C})$ that is satisfiable iff $\mathcal{C}$ is solvable.*

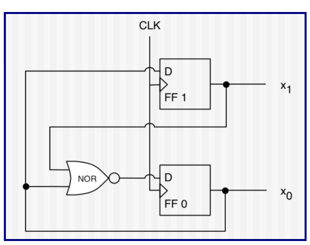
▷ *Proof:* We design a formula, relying on known transformation to CNF

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 13.1.5 (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**: $y_1{}^{\mathsf{F}} \vee x_0{}^{\mathsf{T}}$, $y_1{}^{\mathsf{T}} \vee x_0{}^{\mathsf{F}}$, $y_0{}^{\mathsf{T}} \vee x_1{}^{\mathsf{T}} \vee x_0{}^{\mathsf{T}}$, $y_0{}^{\mathsf{F}} \vee x_1{}^{\mathsf{F}}$, $y_0{}^{\mathsf{F}} \vee x_0{}^{\mathsf{F}}$, $x_1{}^{\mathsf{F}} \vee x_0{}^{\mathsf{F}}$, $y_1{}^{\mathsf{T}}$, $y_0{}^{\mathsf{T}}$.

▷ **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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25026`.

The DPLL Procedure

▷ **Definition 13.2.1.** 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 $\square \in \Delta'$ **then return** "unsatisfiable"

```
 if Δ' = {} 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'') ≠ ''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 13.2.2 (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. $P \mapsto \mathsf{F}$ $Q^{\mathsf{T}} ; Q^{\mathsf{F}}$	2b. $P \mapsto \mathsf{T}$ Q^{F}
3a. UP Rule: $Q \mapsto \mathsf{T}$ $\Box$ **returning "unsatisfiable"**	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)

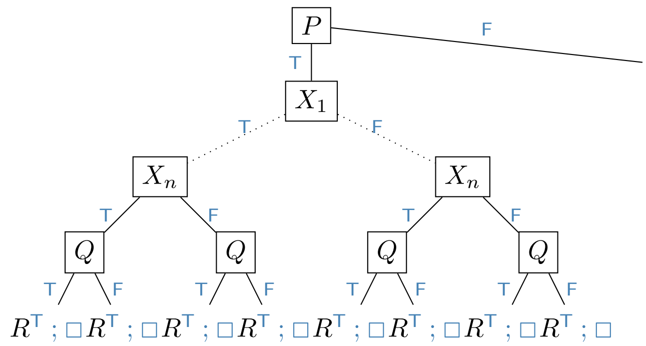
▷ **Observation:** Sometimes UP is all we need.

▷ **Example 13.2.3.** Let $\Delta := Q^{\mathsf{F}} \vee P^{\mathsf{F}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{F}} ; Q^{\mathsf{T}} \vee S^{\mathsf{F}} ; R^{\mathsf{T}} \vee S^{\mathsf{F}} ; S^{\mathsf{T}}$

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

DPLL: Example (Redundance1)

▷ **Example 13.2.4.** 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 $\Theta := X_1{}^T \vee \ldots \vee X_n{}^T ; X_1{}^F \vee \ldots \vee X_n{}^F$



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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/27022`.

In the last slide we have discussed the semantic properties of the DPLL procedure: DPLL is (refutation) sound and complete. Note that this is a theoretical resultin the sense that the algorithm is, but that does not mean that a particular implementation of DPLL might not contain bugs that affect sounds and completeness.

In the satisfiable case, DPLL returns a satisfying variable assignment, which we can check (in low-order polynomial time) but in the unsatisfiable case, it just reports on the fact that it has tried all branches and found nothing. This is clearly unsatisfactory, and we will address this situation now by presenting a way that DPLL can output a resolution proof in the unsatisfiable case.

UP $\widehat{=}$ Unit Resolution

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

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

▷ **Definition 13.3.1 (Unit Resolution).** Unit resolution (UR) is the test calculus consisting of the following inference rule:

$$\frac{C \vee P^{\alpha} \quad P^{\beta} \quad \alpha \neq \beta}{C} \text{UR}$$

▷ Unit propagation $\widehat{=}$ resolution restricted to cases where one parent is unit clause.

▷ **Observation 13.3.2 (Soundness).** UR *is refutation sound.* (*since resolution is*)

▷ **Observation 13.3.3 (Completeness).** UR *is not refutation complete (alone).*

▷ **Example 13.3.4.** $P^{\mathsf{T}} \vee Q^{\mathsf{T}}$; $P^{\mathsf{T}} \vee Q^{\mathsf{F}}$; $P^{\mathsf{F}} \vee Q^{\mathsf{T}}$; $P^{\mathsf{F}} \vee Q^{\mathsf{F}}$ 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 13.3.5.** 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 13.3.6.** *If DPLL returns "unsatisfiable" on Δ, then $\Delta \vdash_{\mathcal{R}_0} \square$ with a resolution proof whose length is at most the number of decisions.*

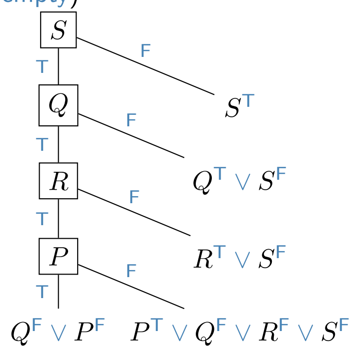
▷ *Proof:* Consider first DPLL without UP

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.

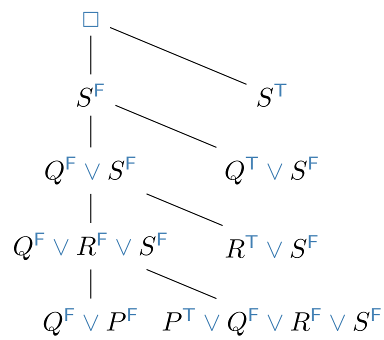
DPLL vs. Resolution: Example (Vanilla2)

- **Observation:** The proof of ?? is constructive, so we can use it as a method to read of a resolution proof from a DPLL trace.
- **Example 13.3.7.** We follow the steps in the proof of ?? for $\Delta := Q^F \vee P^F ; P^T \vee Q^F \vee R^F \vee S^F ; Q^T \vee S^F ; R^T \vee S^F ; S^T$

DPLL: (Without UP; leaves annotated with clauses that became empty)



Resolution proof from that DPLL tree:



- **Intuition:** From a (top-down) DPLL tree, we generate a (bottom-up) resolution proof.

For reference, we give the full proof here.

Theorem 13.3.8. *If DPLL returns "unsatisfiable" on a clause set Δ, then $\Delta \vdash_{\mathcal{R}_0} \Box$ with a $\mathcal{R}_0$-derivation whose length is at most the number of decisions.*

Proof: Consider first DPLL with no unit propagation.

1. If the search tree is not empty, then there exists a leaf node N, i.e., a node associated to proposition X so that, for each value of X, the partial assignment directly results in an empty clause.
2. Denote the parent decisions of N by $L_1, \ldots, L_k$, where L_i is a literal for proposition X_i and the search node containing X_i is N_i.
3. Denote the empty clause for X by $C(N, X)$, and denote the empty clause for X^F by $C(N, X^F)$.
4. For each $x \in \{X^T, X^F\}$ we have the following properties:
   1. $x^F \in C(N, x)$; and
   2. $C(N, x) \subseteq \{x^F, \overline{L_1}, \ldots, \overline{L_k}\}$.

   Due to , we can resolve $C(N, X)$ with $C(N, X^F)$; denote the outcome clause by $C(N)$.
5. We obviously have that (1) $C(N) \subseteq \{\overline{L_1}, \ldots, \overline{L_k}\}$.
6. The proof now proceeds by removing N from the search tree and attaching $C(N)$ at the L_k branch of N_k, in the role of $C(N_k, L_k)$ as above. Then we select the next leaf node N' and iterate the argument; once the tree is empty, by (1) we have derived the empty clause. What we need to show is that, in each step of this iteration, we preserve the properties (a) and (b) for all leaf nodes. Since we did not change anything in other parts of the tree, the only node we need to show this for is $N' := N_k$.
7. Due to (1), we have (b) for N_k. But we do not necessarily have (a): $C(N) \subseteq \{\overline{L_1}, \ldots, \overline{L_k}\}$, but there are cases where $\overline{L_k} \notin C(N)$ (e.g., if X_k is not contained in any clause and thus

branching over it was completely unnecessary). If so, however, we can simply remove N_k and all its descendants from the tree as well. We attach $C(N)$ at the $L_{(k-1)}$ branch of $N_{(k-1)}|$, in the role of $C(N_{(k-1)}, L_{(k-1)})$. If $\overline{L_{(k-1)}} \in C(N)$ then we have (a) for $N' := N_{(k-1)}$ and can stop. If $L_{(k-1)}{}^{\mathsf{F}} \notin C(N)$, then we remove $N_{(k-1)}$ and so forth, until either we stop with (a), or have removed N_1 and thus must already have derived the empty clause (because $C(N) \subseteq \{\overline{L_1}, \ldots, \overline{L_k}\} \backslash \{\overline{L_1}, \ldots, \overline{L_k}\}$).

8. 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: 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 13.3.9.** 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)

Excursion: Practical SAT solvers use a technique called CDCL that analyzes failure and learns from that in terms of inferred clauses. Unfortunately, we cannot cover this in AI-1.??.

13.4 Conclusion

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25090`.

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 13.4.1.** 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 403) 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)?

Suggested Reading:

- *Chapter 7: Logical Agents*, Section 7.6.1 [RN09].
  - Here, RN describe DPLL, i.e., basically what I cover under "The Davis-Putnam (Logemann-Loveland) Procedure".
  - That's the only thing they cover of this Chapter's material. (And they even mark it as "can be skimmed on first reading".)
  - This does not do the state of the art in SAT any justice.
- *Chapter 7: Logical Agents*, Sections 7.6.2, 7.6.3, and 7.7 [RN09].
  - Sections 7.6.2 and 7.6.3 say a few words on local search for SAT, which I recommend as additional background reading. Section 7.7 describes in quite some detail how to build an agent using propositional logic to take decisions; nice background reading as well.

Chapter 14

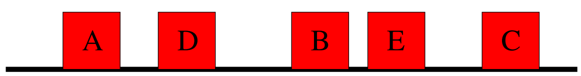
First-Order Predicate Logic

14.1 Motivation: A more Expressive Language

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25091`.

Let's Talk About Blocks, Baby ...

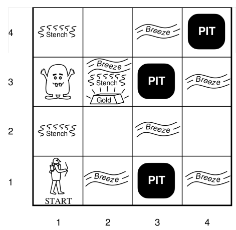
- **Question:** What do you see here?



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

Let's Talk About the Wumpus Instead?

▷ 

Percepts: $[\mathit{Stench}, \mathit{Breeze}, \mathit{Glitter}, \mathit{Bump}, \mathit{Scream}]$

- ▷ Cell adjacent to Wumpus: Stench (else: None).
- ▷ Cell adjacent to Pit: Breeze (else: None).
- ▷ Cell that contains gold: $\mathit{Glitter}$ (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 14.1.2 (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 14.1.3.**

$$\forall n.\mathrm{gt}(n,2) \Rightarrow \neg(\exists a,b,c.\mathrm{eq}(\mathrm{plus}(\mathrm{pow}(a,n),\mathrm{pow}(b,n)),\mathrm{pow}(c,n)))$$

Read: *Forall $n > 2$, there are no a, b, c, such that $a^n + b^n = c^n$* (Fermat's last theorem)

▷ **Theorem proving in PL1:** Arbitrary theorems, in principle.

- ▷ Fermat's last theorem is of course infeasible, but interesting theorems can and have been proved automatically.
- ▷ See `http://en.wikipedia.org/wiki/Automated_theorem_proving`.
- ▷ **Note**: Need to axiomatize "Plus", "PowerOf", "Equals". See `http://en.wikipedia.org/wiki/Peano_axioms`

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.

▷ Prominent PL1 fragment: Web Ontology Language OWL.

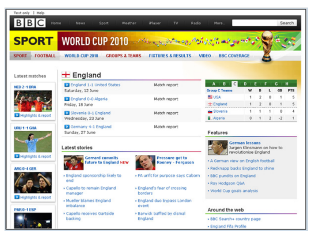
▷ Prominent data set: The WWW. (semantic web)

▷ **Assorted quotes on Semantic Web and OWL:**

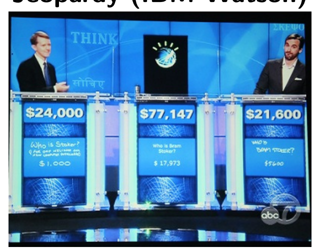
- *The brain of humanity.*
- *The Semantic Web will never work.*
- *A TRULY meaningful way of interacting with the Web may finally be here: the Semantic Web. The idea was proposed 10 years ago. A triumvirate of internet heavyweights – Google, Twitter, and Facebook – are making it real.*

(A Few) Semantic Technology Applications

Web Queries

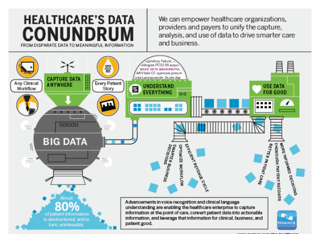


Jeopardy (IBM Watson)



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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25093`.
First-order logic is the most widely used formal systems for modelling knowledge and inference processes. It strikes a very good bargain in the trade-off between expressivity and conceptual and computational complexity. To many people first-order logic is "the logic", i.e. the only logic worth considering, its applications range from the foundations of mathematics to natural language semantics.

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

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/25094`.
The syntax and semantics of first-order logic is systematically organized in two distinct layers: one for truth values (like in propositional logic) and one for individuals (the new, distinctive feature of first-order logic).
The first step of defining a formal language is to specify the alphabet, here the first-order signatures and their components.

PL^1 Syntax (Signature and Variables)

- **Definition 14.2.1.** 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 14.2.2.** 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: $\Sigma^p{}_k = \{p, q, r, \ldots\}$ (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 14.2.3.** We assume a set of individual variables: $\mathcal{V}_\iota := \{X, Y, Z, \ldots\}$. (countably ∞)

We make the deliberate, but non-standard design choice here to include Skolem constants into the signature from the start. These are used in inference systems to give names to objects and construct witnesses. Other than the fact that they are usually introduced by need, they work exactly like regular constants, which makes the inclusion rather painless. As we can never predict how many Skolem constants we are going to need, we give ourselves countably infinitely many for every arity. Our supply of individual variables is countably infinite for the same reason.
The formulae of first-order logic are built up from the signature and variables as terms (to represent individuals) and proposition (to represent proposition). The latter include the connectives from PL^0, but also quantifiers.

PL^1 Syntax (Formulae)

▷ **Definition 14.2.4.** Terms: $\mathbf{A} \in \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$ (denote individuals)

- ▷ $\mathcal{V}_\iota \subseteq \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$,
- ▷ if $f \in \Sigma_k^f$ and $\mathbf{A}^i \in \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $f(\mathbf{A}^1, \ldots, \mathbf{A}^k) \in \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$.

▷ **Definition 14.2.5.** First-order propositions: $\mathbf{A} \in \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota)$: (denote truth values)

- ▷ if $p \in \Sigma^p{}_k$ and $\mathbf{A}^i \in \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $p(\mathbf{A}^1, \ldots, \mathbf{A}^k) \in \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota)$,
- ▷ if $\mathbf{A}, \mathbf{B} \in \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota)$ and $X \in \mathcal{V}_\iota$, then $T, \mathbf{A} \wedge \mathbf{B}, \neg \mathbf{A}, \forall X.\mathbf{A} \in \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota)$. $\forall$ is a binding operator called the universal quantifier.

▷ **Definition 14.2.6.** 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 14.2.7.** 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 14.2.8.** Call formulae without connectives or quantifiers atomic else complex.

Note: We only need e.g. conjunction, negation, and universal quantifier, all other logical constants can be defined from them (as we will see when we have fixed their interpretations).

Alternative Notations for Quantifiers

Here	Elsewhere
$\forall x.\mathbf{A}$	$\bigwedge x.\mathbf{A}$ $\quad (x)\mathbf{A}$
$\exists x.\mathbf{A}$	$\bigvee x.\mathbf{A}$

The introduction of quantifiers to first-order logic brings a new phenomenon: variables that are under the scope of a quantifiers will behave very differently from the ones that are not. Therefore we build up a vocabulary that distinguishes the two.

Free and Bound Variables

▷ **Definition 14.2.9.** We call an occurrence of a variable X bound in a formula $\mathbf{A}$ (otherwise free), iff it occurs in a sub-formula $\forall X.\mathbf{B}$ of $\mathbf{A}$.

For a formula $\mathbf{A}$, we will use $\mathrm{BVar}(\mathbf{A})$ (and $\mathrm{free}(\mathbf{A})$) for the set of bound (free) variables of $\mathbf{A}$, i.e. variables that have a free/bound occurrence in $\mathbf{A}$.

▷ **Definition 14.2.10.** We define the set $\mathrm{free}(\mathbf{A})$ of free variables of a formula $\mathbf{A}$:

$$\begin{aligned}
&\mathrm{free}(X) := \{X\}\\
&\mathrm{free}(f(\mathbf{A}_1,\ldots,\mathbf{A}_n)) := \textstyle\bigcup_{1\leq i\leq n}\mathrm{free}(\mathbf{A}_i)\\
&\mathrm{free}(p(\mathbf{A}_1,\ldots,\mathbf{A}_n)) := \textstyle\bigcup_{1\leq i\leq n}\mathrm{free}(\mathbf{A}_i)\\
&\mathrm{free}(\neg\mathbf{A}) := \mathrm{free}(\mathbf{A})\\
&\mathrm{free}(\mathbf{A}\wedge\mathbf{B}) := \mathrm{free}(\mathbf{A})\cup\mathrm{free}(\mathbf{B})\\
&\mathrm{free}(\forall X.\mathbf{A}) := \mathrm{free}(\mathbf{A})\backslash\{X\}
\end{aligned}$$

▷ **Definition 14.2.11.** We call a formula $\mathbf{A}$ closed or ground, iff $\mathrm{free}(\mathbf{A}) = \emptyset$. We call a closed proposition a sentence, and denote the set of all ground term with $\mathit{cwff}_\iota(\Sigma_\iota)$ and the set of sentences with $\mathit{cwff}_o(\Sigma_\iota)$.

▷ **Axiom 14.2.12.** *Bound variables can be renamed, i.e. any subterm $\forall X.\mathbf{B}$ of a formula $\mathbf{A}$ can be replaced by $\mathbf{A}' := (\forall Y.\mathbf{B}')$, where $\mathbf{B}'$ arises from $\mathbf{B}$ by replacing all $X \in \mathrm{free}(\mathbf{B})$ with a new variable Y that does not occur in $\mathbf{A}$. We call $\mathbf{A}'$ an alphabetical variant of $\mathbf{A}$ – and the other way around too.*

We will be mainly interested in (sets of) sentences – i.e. closed propositions – as the representations of meaningful statements about individuals. Indeed, we will see below that free variables do not gives us expressivity, since they behave like constants and could be replaced by them in all situations, except the recursive definition of quantified formulae. Indeed in all situations where variables occur freely, they have the character of metavariables, i.e. syntactic placeholders that can be instantiated with terms when needed in a calculus.

The semantics of first-order logic is a Tarski-style set-theoretic semantics where the atomic syntactic entities are interpreted by mapping them into a well-understood structure, a first-order universe that is just an arbitrary set.

Semantics of PL^1 (Models)

- **Definition 14.2.13.** 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 14.2.14.** An interpretation $\mathcal{I}$ assigns values to constants, e.g.
  - $\mathcal{I}(\neg)\colon \mathcal{D}_0 \to \mathcal{D}_0$ with $\mathsf{T} \mapsto \mathsf{F}$, $\mathsf{F} \mapsto \mathsf{T}$, and $\mathcal{I}(\wedge) = \ldots$ (as in PL^0)
  - $\mathcal{I}\colon \Sigma^f_k \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (interpret function symbols as arbitrary functions)
  - $\mathcal{I}\colon {\Sigma^p}_k \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicates as arbitrary relations)
- **Definition 14.2.15.** A variable assignment $\varphi\colon \mathcal{V}_\iota \to \mathcal{D}_\iota$ maps variables into the domain.
- **Definition 14.2.16.** A model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$ of PL^1 consists of a domain $\mathcal{D}_\iota$ and an interpretation $\mathcal{I}$.

We do not have to make the domain of truth values part of the model, since it is always the same; we determine the model by choosing a domain and an interpretation functiong.

Given a first-order model, we can define the evaluation function as a homomorphism over the construction of formulae.

Semantics of PL^1 (Evaluation)

- **Definition 14.2.17.** 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
    - $\mathcal{I}_\varphi(f(\mathbf{A}_1, \ldots, \mathbf{A}_k)) := \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_k))$
  - $\mathcal{I}_\varphi\colon \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota) \to \mathcal{D}_0$ assigns values to formulae:
    - $\mathcal{I}_\varphi(T) = \mathcal{I}(T) = \mathsf{T}$,
    - $\mathcal{I}_\varphi(\neg \mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathbf{A}))$
    - $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathbf{A}), \mathcal{I}_\varphi(\mathbf{B}))$ (just as in PL^0)
    - $\mathcal{I}_\varphi(p(\mathbf{A}_1, \ldots, \mathbf{A}_k)) := \mathsf{T}$, iff $\langle \mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_k) \rangle \in \mathcal{I}(p)$
    - $\mathcal{I}_\varphi(\forall X.\mathbf{A}) := \mathsf{T}$, iff $\mathcal{I}_{\varphi,[\mathsf{a}/X]}(\mathbf{A}) = \mathsf{T}$ for all $\mathsf{a} \in \mathcal{D}_\iota$.
- **Definition 14.2.18 (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 $\{(Y,a) \in \varphi \mid Y \neq X\} \cup \{(X,a)\}$: $\varphi,[a/X]$ coincides with φ off X, and gives the result a there.

The only new (and interesting) case in this definition is the quantifier case, there we define the value of a quantified formula by the value of its scope – *but with an extension of the incoming variable assignment*. Note that by passing to the scope $\mathbf{A}$ of $\forall x.\mathbf{A}$, the occurrences of the variable x in $\mathbf{A}$ that were bound in $\forall x.\mathbf{A}$ become free and are amenable to evaluation by the variable

assignment $\psi := \varphi,[a/X]$. Note that as an extension of φ, the assignment ψ supplies exactly the right value for x in $\mathbf{A}$. This variability of the variable assignment in the definition of the value function justifies the somewhat complex setup of first-order evaluation, where we have the (static) interpretation function for the symbols from the signature and the (dynamic) variable assignment for the variables.
Note furthermore, that the value $\mathcal{I}_\varphi(\exists x.\mathbf{A})$ of $\exists x.\mathbf{A}$, which we have defined to be $\neg(\forall x.\neg\mathbf{A})$ is true, iff it is not the case that $\mathcal{I}_\varphi(\forall x.\neg\mathbf{A}) = \mathcal{I}_\psi(\neg\mathbf{A}) = \mathsf{F}$ for all $a \in \mathcal{D}_\iota$ and $\psi := \varphi,[a/X]$. This is the case, iff $\mathcal{I}_\psi(\mathbf{A}) = \mathsf{T}$ for some $a \in \mathcal{D}_\iota$. So our definition of the existential quantifier yields the appropriate semantics.

Semantics Computation: Example

▷ **Example 14.2.19.** We define an instance of first-order logic:

- ▷ Signature: Let $\Sigma_0^f := \{j, m\}$, $\Sigma_1^f := \{f\}$, and $\Sigma^p{}_2 := \{o\}$
- ▷ Universe: $\mathcal{D}_\iota := \{J, M\}$
- ▷ Interpretation: $\mathcal{I}(j) := J$, $\mathcal{I}(m) := M$, $\mathcal{I}(f)(J) := M$, $\mathcal{I}(f)(M) := M$, and $\mathcal{I}(o) := \{(M,J)\}$.

Then $\forall X.o(f(X), X)$ is a sentence and with $\psi := \varphi,[a/X]$ 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

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/25156`.

We will now turn our attention to substitutions, special formula-to-formula mappings that operationalize the intuition that (individual) variables stand for arbitrary terms.

Substitutions on Terms

▷ **Intuition:** If $\mathbf{B}$ is a term and X is a variable, then we denote the result of systematically replacing all occurrences of X in a term $\mathbf{A}$ by $\mathbf{B}$ with $[\mathbf{B}/X](\mathbf{A})$.

▷ **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 14.2.20.** Let $wfe(\Sigma, \mathcal{V})$ be an expression language, then we call $\sigma : \mathcal{V} \to wfe(\Sigma, \mathcal{V})$ a substitution, iff the support $\mathrm{supp}(\sigma) := \{X \mid (X,\mathbf{A}) \in \sigma, X \neq \mathbf{A}\}$ of σ is finite. We denote the empty substitution with ϵ.

▷ **Definition 14.2.21 (Substitution Application).** We define substitution application by

- ▹ $\sigma(c) = c$ for $c \in \Sigma$
- ▹ $\sigma(X) = \mathbf{A}$, iff $X \in \mathcal{V}$ and $(X,\mathbf{A}) \in \sigma$.
- ▹ $\sigma(f(\mathbf{A}_1, \ldots, \mathbf{A}_n)) = f(\sigma(\mathbf{A}_1), \ldots, \sigma(\mathbf{A}_n))$,
- ▹ $\sigma(\forall X.\mathbf{A}) = \forall X.\sigma_{-X}(\mathbf{A})$. (∃ analogous)

▷ **Example 14.2.22.** $[a/x], [f(b)/y], [a/z]$ instantiates $g(x, y, h(z))$ to $g(a, f(b), h(a))$.

The extension of a substitution is an important operation, which you will run into from time to time. Given a substitution σ, a variable x, and an expression $\mathbf{A}$, $\sigma,[\mathbf{A}/x]$ extends σ with a new value for x. The intuition is that the values right of the comma overwrite the pairs in the substitution on the left, which already has a value for x, even though the representation of σ may not show it.

Substitution Extension

▷ **Definition 14.2.23 (Substitution Extension).** Let σ be a substitution, then we denote the extension of σ with $[\mathbf{A}/X]$ by $\sigma,[\mathbf{A}/X]$ and define it as $\{(Y,\mathbf{B}) \in \sigma \mid Y \neq X\} \cup \{(X,\mathbf{A})\}$: $\sigma,[\mathbf{A}/X]$ coincides with σ off X, and gives the result $\mathbf{A}$ there.

▷ **Note:** If σ is a substitution, then $\sigma,[\mathbf{A}/X]$ is also a substitution.

▷ We also need the dual operation: removing a variable from the support:

▷ **Definition 14.2.24.** We can discharge a variable X from a substitution σ by setting $\sigma_{-X} := \sigma,[X/X]$.

Note that the use of the comma notation for substitutions defined in ?? is consistent with substitution extension. We can view a substitution $[a/x], [f(b)/y]$ as the extension of the empty substitution (the identity function on variables) by $[f(b)/y]$ and then by $[a/x]$. Note furthermore, that substitution extension is not commutative in general.
For first-order substitutions we need to extend the substitutions defined on terms to act on propositions. This is technically more involved, since we have to take care of bound variables.

Substitutions on Propositions

▷ **Problem:** We want to extend substitutions to propositions, in particular to quantified formulae: What is $\sigma(\forall X.\mathbf{A})$?

▷ **Idea:** σ should not instantiate bound variables. ($[\mathbf{A}/X](\forall X.\mathbf{B}) = \forall \mathbf{A}.\mathbf{B}'$ ill-formed)

▷ **Definition 14.2.25.** $\sigma(\forall X.\mathbf{A}) := (\forall X.\sigma_{-X}(\mathbf{A}))$.

▷ **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 14.2.26 (Capture-Avoiding Substitution Application).** Let σ be a substitution, $\mathbf{A}$ a formula, and $\mathbf{A}'$ an alphabetic variant of $\mathbf{A}$, such that $\mathrm{intro}(\sigma) \cap \mathrm{BVar}(\mathbf{A}) = \emptyset$. Then we define capture-avoiding substitution application via $\sigma(\mathbf{A}) := \sigma(\mathbf{A}')$.

We now introduce a central tool for reasoning about the semantics of substitutions: the "substitution value Lemma", which relates the process of instantiation to (semantic) evaluation. This result will be the motor of all soundness proofs on axioms and inference rules acting on variables via substitutions. In fact, any logic with variables and substitutions will have (to have) some form of a substitution value Lemma to get the meta-theory going, so it is usually the first target in any development of such a logic. We establish the substitution-value Lemma for first-order logic in two steps, first on terms, where it is very simple, and then on propositions.

Substitution Value Lemma for Terms

▷ **Lemma 14.2.27.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be terms, then* $\mathcal{I}_\varphi([\mathbf{B}/X]\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$, *where* $\psi = \varphi,[\mathcal{I}_\varphi(\mathbf{B})/X]$.

▷ *Proof:* by induction on the depth of $\mathbf{A}$:

1. depth=0 *Then* $\mathbf{A}$ *is a variable (say* Y*), or constant, so we have three cases*
   1.1. $\mathbf{A} = Y = X$
      1.1.1. then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](X)) = \mathcal{I}_\varphi(\mathbf{B}) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(\mathbf{A})$.
   1.2. $\mathbf{A} = Y \neq X$
      1.2.1. then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(\mathbf{A})$.
   1.3. $\mathbf{A}$ 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 $\mathbf{A} = f(\mathbf{A}_1,\ldots,\mathbf{A}_n)$ and we have

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

      by induction hypothesis
   2.2. This completes the induction step, and we have proven the assertion.

Substitution Value Lemma for Propositions

▷ **Lemma 14.2.28.** $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\psi(\mathbf{A})$, *where* $\psi = \varphi,[\mathcal{I}_\varphi(\mathbf{B})/X]$.

▷ *Proof:* by induction on the number n of connectives and quantifiers in $\mathbf{A}$:

1. $n = 0$
   1.1. then $\mathbf{A}$ is an atomic proposition, and we can argue like in the induction step of the substitution value lemma for terms.
2. $n > 0$ and $\mathbf{A} = \neg\mathbf{B}$ or $\mathbf{A} = \mathbf{C} \circ \mathbf{D}$
   2.1. Here we argue like in the induction step of the term lemma as well.
3. $n > 0$ and $\mathbf{A} = \forall Y.\mathbf{C}$ where (WLOG) $X \neq Y$ *(otherwise rename)*
   3.1. then $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\psi(\forall Y.\mathbf{C}) = \mathsf{T}$, iff $\mathcal{I}_{\psi,[a/Y]}(\mathbf{C}) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$.
   3.2. But $\mathcal{I}_{\psi,[a/Y]}(\mathbf{C}) = \mathcal{I}_{\varphi,[a/Y]}([\mathbf{B}/X](\mathbf{C})) = \mathsf{T}$, by induction hypothesis.
   3.3. So $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\varphi(\forall Y.[\mathbf{B}/X](\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\forall Y.\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}))$

To understand the proof fully, you should think about where the *WLOG* – it stands for without loss of generality comes from.

14.3 First-Order Natural Deduction

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25157`.

In this section, we will introduce the first-order natural deduction calculus. Recall from **??** that natural deduction calculus have introduction and elimination for every logical constant (the connectives in PL^0). Recall furthermore that we had two styles/notations for the calculus, the classical ND calculus and the sequent-style notation. These principles will be carried over to natural deduction in PL^1.

This allows us to introduce the calculi in two stages, first for the (propositional) connectives and then extend this to a calculus for first-order logic by adding rules for the quantifiers. In particular, we can define the first-order calculi simply by adding (introduction and elimination) rules for the (universal and existential) quantifiers to the calculus $\mathcal{ND}_0$ defined in **??**.

To obtain a first-order calculus, we have to extend $\mathcal{ND}_0$ with (introduction and elimination) rules for the quantifiers.

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

▷ Rules for connectives just as always

▷ **Definition 14.3.1 (New Quantifier Rules).** The first-order natural deduction calculus $\mathcal{ND}^1$ extends $\mathcal{ND}_0$ by the following four rules:

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

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

* means that $\mathbf{A}$ does not depend on any hypothesis in which X is free.

The intuition behind the rule $\mathcal{ND}^1\forall I$ is that a formula $\mathbf{A}$ with a (free) variable X can be generalized to $\forall X.\mathbf{A}$, if X stands for an arbitrary object, i.e. there are no restricting assumptions about X. The $\mathcal{ND}^1\forall E$ rule is just a substitution rule that allows to instantiate arbitrary terms $\mathbf{B}$ for X

in $\mathbf{A}$. The $\mathcal{ND}^1\exists I$ rule says if we have a witness $\mathbf{B}$ for X in $\mathbf{A}$ (i.e. a concrete term $\mathbf{B}$ that makes $\mathbf{A}$ true), then we can existentially close $\mathbf{A}$. The $\mathcal{ND}^1\exists E$ rule corresponds to the common mathematical practice, where we give objects we know exist a new name c and continue the proof by reasoning about this concrete object c. Anything we can prove from the assumption $[c/X](\mathbf{A})$ we can prove outright if $\exists X.\mathbf{A}$ is known.

Now we reformulate the classical formulation of the calculus of natural deduction as a sequent calculus by lifting it to the "judgments level" as we did for propositional logic. We only need provide new quantifier rules.

First-Order Natural Deduction in Sequent Formulation

- Rules for connectives from $\mathcal{ND}^0_\vdash$
- **Definition 14.3.2 (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:

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

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

Natural Deduction with Equality

- **Definition 14.3.3 (First-Order Logic with Equality).** We extend PL^1 with a new logical constant for equality $=\in\Sigma^p{}_2$ and fix its interpretation to $\mathcal{I}(=):=\{(x,x)\,|\,x\in\mathcal{D}_\iota\}$. We call the extended logic first-order logic with equality ($\mathrm{PL}^1_=$)
- We now extend natural deduction as well.
- **Definition 14.3.4.** For the calculus of natural deduction with equality ($\mathcal{ND}^1_=$) we add the following two rules to $\mathcal{ND}^1$ to deal with equality:

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

  where $\mathbf{C}\,[\mathbf{A}]_p$ if the formula $\mathbf{C}$ has a subterm $\mathbf{A}$ at position p and $[\mathbf{B}/p]\mathbf{C}$ is the result of replacing that subterm with $\mathbf{B}$.
- In many ways equivalence behaves like equality, we will use the following rules in $\mathcal{ND}^1$
- **Definition 14.3.5.** $\Leftrightarrow I$ is derivable and $\Leftrightarrow E$ is admissible in $\mathcal{ND}^1$:

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

Again, we have two rules that follow the introduction/elimination pattern of natural deduction calculi.

Definition 14.3.6. We have the canonical sequent rules that correspond to them: $=I$, $=E$, $\Leftrightarrow I$, and $\Leftrightarrow E$

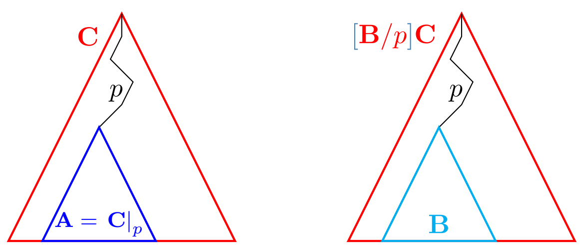
To make sure that we understand the constructions here, let us get back to the "replacement at position" operation used in the equality rules.

Positions in Formulae

- **Idea:** Formulae are (naturally) trees, so we can use tree positions to talk about subformulae
- **Definition 14.3.7.** A position p is a tuple of natural numbers that in each node of an expression (tree) specifies into which child to descend. For an expression $\mathbf{A}$ we denote the subexpression at p with $\mathbf{A}|_p$.

  We will sometimes write an expression $\mathbf{C}$ as $\mathbf{C}[\mathbf{A}]_p$ to indicate that $\mathbf{C}$ the subexpression $\mathbf{A}$ at position p.

  If $\mathbf{C}[\mathbf{A}]_p$ and $\mathbf{A}$ is atomic, then we speak of an occurrence of $\mathbf{A}$ in $\mathbf{C}$.
- **Definition 14.3.8.** Let p be a position, then $[\mathbf{A}/p]\mathbf{C}$ is the expression obtained from $\mathbf{C}$ by replacing the subexpression at p by $\mathbf{A}$.
- **Example 14.3.9 (Schematically).**



The operation of replacing a subformula at position p is quite different from e.g. (first-order) substitutions:

- We are replacing subformulae with subformulae instead of instantiating variables with terms.
- Substitutions replace all occurrences of a variable in a formula, whereas formula replacement only affects the (one) subformula at position p.

We conclude this section with an extended example: the proof of a classical mathematical result in the natural deduction calculus with equality. This shows us that we can derive strong properties about complex situations (here the real numbers; an uncountably infinite set of numbers).

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

- We can do real mathematics with $\mathcal{ND}^1_=$:
- **Theorem 14.3.10.** *$\sqrt{2}$ is irrational*

  Proof: We prove the assertion by contradiction

  1. Assume that $\sqrt{2}$ is rational.
  2. Then there are numbers p and q such that $\sqrt{2} = p/q$.

3. So we know $2q^2 = p^2$.
4. But $2q^2$ has an odd number of prime factors while p^2 an even number.
5. This is a contradiction (since they are equal), so we have proven the assertion

If we want to formalize this into $\mathcal{ND}^1$, we have to write down all the assertions in the proof steps in PL^1 syntax and come up with justifications for them in terms of $\mathcal{ND}^1$ inference rules. The next two slides show such a proof, where we write $/n$ to denote that n is prime, use $\#(n)$ for the number of prime factors of a number n, and write $\mathrm{irr}(r)$ if r is irrational.

$\mathcal{ND}^1_=$ 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	$\mathrm{prime}(2) \Rightarrow \#(2q^2) = (\#(q^2)+1)$	$\mathcal{ND}^1_\vdash\forall E^2(1)$

Lines 6 and 9 are local hypotheses for the proof (they only have an implicit counterpart in the inference rules as defined above). Finally we have abbreviated the arithmetic simplification of line 9 with the justification "arith" to avoid having to formalize elementary arithmetic.

$\mathcal{ND}^1_=$ 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		$\mathrm{irr}(\sqrt{2})$	$\mathcal{ND}_0\neg E^2(23)$

We observe that the $\mathcal{ND}^1$ proof is much more detailed, and needs quite a few Lemmata about $\#$ to go through. Furthermore, we have added a definition of irrationality (and treat definitional equality via the equality rules). Apart from these artefacts of formalization, the two representations of proofs correspond to each other very directly.

14.4 Conclusion

Summary (Predicate Logic)

- First-order logic allows to explicitly speak about objects and their properties. It is thus a more natural and compact representation language than propositional logic; it also enables us to speak about infinite sets of objects.
- Logic has thousands of years of history. A major current application in AI is *semantic technology*. (up soon)
- First-order logic (PL^1) allows universal and existential quantifier quantification over individuals.
- A PL^1 model consists of a universe $\mathcal{D}_\iota$ and a function $\mathcal{I}$ mapping individual constants/predicate constants/function constants to elements/relations/functions on $\mathcal{D}_\iota$.
- First-order natural deduction is a sound and complete calculus for PL^1 intended and optimized for human understanding.

Applications for $\mathcal{ND}^1$ (and extensions)

- **Recap:** We can express mathematical theorems in PL^1 and prove them in $\mathcal{ND}^1$.
- **Problem:** These proofs can be huge (giga-steps), how can we trust them?
- **Definition 14.4.1.** A proof checker for a calculus $\mathcal{C}$ is a program that reads (a formal representation) of a $\mathcal{C}$-proof $\mathcal{P}$ and performs proof-checking, i.e. it checks whether all rule applications in $\mathcal{P}$ are (syntactically) correct.
- **Remark:** Proof-checking goes step-by-step $\rightsquigarrow$ proof checkers run in linear time.
- **Idea:** If the logic can express (safety)-properties of programs, we can use proof checkers for formal program verification. (there are extensions of PL^1 that can)
- **Problem:** These proofs can be humongous, how can humans write them?
- **Idea:** Automate proof construction via
  - lemma/theorem libraries that collect useful intermediate results
  - tactics $\widehat{=}$ subroutines that construct recurring sub-proofs
  - calls to automated theorem prover (ATP) (next chapter)

Proof checkers that do any/all of these are called proof assistants.

▷ **Definition 14.4.2.** Formal methods are logic-based techniques for the specification, development, analysis, and verification of software and hardware.

▷ Formal methods is a major (industrial) application of AI/logic technology.

Suggested Reading:

- *Chapter 8: First-Order Logic*, Sections 8.1 and 8.2 in [RN09]
  - A less formal account of what I cover in "Syntax" and "Semantics". Contains different examples, and complementary explanations. Nice as additional background reading.
- Sections 8.3 and 8.4 provide additional material on using PL1, and on modeling in PL1, that I don't cover in this lecture. Nice reading, not required for exam.
- *Chapter 9: Inference in First-Order Logic*, Section 9.5.1 in [RN09]
  - A very brief (2 pages) description of what I cover in "Normal Forms". Much less formal; I couldn't find where (if at all) RN cover transformation into prenex normal form. Can serve as additional reading, can't replace the lecture.
- **Excursion:** A full analysis of any calculus needs a completeness proof. We will not cover this in AI-1, but provide one for the calculi introduced so far in**??**.

Chapter 15

Automated Theorem Proving in First-Order Logic

In this chapter, we take up the machine-oriented calculi for propositional logic from ?? and extend them to the first-order case. While this has been relatively easy for the natural deduction calculus – we only had to introduce the notion of substitutions for the elimination rule for the universal quantifier we have to work much more here to make the calculi effective for implementation.

15.1 First-Order Inference with Tableaux

15.1.1 First-Order Tableau Calculi

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/25156`.

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 ($\rightsquigarrow$ models).

▷ **Example 15.1.1.** Tableau calculi try to construct models for labeled formulae:

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

▷ **Idea:** Open branches in saturated tableaux yield models.

▷ **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)

- Satisfiable, iff there are open branches (correspond to models)

Tableau calculi develop a formula in a tree-shaped arrangement that represents a case analysis on when a formula can be made true (or false). Therefore the formulae are decorated with upper indices that hold the intended truth value.
On the left we have a refutation tableau that analyzes a negated formula (it is decorated with the intended truth value F). Both branches contain an elementary contradiction ⊥.

On the right we have a model generation tableau, which analyzes a positive formula (it is decorated with the intended truth value T). This tableau uses the same rules as the refutation tableau, but makes a case analysis of when this formula can be satisfied. In this case we have a closed branch and an open one. The latter corresponds a model.

Now that we have seen the examples, we can write down the tableau rules formally.

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 15.1.2.** The propositional tableau calculus $\mathcal{T}_0$ has two inference rules per connective (one for each possible label)

$$\frac{(\mathbf{A}\wedge\mathbf{B})^{\mathsf{T}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}}\\ \mathbf{B}^{\mathsf{T}}\end{array}}\,\mathcal{T}_0\wedge \qquad \frac{(\mathbf{A}\wedge\mathbf{B})^{\mathsf{F}}}{\mathbf{A}^{\mathsf{F}}\ \big|\ \mathbf{B}^{\mathsf{F}}}\,\mathcal{T}_0\vee \qquad \frac{\neg\mathbf{A}^{\mathsf{T}}}{\mathbf{A}^{\mathsf{F}}}\,\mathcal{T}_0\neg^{\mathsf{T}} \qquad \frac{\neg\mathbf{A}^{\mathsf{F}}}{\mathbf{A}^{\mathsf{T}}}\,\mathcal{T}_0\neg^{\mathsf{F}} \qquad \frac{\begin{array}{c}\mathbf{A}^{\alpha}\\ \mathbf{A}^{\beta}\end{array}\quad \alpha\neq\beta}{\bot}\,\mathcal{T}_0\bot$$

  Use rules exhaustively as long as they contribute new material (⇝ termination)
- **Definition 15.1.3.** We call any tree (| introduces branches) produced by the $\mathcal{T}_0$ inference rules from a set Φ of labeled formulae a tableau for Φ.
- **Definition 15.1.4.** Call a tableau saturated, iff no rule adds new material and a branch closed, iff it ends in ⊥, else open. A tableau is closed, iff all of its branches are.

  In analogy to the ⊥ at the end of closed branches, we sometimes decorate open branches with a □ symbol.

These inference rules act on tableaux have to be read as follows: if the formulae over the line appear in a tableau branch, then the branch can be extended by the formulae or branches below the line. There are two rules for each primary connective, and a branch closing rule that adds the special symbol ⊥ (for unsatisfiability) to a branch.
We use the tableau rules with the convention that they are only applied, if they contribute new material to the branch. This ensures termination of the tableau procedure for propositional logic (every rule eliminates one primary connective).
Definition 15.1.5. We will call a closed tableau with the labeled formula $\mathbf{A}^{\alpha}$ at the root a tableau refutation for $\mathcal{A}^{\alpha}$.

The saturated tableau represents a full case analysis of what is necessary to give $\mathbf{A}$ the truth value α; since all branches are closed (contain contradictions) this is impossible.

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

▷ **Definition 15.1.6 ($\mathcal{T}_0$-Theorem/Derivability).** $\mathbf{A}$ is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau with $\mathbf{A}^{\mathsf{F}}$ at the root.

$\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ derives $\mathbf{A}$ in $\mathcal{T}_0$ ($\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau starting with $\mathbf{A}^{\mathsf{F}}$ and Φ^{T}. The tableau with only a branch of $\mathbf{A}^{\mathsf{F}}$ and Φ^{T} is called initial for $\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$.

Definition 15.1.7. We will call a tableau refutation for $\mathbf{A}^{\mathsf{F}}$ a tableau proof for $\mathbf{A}$, since it refutes the possibility of finding a model where $\mathbf{A}$ evaluates to F. Thus $\mathbf{A}$ must evaluate to T in all models, which is just our definition of validity.

Thus the tableau procedure can be used as a calculus for propositional logic. In contrast to the propositional Hilbert calculus it does not prove a theorem $\mathbf{A}$ by deriving it from a set of axioms, but it proves it by refuting its negation. Such calculi are called negative or test calculi. Generally negative calculi have computational advantages over positive ones, since they have a built-in sense of direction.

We have rules for all the necessary connectives (we restrict ourselves to $\wedge$ and $\neg$, since the others can be expressed in terms of these two via the propositional identities above. For instance, we can write $\mathbf{A} \vee \mathbf{B}$ as $\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$, and $\mathbf{A} \Rightarrow \mathbf{B}$ as $\neg\mathbf{A} \vee \mathbf{B}$,...)

We will now extend the propositional tableau techniques to first-order logic. We only have to add two new rules for the universal quantifier (in positive and negative polarity).

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

▷ **Definition 15.1.8.** The standard tableau calculus ($\mathcal{T}_1$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the following quantifier rules:

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

▷ **Problem:** The rule $\mathcal{T}_1\forall$ displays a case of "don't know indeterminism": to find a refutation we have to guess a formula $\mathbf{C}$ from the (usually infinite) set $\mathit{cwff}_\iota(\Sigma_\iota)$.

For proof search, this means that we have to systematically try all, so $\mathcal{T}_1\forall$ is infinitely branching in general.

The rule $\mathcal{T}_1\forall$ operationalizes the intuition that a universally quantified formula is true, iff all of the instances of the scope are. To understand the $\mathcal{T}_1\exists$ rule, we have to keep in mind that $\exists X.\mathbf{A}$ abbreviates $\neg(\forall X.\neg\mathbf{A})$, so that we have to read $(\forall X.\mathbf{A})^{\mathsf{F}}$ existentially — i.e. as $(\exists X.\neg\mathbf{A})^{\mathsf{T}}$, stating that there is an object with property $\neg\mathbf{A}$. In this situation, we can simply give this object a name: c, which we take from our (infinite) set of witness constants Σ_0^{sk}, which we have given ourselves expressly for this purpose when we defined first-order syntax. In other words $([c/X](\neg\mathbf{A}))^{\mathsf{T}} = ([c/X](\mathbf{A}))^{\mathsf{F}}$ holds, and this is just the conclusion of the $\mathcal{T}_1\exists$ rule.

Note that the $\mathcal{T}_1\forall$ rule is computationally extremely inefficient: we have to guess an (i.e. in a search setting to systematically consider all) instance $\mathbf{C} \in \mathit{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$ for X. This makes the rule infinitely branching.

In the next calculus we will try to remedy the computational inefficiency of the $\mathcal{T}_1\forall$ rule. We do this by delaying the choice in the universal rule.

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

▷ **Definition 15.1.9.** The free variable tableau calculus ($\mathcal{T}_1^f$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the quantifier rules:

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

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

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

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

Metavariables: Instead of guessing a concrete instance for the universally quantified variable as in the $\mathcal{T}_1\forall$ rule, $\mathcal{T}_1^f\forall$ instantiates it with a new metavariable Y, which will be instantiated by need in the course of the derivation.
Skolem terms as witnesses: The introduction of metavariables makes is necessary to extend the treatment of witnesses in the existential rule. Intuitively, we cannot simply invent a new name, since the meaning of the body $\mathbf{A}$ may contain metavariables introduced by the $\mathcal{T}_1^f\forall$ rule. As we do not know their values yet, the witness for the existential statement in the antecedent of the $\mathcal{T}_1^f\exists$ rule needs to depend on that. So witness it using a witness term, concretely by applying a Skolem function to the metavariables in $\mathbf{A}$.
Instantiating Metavariables: Finally, the $\mathcal{T}_1^f\bot$ rule completes the treatment of metavariables, it allows to instantiate the whole tableau in a way that the current branch closes. This leaves us with the problem of finding substitutions that make two terms equal.

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

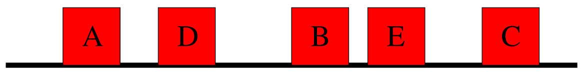
▷ **Definition 15.1.10.** Derivable quantifier rules in $\mathcal{T}_1^f$:

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

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

Tableau Reasons about Blocks

▷ **Example 15.1.11 (Reasoning about Blocks).** Returing to slide 405



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

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

15.1.2 First-Order Unification

Video Nuggets covering this subsection can be found at `https://fau.tv/clip/id/26810` and `https://fau.tv/clip/id/26811`.

We will now look into the problem of finding a substitution σ that make two terms equal (we say it unifies them) in more detail. The presentation of the unification algorithm we give here "transformation-based" this has been a very influential way to treat certain algorithms in theoretical computer science.

A transformation-based view of algorithms: The "transformation-based" view of algorithms divides two concerns in presenting and reasoning about algorithms according to Kowalski's slogan [Kow97]

algorithm $=$ logic $+$ control

The computational paradigm highlighted by this quote is that (many) algorithms can be thought of as manipulating representations of the problem at hand and transforming them into a form that makes it simple to read off solutions. Given this, we can simplify thinking and reasoning about such algorithms by separating out their "logical" part, which deals with is concerned with how the problem representations can be manipulated in principle from the "control" part, which is concerned with questions about when to apply which transformations.

It turns out that many questions about the algorithms can already be answered on the "logic" level, and that the "logical" analysis of the algorithm can already give strong hints as to how to optimize control.

In fact we will only concern ourselves with the "logical" analysis of unification here.

The first step towards a theory of unification is to take a closer look at the problem itself. A first set of examples show that we have multiple solutions to the problem of finding substitutions that make two terms equal. But we also see that these are related in a systematic way.

Unification (Definitions)

▷ **Definition 15.1.12.** For given terms $\mathbf{A}$ and $\mathbf{B}$, unification is the problem of finding a substitution σ, such that $\sigma(\mathbf{A}) = \sigma(\mathbf{B})$.

▷ **Notation:** We write term pairs as $\mathbf{A}=^?\mathbf{B}$ e.g. $f(X)=^?f(g(Y))$.

▷ **Definition 15.1.13.** Solutions (e.g. $[g(a)/X], [a/Y]$, $[g(g(a))/X], [g(a)/Y]$, or $[g(Z)/X], [Z/Y]$) are called unifiers, $\mathbf{U}(\mathbf{A}{=}^?\mathbf{B}) := \{\sigma \mid \sigma(\mathbf{A}) = \sigma(\mathbf{B})\}$.

▷ **Idea:** Find representatives in $\mathbf{U}(\mathbf{A}{=}^?\mathbf{B})$, that generate the set of solutions.

▷ **Definition 15.1.14.** Let σ and θ be substitutions and $W \subseteq \mathcal{V}_\iota$, we say that a substitution σ is more general than θ (on W; write $\sigma{\leq}\theta[W]$), iff there is a substitution ρ, such that $\theta{=}\rho \circ \sigma[W]$, where $\sigma{=}\rho[W]$, iff $\sigma(X) = \rho(X)$ for all $X \in W$.

▷ **Definition 15.1.15.** σ is called a most general unifier (mgu) of $\mathbf{A}$ and $\mathbf{B}$, iff it is minimal in $\mathbf{U}(\mathbf{A}{=}^?\mathbf{B})$ wrt. $\leq[(\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathbf{B}))]$.

FAU Michael Kohlhase: Artificial Intelligence 1 442 2025-02-06

The idea behind a most general unifier is that all other unifiers can be obtained from it by (further) instantiation. In an automated theorem proving setting, this means that using most general unifiers is the least committed choice — any other choice of unifiers (that would be necessary for completeness) can later be obtained by other substitutions.
Note that there is a subtlety in the definition of the ordering on substitutions: we only compare on a subset of the variables. The reason for this is that we have defined substitutions to be total on (the infinite set of) variables for flexibility, but in the applications (see the definition of most general unifiers), we are only interested in a subset of variables: the ones that occur in the initial problem formulation. Intuitively, we do not care what the unifiers do off that set. If we did not have the restriction to the set W of variables, the ordering relation on substitutions would become much too fine-grained to be useful (i.e. to guarantee unique most general unifiers in our case).

Now that we have defined the problem, we can turn to the unification algorithm itself. We will define it in a way that is very similar to logic programming: we first define a calculus that generates “solved forms” (formulae from which we can read off the solution) and reason about control later. In this case we will reason that control does not matter.

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

▷ **Idea:** Unification is equation solving.

▷ **Definition 15.1.16.** We call a formula $\mathbf{A}^1{=}^?\mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n{=}^?\mathbf{B}^n$ an unification problem iff $\mathbf{A}^i, \mathbf{B}^i \in \mathit{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$.

▷ **Note:** We consider unification problems as sets of equations ($\wedge$ is ACI), and equations as two-element multisets ($=^?$ is C).

▷ **Definition 15.1.17.** 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 $\mathcal{E}$.

FAU Michael Kohlhase: Artificial Intelligence 1 443 2025-02-06

In principle, unification problems are sets of equations, which we write as conjunctions, since all of them have to be solved for finding a unifier. Note that it is not a problem for the “logical view” that the representation as conjunctions induces an order, since we know that conjunction is associative, commutative and idempotent, i.e. that conjuncts do not have an intrinsic order or multiplicity, if we consider two equational problems as equal, if they are equivalent as propositional formulae. In the same way, we will abstract from the order in equations, since we know that the equality relation is symmetric. Of course we would have to deal with this somehow in the implementation (typically, we would implement equational problems as lists of pairs), but that belongs into the “control” aspect of the algorithm, which we are abstracting from at the moment.

Solved forms and Most General Unifiers

- **Definition 15.1.18.** We call a pair $\mathbf{A}=^?\mathbf{B}$ solved in a unification problem $\mathcal{E}$, iff $\mathbf{A} = X$, $\mathcal{E} = X=^?\mathbf{A} \wedge \mathcal{E}'$, and $X \notin (\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathcal{E}'))$. We call an unification problem $\mathcal{E}$ a solved form, iff all its pairs are solved.
- **Lemma 15.1.19.** *Solved forms are of the form* $X^1=^?\mathbf{B}^1 \wedge \ldots \wedge X^n=^?\mathbf{B}^n$ *where the* X^i *are distinct and* $X^i \notin \mathrm{free}(\mathbf{B}^j)$.
- **Definition 15.1.20.** Any substitution $\sigma = [\mathbf{B}^1/X^1],\ldots,[\mathbf{B}^n/X^n]$ induces a solved unification problem $\mathcal{E}_\sigma:=(X^1=^?\mathbf{B}^1 \wedge \ldots \wedge X^n=^?\mathbf{B}^n)$.
- **Lemma 15.1.21.** *If* $\mathcal{E} = X^1=^?\mathbf{B}^1 \wedge \ldots \wedge X^n=^?\mathbf{B}^n$ *is a solved form, then* $\mathcal{E}$ *has the unique most general unifier* $\sigma_\mathcal{E}:=[\mathbf{B}^1/X^1],\ldots,[\mathbf{B}^n/X^n]$.
- *Proof:* Let $\theta \in \mathbf{U}(\mathcal{E})$
  1. then $\theta(X^i) = \theta(\mathbf{B}^i) = \theta \circ \sigma_\mathcal{E}(X^i)$
  2. and thus $\theta{=}\theta \circ \sigma_\mathcal{E}[\mathrm{supp}(\sigma)]$.
- **Note:** We can rename the introduced variables in most general unifiers!

It is essential to our "logical" analysis of the unification algorithm that we arrive at unification problems whose unifiers we can read off easily. Solved forms serve that need perfectly as **??** shows.
Given the idea that unification problems can be expressed as formulae, we can express the algorithm in three simple rules that transform unification problems into solved forms (or unsolvable ones).

Unification Algorithm

- **Definition 15.1.22.** The inference system $\mathcal{U}$ consists of the following rules:

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

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

- **Lemma 15.1.23.** $\mathcal{U}$ *is correct:* $\mathcal{E}\vdash_\mathcal{U}\mathcal{F}$ *implies* $\mathbf{U}(\mathcal{F}) \subseteq \mathbf{U}(\mathcal{E})$.
- **Lemma 15.1.24.** $\mathcal{U}$ *is complete:* $\mathcal{E}\vdash_\mathcal{U}\mathcal{F}$ *implies* $\mathbf{U}(\mathcal{E}) \subseteq \mathbf{U}(\mathcal{F})$.
- **Lemma 15.1.25.** $\mathcal{U}$ *is confluent: the order of derivations does not matter.*
- **Corollary 15.1.26.** *First-order unification is unitary: i.e. most general unifiers are unique up to renaming of introduced variables.*
- *Proof sketch:* $\mathcal{U}$ is trivially branching.

The decomposition rule $\mathcal{U}$dec is completely straightforward, but note that it transforms one unification pair into multiple argument pairs; this is the reason, why we have to directly use unification problems with multiple pairs in $\mathcal{U}$.

Note furthermore, that we could have restricted the $\mathcal{U}$triv rule to variable-variable pairs, since for any other pair, we can decompose until only variables are left. Here we observe, that constant-constant pairs can be decomposed with the $\mathcal{U}$dec rule in the somewhat degenerate case without arguments.

Finally, we observe that the first of the two variable conditions in $\mathcal{U}$elim (the "occurs-in-check") makes sure that we only apply the transformation to unifiable unification problems, whereas the second one is a termination condition that prevents the rule to be applied twice.

The notion of completeness and correctness is a bit different than that for calculi that we compare to the entailment relation. We can think of the "logical system of unifiability" with the model class of sets of substitutions, where a set satisfies an equational problem $\mathcal{E}$, iff all of its members are unifiers. This view induces the soundness and completeness notions presented above.

The three meta-properties above are relatively trivial, but somewhat tedious to prove, so we leave the proofs as an exercise to the reader.

We now fortify our intuition about the unification calculus by two examples. Note that we only need to pursue one possible $\mathcal{U}$ derivation since we have confluence.

Unification Examples

▷ **Example 15.1.27.** Two similar unification problems:

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

FAU Michael Kohlhase: Artificial Intelligence 1 446 2025-02-06

We will now convince ourselves that there cannot be any infinite sequences of transformations in $\mathcal{U}$. Termination is an important property for an algorithm.

The proof we present here is very typical for termination proofs. We map unification problems into a partially ordered set $\langle S, \prec\rangle$ where we know that there cannot be any infinitely descending sequences (we think of this as measuring the unification problems). Then we show that all transformations in $\mathcal{U}$ strictly decrease the measure of the unification problems and argue that if there were an infinite transformation in $\mathcal{U}$, then there would be an infinite descending chain in S, which contradicts our choice of $\langle S, \prec\rangle$.

The crucial step in coming up with such proofs is finding the right partially ordered set. Fortunately, there are some tools we can make use of. We know that $\langle \mathbb{N}, <\rangle$ is terminating, and there are some ways of lifting component orderings to complex structures. For instance it is well-known that the lexicographic ordering lifts a terminating ordering to a terminating ordering on finite dimensional Cartesian spaces. We show a similar, but less known construction with multisets for our proof.

Unification (Termination)

- **Definition 15.1.28.** Let S and T be multisets and $\leq$ a partial ordering on $S \cup T$. Then we define $S \prec^m S$, iff $S = C \uplus T'$ and $T = C \uplus \{t\}$, where $s \leq t$ for all $s \in S'$. We call $\leq^m$ the multiset ordering induced by $\leq$.
- **Definition 15.1.29.** We call a variable X solved in an unification problem $\mathcal{E}$, iff $\mathcal{E}$ contains a solved pair $X =^? \mathbf{A}$.
- **Lemma 15.1.30.** *If $\prec$ is linear/terminating On S, then $\prec^m$ is linear/terminating on $\mathcal{P}(S)$.*
- **Lemma 15.1.31.** *$\mathcal{U}$ is terminating.* *(any $\mathcal{U}$-derivation is finite)*
- *Proof:* We prove termination by mapping $\mathcal{U}$ transformation into a Noetherian space.
  1. Let $\mu(\mathcal{E}) := \langle n, \mathcal{N} \rangle$, where
     - n is the number of unsolved variables in $\mathcal{E}$
     - $\mathcal{N}$ is the multiset of term depths in $\mathcal{E}$
  2. The lexicographic order $\prec$ on pairs $\mu(\mathcal{E})$ is decreased by all inference rules.
     2.1. $\mathcal{U}\mathrm{dec}$ and $\mathcal{U}\mathrm{triv}$ decrease the multiset of term depths without increasing the unsolved variables.
     2.2. $\mathcal{U}\mathrm{elim}$ decreases the number of unsolved variables (by one), but may increase term depths.

But it is very simple to create terminating calculi, e.g. by having no inference rules. So there is one more step to go to turn the termination result into a decidability result: we must make sure that we have enough inference rules so that any unification problem is transformed into solved form if it is unifiable.

First-Order Unification is Decidable

- **Definition 15.1.32.** We call an equational problem $\mathcal{E}$ $\mathcal{U}$-reducible, iff there is a $\mathcal{U}$-step $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ from $\mathcal{E}$.
- **Lemma 15.1.33.** *If $\mathcal{E}$ is unifiable but not solved, then it is $\mathcal{U}$-reducible.*
- *Proof:* We assume that $\mathcal{E}$ is unifiable but unsolved and show the $\mathcal{U}$ rule that applies.
  1. There is an unsolved pair $\mathbf{A} =^? \mathbf{B}$ in $\mathcal{E} = \mathcal{E} \wedge \mathbf{A} =^? \mathbf{B}'$.
  we have two cases
  2. $\mathbf{A}, \mathbf{B} \notin \mathcal{V}_\iota$
     2.1. then $\mathbf{A} = f(\mathrm{A}^1 \ldots \mathrm{A}^n)$ and $\mathbf{B} = f(\mathrm{B}^1 \ldots \mathrm{B}^n)$, and thus $\mathcal{U}\mathrm{dec}$ is applicable
  3. $\mathbf{A} = X \in \mathrm{free}(\mathcal{E})$
     3.1. then $\mathcal{U}\mathrm{elim}$ (if $\mathbf{B} \neq X$) or $\mathcal{U}\mathrm{triv}$ (if $\mathbf{B} = X$) is applicable.
- **Corollary 15.1.34.** *First-order unification is decidable in PL^1.*

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

15.1.3 Efficient Unification

Now that we have seen the basic ingredients of an unification algorithm, let us as always consider complexity and efficiency issues.
We start with a look at the complexity of unification and – somewhat surprisingly – find exponential time/space complexity based simply on the fact that the results – the unifiers – can be exponentially large.

Complexity of Unification

- **Observation:** Naive implementations of unification are exponential in time and space.
- **Example 15.1.35.** Consider the terms

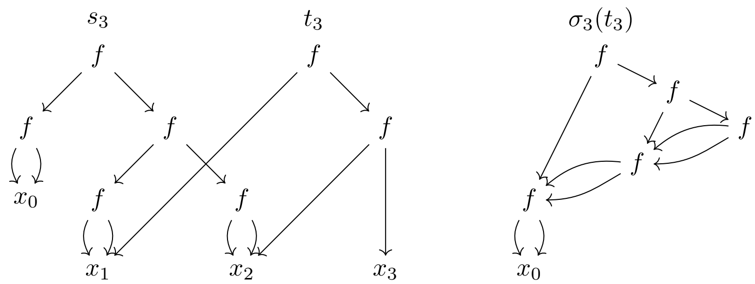
$$s_n = f(f(x_0,x_0),f(f(x_1,x_1),f(\ldots,f(x_{n-1},x_{n-1}))\ldots))$$
$$t_n = f(x_1,f(x_2,f(x_3,f(\cdots,x_n)\cdots)))$$

  - The most general unifier of s_n and t_n is
    $\sigma_n := [f(x_0,x_0)/x_1],[f(f(x_0,x_0),f(x_0,x_0))/x_2],[f(f(f(x_0,x_0),f(x_0,x_0)),f(f(x_0,x_0),f(x_0,x_0)))/x_3],\ldots$
  - 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.

Indeed, the only way to escape this combinatorial explosion is to find representations of substitutions that are more space efficient.

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 15.1.36.** *Terms can be transformed into DAGs in linear time.*
- **Example 15.1.37.** Continuing from ?? … s_3, t_3, and $\sigma_3(s_3)$ as DAGs:



In general: s_n, t_n, and $\sigma_n(s_n)$ only need space in $\mathcal{O}(n)$. (just count)

If we look at the unification algorithm from **??** and the considerations in the termination proof (**??**) with a particular focus on the role of copying, we easily find the culprit for the exponential blowup: $\mathcal{U}$elim, which applies solved pairs as substitutions.

DAG Unification Algorithm

- **Observation:** In $\mathcal{U}$, the $\mathcal{U}$elim rule applies solved pairs $\rightsquigarrow$ subterm duplication.
- **Idea:** Replace $\mathcal{U}$elim the notion of solved forms by something better.
- **Definition 15.1.38.** We say that $X^1 =^? \mathbf{B}^1 \wedge \ldots \wedge X^n =^? \mathbf{B}^n$ is a DAG solved form, iff the X^i are distinct and $X^i \notin \mathrm{free}(\mathbf{B}^j)$ for $i \leq j$.
- **Definition 15.1.39.** The inference system $\mathcal{DU}$ contains rules $\mathcal{U}$dec and $\mathcal{U}$triv from $\mathcal{U}$ plus the following:

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

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

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

We will now turn the ideas we have developed in the last couple of slides into a usable functional algorithm. The starting point is treating terms as DAGs. Then we try to conduct the transformation into solved form without adding new nodes.

Unification by DAG-chase

- **Idea:** Extend the Input-DAGs by edges that represent unifiers.
- **Definition 15.1.40.** Write $n.a$, if a is the symbol of node n.
- (standard) auxiliary procedures: (all constant or linear time in DAGs)
  - find(n) follows the path from n and returns the end node.

- union(n, m) adds an edge between n and m.
- occur(n, m) determines whether $n.x$ occurs in the DAG with root m.

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 15.1.41.** dag—unify *uses linear space, since no new nodes are created, and at most one link per variable.*

▷ **Problem:** dag—unify still uses exponential time.

▷ **Example 15.1.42.** Consider terms $f(s_n, f(t'_n, x_n)), f(t_n, f(s'_n, y_n)))$, where $s'_n = [y_i/x_i](s_n)$ und $t'_n = [y_i/x_i](t_n)$.

dag—unify needs exponentially many recursive calls to unify the nodes x_n and y_n. (they are unified after n calls, but checking needs the time)

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

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/26797`.

We now come to some issues (and clarifications) pertaining to implementing proof search for free variable tableaux. They all have to do with the – often overlooked – fact that $\mathcal{T}_1^f\bot$ instantiates the whole tableau.

The first question one may ask for implementation is whether we expect a terminating proof search; after all, $\mathcal{T}_0$ terminated. We will see that the situation for $\mathcal{T}_1^f$ is different.

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 15.1.43.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 15.1.44.** A tableau proof for $(p(a) \lor p(b)) \Rightarrow (\exists.p())$.

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

  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 $\mathcal{T}_1^f\forall$.
- **Definition 15.1.45.** Let $\mathcal{T}$ be a tableau for $\mathbf{A}$, and a positive occurrence of $\forall x.\mathbf{B}$ in $\mathbf{A}$, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.\mathbf{B}$ its multiplicity.
- **Observation 15.1.46.** *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 15.1.47.** *$\mathcal{T}_1^f$ is only complete with unbounded multiplicities.*
- *Proof sketch:* Replace $p(a) \lor p(b)$ with $p(a_1) \lor \ldots \lor p(a_n)$ in ??.
- **Remark:** Otherwise validity in PL^1 would be decidable.
- **Implementation:** We need an iterative multiplicity deepening process.

The other thing we need to realize is that there may be multiple ways we can use $\mathcal{T}_1^f\bot$ to close a branch in a tableau, and – as $\mathcal{T}_1^f\bot$ instantiates the whole tableau and not just the branch itself –

this choice matters.

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

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

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)

The method of spanning matings follows the intuition that if we do not have good information on how to decide for a pair of opposite literals on a branch to use in $\mathcal{T}_1^f\bot$, we delay the choice by initially disregarding the rule altogether during saturation and then – in a later phase– looking for a configuration of cuts that have a joint overall unifier. The big advantage of this is that we only need to know that one exists, we do not need to compute or apply it, which would lead to exponential blow-up as we have seen above.

Spanning Matings for $\mathcal{T}_1^f\bot$

- **Observation 15.1.49.** *$\mathcal{T}_1^f$ 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 15.1.50.**

  Let $\mathcal{T}$ be a $\mathcal{T}_1^f$ tableau, then we call a unification problem $\mathcal{E} := \mathbf{A}_1 =^? \mathbf{B}_1 \wedge \ldots \wedge \mathbf{A}_n =^? \mathbf{B}_n$ a mating for $\mathcal{T}$, iff $\mathbf{A}_i^{\mathsf{T}}$ and $\mathbf{B}_i^{\mathsf{F}}$ 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 $\mathbf{A}_i^{\mathsf{T}}$ and $\mathbf{B}_i^{\mathsf{F}}$ for some i.
- **Theorem 15.1.51.** *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.

Excursion: Now that we understand basic unification theory, we can come to the meta-theoretical properties of the tableau calculus. We delegate this discussion to**??**.

15.2 First-Order Resolution

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26817`.

First-Order Resolution (and CNF)

▷ **Definition 15.2.1.** The first-order CNF calculus CNF_1 is given by the inference rules of CNF_0 extended by the following quantifier rules:

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

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

the first-order CNF $\mathit{CNF}_1(\Phi)$ of Φ is the set of all clauses that can be derived from Φ.

▷ **Definition 15.2.2 (First-Order Resolution Calculus).** The First-order resolution calculus ($\mathcal{R}_1$) is a test calculus that manipulates formulae in conjunctive normal form. $\mathcal{R}_1$ has two inference rules:

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

First-Order CNF – Derived Rules

▷ **Definition 15.2.3.** The following inference rules are derivable from the ones above via $(\exists X.\mathbf{A}) = \neg(\forall X.\neg\mathbf{A})$:

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

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

Excursion: Again, we relegate the meta-theoretical properties of the first-order resolution calculus to??.

15.2.1 Resolution Examples

Col. West, *a Criminal?*

▷ **Example 15.2.4.** 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: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$

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

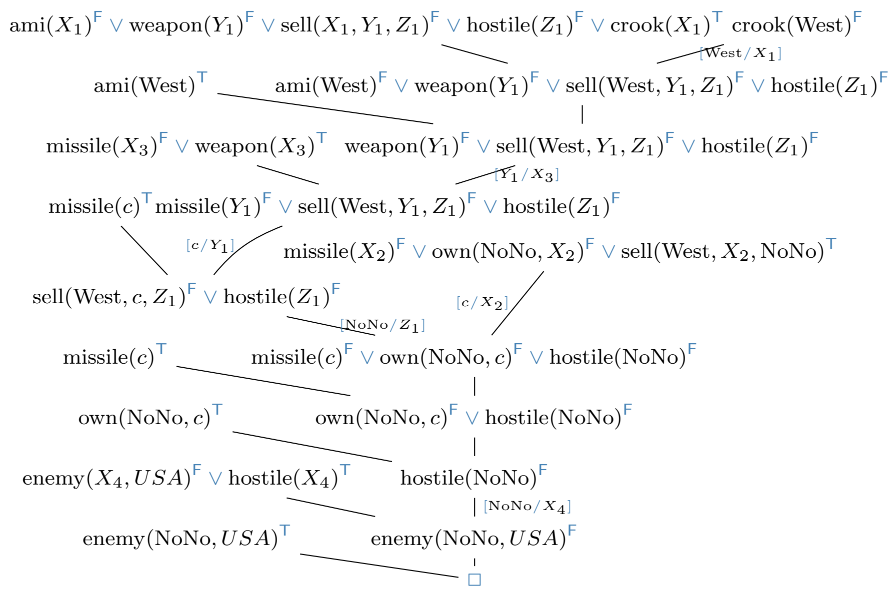
▷ *Missiles are weapons*:
Clause: $\mathrm{mle}(X_3) \Rightarrow \mathrm{weap}(X_3)$

▷ *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



Curiosity Killed the Cat?

▷ **Example 15.2.5.** 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: $\mathrm{animal}(Y_3)^{\mathsf{F}} \vee \mathrm{kill}(X_3,Y_3)^{\mathsf{F}} \vee \mathrm{love}(Z_3,X_3)^{\mathsf{F}}$

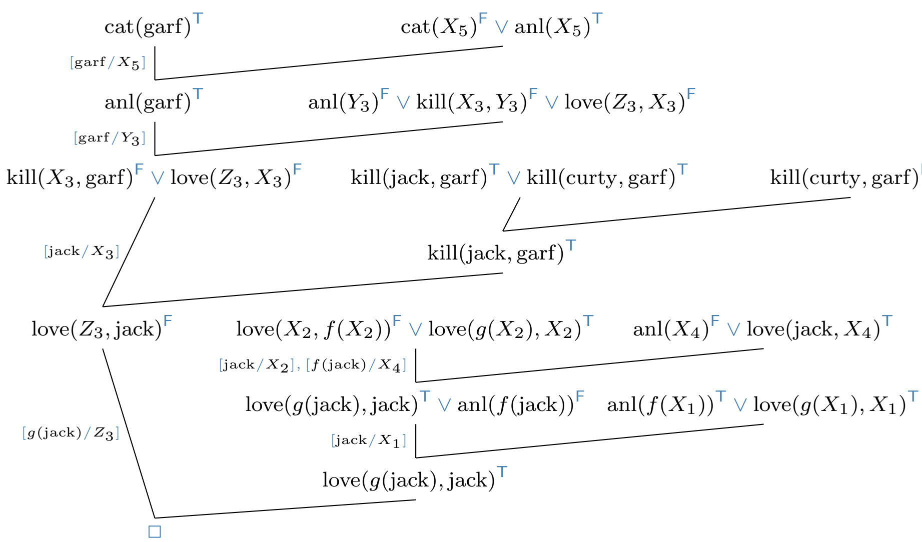
▷ *Jack loves all animals*:

Clause: $\text{animal}(X_4)^{\mathsf{F}} \vee \text{love}(\text{jack}, X_4)^{\mathsf{T}}$

▷ *Cats are animals*:
Clause: $\text{cat}(X_5)^{\mathsf{F}} \vee \text{animal}(X_5)^{\mathsf{T}}$

▷ *Either Jack or curiosity killed the cat (whose name is "Garfield")*:
Clauses: $\text{kill}(\text{jack}, \text{garf})^{\mathsf{T}} \vee \text{kill}(\text{curiosity}, \text{garf})^{\mathsf{T}}$ and $\text{cat}(\text{garf})^{\mathsf{T}}$

Curiosity Killed the Cat! PL1 Resolution Proof



Excursion: A full analysis of any calculus needs a completeness proof. We will not cover this in the course, but provide one for the calculi introduced so far in**??**.

15.3 Logic Programming as Resolution Theorem Proving

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26820`. To understand Prolog better, we can interpret the language of Prolog as resolution in PL^1.

We know all this already

▷ Goals, goal sets, rules, and facts are just clauses. (called Horn clauses)

▷ **Observation 15.3.1 (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 15.3.2 (Goal set).** *?– G_1,. . .,G_n. corresponds to $G_1{}^{\mathsf{F}} \vee \ldots \vee G_n{}^{\mathsf{F}}$*

▷ **Observation 15.3.3 (Fact).** *F. corresponds to the unit clause F^{T}.*

▷ **Definition 15.3.4.** 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

$$\frac{P^{\mathsf{T}} \vee \mathbf{A} \quad P^{\mathsf{F}} \vee \mathbf{B}}{\mathbf{A} \vee \mathbf{B}}$$

positive, unit-resulting hyperresolution (PURR)

This observation helps us understand Prolog better, and use implementation techniques from automated theorem proving.

PROLOG (Horn Logic)

▷ **Definition 15.3.5.** 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 15.3.6.** 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 15.3.7.** 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 15.3.8 (Query).** ?– fallible(X),greek(X).

▷ **Answer substitution:** $[\text{sokrates}/X]$

To gain an intuition for this quite abstract definition let us consider a concrete knowledge base about cars. Instead of writing down everything we know about cars, we only write down that cars are motor vehicles with four wheels and that a particular object c has a motor and four wheels. We can see that the fact that c is a car can be derived from this. Given our definition of a knowledge base as the deductive closure of the facts and rule explicitly written down, the assertion that c is a car is in the induced knowledge base, which is what we are after.

Knowledge Base (Example)

- **Example 15.3.9.** 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).
```

$$\dfrac{\dfrac{m(c) \quad w(c,4)}{m(c) \wedge w(c,4)} \wedge I \qquad \dfrac{m(x) \wedge w(x,4) \Rightarrow car()}{m(c) \wedge w(c,4) \Rightarrow car()} \text{Subst}}{car(c)} \text{MP}$$

In this very simple example $car(c)$ is about the only fact we can derive, but in general, knowledge bases can be infinite (we will see examples below).

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 15.3.10.** Deduction $\widehat{=}$ knowledge extension

▷ **Example 15.3.11.** $\dfrac{rains \Rightarrow wet\_street \;\; rains}{wet\_street} D$

▷ **Definition 15.3.12.** Abduction $\widehat{=}$ explanation

▷ **Example 15.3.13.** $\dfrac{rains \Rightarrow wet\_street \;\; wet\_street}{rains} A$

▷ **Definition 15.3.14.** Induction $\widehat{=}$ learning general rules from examples

▷ **Example 15.3.15.** $\dfrac{wet\_street \;\; rains}{rains \Rightarrow wet\_street} I$

15.4 Summary: ATP in First-Order Logic

Summary: ATP in First-Order Logic

▷ The propositional calculi for ATP can be extended to first-order logic by adding quantifier rules.

- ▷ The rule for the universal quantifier can be made efficient by introducing metavariables that postpone the decision for instances.
- ▷ We have to extend the witness constants in the rules for existential quantifiers to Skolem functions.
- ▷ The cut rules can used to instantiate the metavariables by unification.

These ideas are enough to build a tableau calculus for first-order logic.

▷ Unification is an efficient decision procdure for finding substitutions that make first-order terms (syntactically) equal.

▷ In prenex normal form, all quantifiers are up front. In Skolem normal form, additionally there are no existential quantifiers. In claus normal form, additionally the formula is in CNF.

▷ Any PL^1 formula can efficiently be brought into a satisfiability-equivalent clause normal form.

▷ This allows first-order resolution.

Chapter 16

Knowledge Representation and the Semantic Web

The field of "Knowledge Representation" is usually taken to be an area in Artificial Intelligence that studies the representation of knowledge in formal systems and how to leverage inference techniques to generate new knowledge items from existing ones. Note that this definition coincides with with what we know as logical systems in this course. This is the view taken by the subfield of "description logics", but restricted to the case, where the logical systems have an entailment relation to ensure applicability. This chapter is organized as follows. We will first give a general introduction to the concepts of knowledge representation using semantic networks – an early and very intuitive approach to knowledge representation – as an object-to-think-with. In ?? we introduce the principles and services of logic-based knowledge-representation using a non-standard interpretation of propositional logic as the basis, this gives us a formal account of the taxonomic part of semantic networks. In ?? we introduce the logic $\mathcal{ALC}$ that adds relations (called "roles") and restricted quantification and thus gives us the full expressive power of semantic networks. Thus $\mathcal{ALC}$ can be seen as a prototype description logic. In ?? we show how description logics are applied as the basis of the "semantic web".

16.1 Introduction to Knowledge Representation

A Video Nugget covering the introduction to knowledge representation can be found at `https://fau.tv/clip/id/27279`.

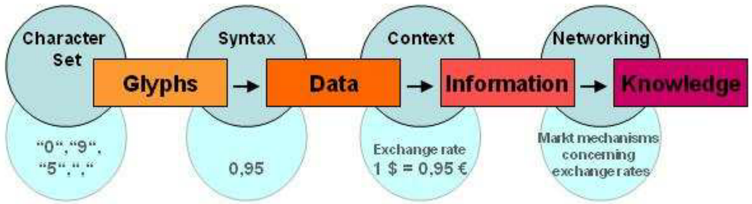
Before we start into the development of description logics, we set the stage by looking into alternatives for knowledge representation.

16.1.1 Knowledge & Representation

To approach the question of knowledge representation, we first have to ask ourselves, what knowledge might be. This is a difficult question that has kept philosophers occupied for millennia. We will not answer this question in this course, but only allude to and discuss some aspects that are relevant to our cause of 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

FAU Michael Kohlhase: Artificial Intelligence 1 473 2025-02-06

According to an influential view of [PRR97], knowledge appears in layers. Staring with a character set that defines a set of glyphs, we can add syntax that turns mere strings into data. Adding context information gives information, and finally, by relating the information to other information allows to draw conclusions, turning information into knowledge.

Note that we already have aspects of representation and function in the diagram at the top of the slide. In this, the additional functionaltiy added in the successive layers gives the representations more and more functions, until we reach the knowledge level, where the function is given by inferencing. In the second example, we can see that representations determine possible functions.

Let us now strengthen our intuition about knowledge by contrasting knowledge representations from "regular" data structures in computation.

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)

FAU Michael Kohlhase: Artificial Intelligence 1 474 2025-02-06

As knowledge is such a central notion in artificial intelligence, it is not surprising that there are multiple approaches to dealing with it. We will only deal with the first one and leave the others to self-study.

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.

When assessing the relative strengths of the respective approaches, we should evaluate them with respect to a pre-determined set of criteria.

KR Approaches/Evaluation Criteria

- **Definition 16.1.1.** 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

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27280`.

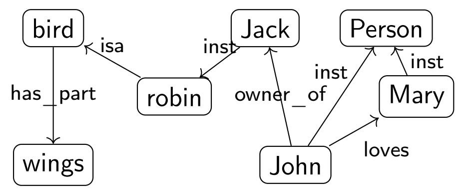
To get a feeling for early knowledge representation approaches from which description logics developed, we take a look at "semantic networks" and contrast them to logical approaches. Semantic networks are a very simple way of arranging knowledge about objects and concepts and their relationships in a graph.

Semantic Networks [CQ69]

- **Definition 16.1.2.** 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 16.1.3.** A semantic network for birds and persons:



▷ **Problem:** How do we derive new information from such a network?

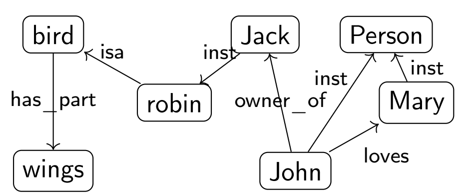
▷ **Idea:** Encode taxonomic information about objects and concepts in special links (“isa” and “inst”) and specify property inheritance along them in the process model.

Even though the network in **??** is very intuitive (we immediately understand the concepts depicted), it is unclear how we (and more importantly a machine that does not associate meaning with the labels of the nodes and edges) can draw inferences from the “knowledge” represented.

Deriving Knowledge Implicit in Semantic Networks

▷ **Observation 16.1.4.** *There is more knowledge in a semantic network than is explicitly written down.*

▷ **Example 16.1.5.** 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 16.1.6.** We call links labeled by

- “isa” an inclusion or isa link (inclusion of concepts)
- “inst” instance or inst link (concept membership)

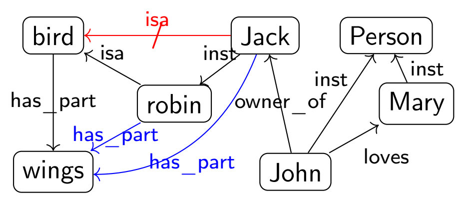
We now make the idea of “propagating properties” rigorous by defining the notion of derived relations, i.e. the relations that are left implicit in the network, but can be added without changing its meaning.

Deriving Knowledge Semantic Networks

- **Definition 16.1.7 (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 16.1.8.** Derived relations in ??



- **Slogan:** Get out more knowledge from a semantic networks than you put in.

Note that ?? does not quite allow to derive that *Jack is a bird* (did you spot that "isa" is not a relation that can be inferred?), even though we know it is true in the world. This shows us that inference in semantic networks has be to very carefully defined and may not be "complete", i.e. there are things that are true in the real world that our inference procedure does not capture.

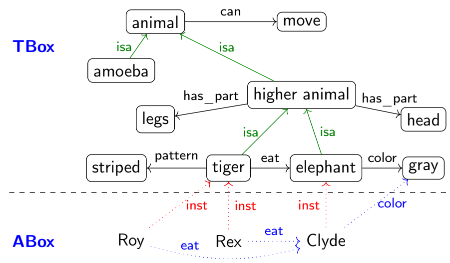
Dually, if we are not careful, then the inference procedure might derive properties that are not true in the real world even if all the properties explicitly put into the network are. We call such an inference procedure unsound or incorrect.

These are two general phenomena we have to keep an eye on.

Another problem is that semantic networks (e.g. in ??) confuse two kinds of concepts: individuals (represented by proper names like *John* and *Jack*) and concepts (nouns like *robin* and *bird*). Even though the isa and inst link already acknowledge this distinction, the "has_part" and "loves" relations are at different levels entirely, but not distinguished in the networks.

Terminologies and Assertions

- *Remark 16.1.9.* We should distinguish concepts from objects.
- **Definition 16.1.10.** We call the subgraph of a semantic network N spanned by the isa links and relations between concepts the terminology (or TBox, or the famous Isa Hierarchy) and the subgraph spanned by the inst links and relations between objects, the assertions (together the ABox) of N.
- **Example 16.1.11.** 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*).

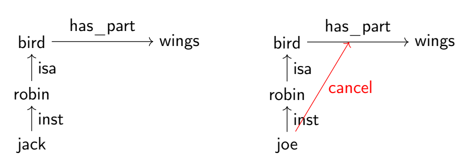
FAU Michael Kohlhase: Artificial Intelligence 1 480 2025-02-06

But there are severe shortcomings of semantic networks: the suggestive shape and node names give (humans) a false sense of meaning, and the inference rules are only given in the process model (the implementation of the semantic network processing system).

This makes it very difficult to assess the strength of the inference system and make assertions e.g. about completeness.

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 16.1.12.** Consider a robin that has lost its wings in an accident:



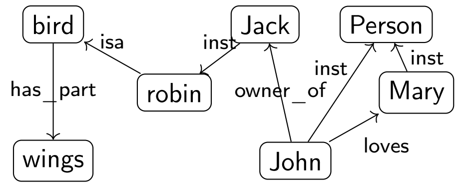
"Cancel-links" have been proposed, but their status and process model are debatable.

FAU Michael Kohlhase: Artificial Intelligence 1 481 2025-02-06

To alleviate the perceived drawbacks of semantic networks, we can contemplate another notation that is more linear and thus more easily implemented: function/argument notation.

Another Notation for Semantic Networks

- **Definition 16.1.13.** Function/argument notation for semantic networks
  - interprets nodes as arguments (reification to individuals)
  - interprets links as functions (predicates actually)
- **Example 16.1.14.**



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)

FAU Michael Kohlhase: Artificial Intelligence 1 482 2025-02-06

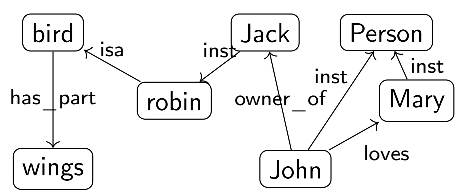
Indeed the function/argument notation is the immediate idea how one would naturally represent semantic networks for implementation.

This notation has been also characterized as subject/predicate/object triples, alluding to simple (English) sentences. This will play a role in the "semantic web" later.

Building on the function/argument notation from above, we can now give a formal semantics for semantic network: we translate them into first-order logic and use the semantics of that.

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)

FAU Michael Kohlhase: Artificial Intelligence 1 483 2025-02-06

Indeed, the semantics induced by the translation to first-order logic, gives the intuitive meaning to the semantic networks. Note that this only holds only for the features of semantic networks that are representable in this way, e.g. the "cancel links" shown above are not (and that is a feature, not a bug).

But even more importantly, the translation to first-order logic gives a first process model: we can use first-order inference to compute the set of inferences that can be drawn from a semantic network.

Before we go on, let us have a look at an important application of knowledge representation technologies: the semantic web.

16.1.3 The Semantic Web

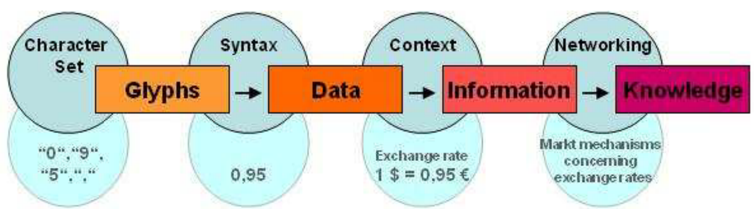
A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27281`. We will now define the term semantic web and discuss the pertinent ideas involved. There are two central ones, we will cover here:

- Information and data come in different levels of explicitness; this is usually visualized by a "ladder" of information.
- if information is sufficiently machine-understandable, then we can automate drawing conclusions.

The Semantic Web

▷ **Definition 16.1.15.** 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 16.1.16.** Information not explicitly represented (in one place)

Query: *Who was US president when Barak Obama was born?*

Google: *. . . BIRTH DATE: August 04, 1961. . .*

Query: *Who was US president in 1961?*

Google: *President: Dwight D. Eisenhower [. . .] John F. Kennedy (starting Jan. 20.)*

Humans understand the text and combine the information to get the answer. Machines need more than just text ⇝ semantic web technology.

The term "semantic web" was coined by Tim Berners Lee in analogy to semantic networks, only applied to the world wide web. And as for semantic networks, where we have inference processes that allow us the recover information that is not explicitly represented from the network (here the world-wide-web).

To see that problems have to be solved, to arrive at the semantic web, we will now look at a concrete example about the "semantics" in web pages. Here is one that looks typical enough.

What is the Information a User sees?

▷ **Example 16.1.17.** 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.

But as for semantic networks, what you as a human can see ("understand" really) is deceptive, so let us obfuscate the document to confuse your "semantic processor". This gives an impression of what the computer "sees".

What the machine sees

▷ **Example 16.1.18.** 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∇⟩⌈⇔⊔⟨]\]§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇\]⊔↙

Obviously, there is not much the computer understands, and as a consequence, there is not a lot the computer can support the reader with. So we have to "help" the computer by providing some meaning. Conventional wisdom is that we add some semantic/functional markup. Here we pick XML without loss of generality, and characterize some fragments of text e.g. as dates.

Solution: XML markup with "meaningful" Tags

▷ **Example 16.1.19.** 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]}⟩ʃ⊔]∇]⌈√⊣∇⊔⟩|⟩√⊣\⊔ʃ]≀↕⟩\}{∇↕
𝒜⊓ʃ⊔∇⊣↕⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩↕]𝒟]\↕⊣∇∥⇔ℱ∇⊣\|]⇔𝒢]∇↕⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔
ℐ∇]↕⊣\⌈⇔ℐ⊔⊣↕†⇔𝒥⊣√⊣\⇔ℳ⊣↕⊔⊣⇔𝒩]⊒𝒵]⊣↕⊣\⌈⇔𝒯⟨]𝒩]⊔⟨]∇↕⊣\⌈ʃ⇔𝒩≀∇⊒⊣†⇔
𝒮⟩\}⊣√≀∇]⇔𝒮⊒⟩⊔‡]∇↕⊣\⌈⇔⊔⟨]𝒰\⟩⊔]⌈𝒦⟩\}⌈≀↕⇔⊔⟨]𝒰\⟩⊔]⌈𝒮⊔⊣⊔]ʃ⇔𝒱]⊔\⊣↕⇔𝒵⊣⟩∇]
</participants>
<introduction>𝒪\⊔⟨]⊔⟨ℳ⊣†ℋ≀\↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈]⊔⟨]⌊⊣]∥⌈∇≀√≀{⊔⟨]]↕⊑]\⊔⟨ℐ\⊔]∇↖
\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\|]↙</introduction>
<program>𝒮√]⊣∥]∇ʃ]≀\{⟩∇↕]⌈
<speaker>𝒯⟩↕ℬ]∇\]∇ʃ↖ℒ]]¬𝒯⟩↕⟩ʃ⊔⟨]⊒]↕↕∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]𝒲]⌊</speaker>
<speaker>ℐ⊣\ℱ≀ʃ⊔]∇¬ℐ⊣\⟩ʃ⊔⟨]√≀\]]∇≀{⊔⟨]𝒢∇⟩⌈⇔⊔⟨]\§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇\]⊔<speaker>
</program>
```


But does this really help? Is conventional wisdom correct?

What can we do with this?

▷ **Example 16.1.20.** Consider the following fragments:

```
ℜ⊔⟩⊔↕]⊤𝒲𝒲𝒲∈′′∈
𝒯⟨]]↕⊑]\⊔⟨ℐ\⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\|]ℜ∝⊔⟩⊔↕]⊤
ℜ√↕⊣|]⊤𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕ℋ≀\↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜ℜ∝√↕⊣|]⊤
ℜ⌈⊣⊔]⊤↖∞∞ℳ⊣†∈′′∈ℜ∝⌈⊣⊔]⊤
```

Given the markup above, a machine agent can

- ▷ parse ∞∞ℳ⊣†∈′′∈ as the date May 7 11 2002 and add this to the user's calendar,
- ▷ parse 𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]↕ℋ≀\↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜 as a destination and find flights.

▷ **But:** do not be deceived by your ability to understand English!

To understand what a machine can understand we have to obfuscate the markup as well, since it does not carry any intrinsic meaning to the machine either.

What the machine sees of the XML

▷ **Example 16.1.21.** Here is what the machine sees of the XML

```
<title>𝒲𝒲𝒲∈′′∈
𝒯⟨]]↕⊑]\⊔⟨ℐ\⊔]∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\|]</⊔⟩⊔↕]>
```

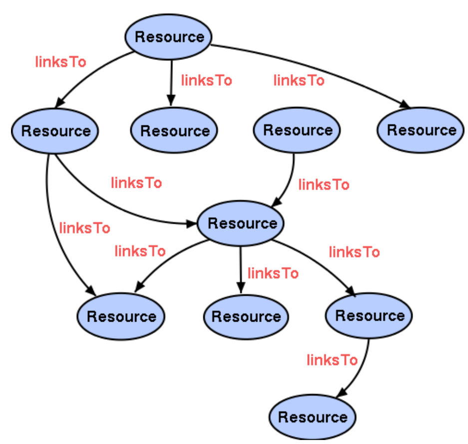
<√‡⊣∣]>𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩ℋ≀⊔]‡ℋ≀\≀‡⊓‡⊓⇔ℋ⊣⊒⊣⟩⇔𝒰𝒮𝒜</√‡⊣∣]>
<⌈⊣⊔]>↖∞∞ℳ⊣†∈′′∈</⌈⊣⊔]>
<√⊣∇⊔⟩⌋⟩√⊣\⊔∫>ℛ]}⟩∫⊔]∇]⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫]≀‡⟩\}{∇≀‡
𝒜⊓∫⊔∇⊣‡⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩‡]𝒟]\‡⊣∇∥⇔ℱ∇⊣\⌋]⇔𝒢]∇‡⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔
ℐ∇]‡⊣\⌈⇔ℐ⊔⊣‡†⇔𝒥⊣√⊣\⇔ℳ⊣‡⊔⊣⇔𝒩]⊒𝒵]⊣‡⊣\⌈⇔𝒯⟨]𝒩]⊔⟨]∇‡⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔
𝒮⟩\}⊣√≀∇]⇔𝒮⊒⟩⊔‡]∇‡⊣\⌈⇔⊔⟨]𝒰\⟩⊔]⌈𝒦⟩\}⌈≀‡⇔⊔⟨]𝒰\⟩⊔]⌈𝒮⊔⊣⊔]∫⇔𝒱⟩]⊔\⊣‡⇔𝒵⊣⟩∇]
</√⊣∇⊔⟩⌋⟩√⊣\⊔∫>
<⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>𝒪\⊔⟨]⊔⟨ℳ⊣†ℋ≀\≀‡⊓‡⊓⊒⟩‡‡√∇≀⊑⟩⌈]⊔⟨]⌊⊣∥]∥⌈∇≀√≀{⊔⟨]‡]⊑]\⊔⟨ℐ\⊔]∇\⊣↖
⊔⟩≀\⊣‡𝒲≀∇‡⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{∇]\⌋]↙</⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>
<√∇≀}∇⊣‡>𝒮√]⊣∥]∇∫⌋≀\{⟩∇‡]⌈
<∫√]⊣∥]∇>𝒯⟩‡ℬ]∇\]∇∫↖ℒ]]¬𝒯⟩‡⟩∫⊔⟨]⊒]‡‡∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]𝒲]⌊</∫√]⊣∥]∇>
<∫√]⊣∥]∇>ℐ⊣\ℱ≀∫⊔]∇¬ℐ⊣\⟩∫⊔⟨]√≀\]]∇≀{⊔⟨]𝒢∇⟩⌈⇔⊔⟨]\§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇\]⊔<∫√]⊣∥]∇>
</√∇≀}∇⊣‡>

So we have not really gained much either with the markup, we really have to give meaning to the markup as well, this is where techniques from semenatic web come into play.

To understand how we can make the web more semantic, let us first take stock of the current status of (markup on) the web. It is well-known that world-wide-web is a hypertext, where multimedia documents (text, images, videos, etc. and their fragments) are connected by hyperlinks. As we have seen, all of these are largely opaque (non-understandable), so we end up with the following situation (from the viewpoint of a machine).

The Current Web

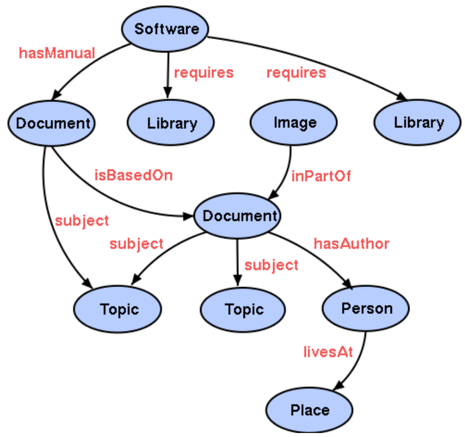
- **Resources:** identified by URIs, untyped
- **Links:** href, src, . . . limited, non-descriptive
- **User:** Exciting world - semantics of the resource, however, gleaned from content
- **Machine:** Very little information available - significance of the links only evident from the context around the anchor.



Let us now contrast this with the envisioned semantic web.

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.



Essentially, to make the web more machine-processable, we need to classify the resources by the concepts they represent and give the links a meaning in a way, that we can do inference with that. The ideas presented here gave rise to a set of technologies jointly called the "semantic web", which we will now summarize before we return to our logical investigations of knowledge representation techniques.

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+13b] 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

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27282`.
Now that we know what semantic networks mean, let us look at a couple of other approaches that were influential for the development of knowledge representation. We will just mention them for reference here, but not cover them in any depth.

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 16.1.22.** Frames (group everything around the object)

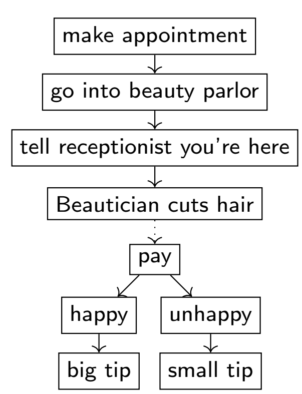
```
(catch_object catch_22
              (catcher jack_2)
              (caught ball_5))
```

  + 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 16.1.23.** 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 16.1.24.** 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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/27297`.

We now turn to knowledge representation approaches that are based on some kind of logical system. These have the advantage that we know exactly what we are doing: as they are based on symbolic representations and declaratively given inference calculi as process models, we can inspect them thoroughly and even prove facts about them.

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)

But of course logic-based approaches have big drawbacks as well. The first is that we have to obtain the symbolic representations of knowledge to do anything – a non-trivial challenge, since most knowledge does not exist in this form in the wild, to obtain it, some agent has to experience the word, pass it through its cognitive apparatus, conceptualize the phenomena involved, systematize them sufficiently to form symbols, and then represent those in the respective formalism at hand.

The second drawback is that the process models induced by logic-based approaches (inference with calculi) are quite intractable. We will see that all inferences can be played back to satisfiability tests in the underlying logical system, which are exponential at best, and undecidable or even incomplete at worst.

Therefore a major thrust in logic-based knowledge representation is to investigate logical systems that are expressive enough to be able to represent most knowledge, but still have a decidable – and maybe even tractable in practice – satisfiability problem. Such logics are called "description logics". We will study the basics of such logical systems and their inference procedures in the following.

16.2.1 Propositional Logic as a Set Description Language

Before we look at "real" description logics in **??**, we will make a "dry run" with a logic we already understand: propositional logic, which we will re-interpret the system as a set description language by giving a new, non-standard semantics. This allows us to already preview most of the inference procedures and knowledge services of knowledge representation systems in the next subsection.
To establish propositional logic as a set description language, we use a different interpretation than usual. We interpret propositional variables as names of sets and the connectives as set operations, which is why we give them a different – more suggestive – syntax.

Propositional Logic as Set Description Language

▷ **Idea:** Use propositional logic as a set description language: (variant syntax/semantics)

▷ **Definition 16.2.1.** 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 16.2.2 (Formal Semantics).** Let $\mathcal{D}$ be a given set (called the domain of discourse) and $\varphi\colon \mathcal{V}_0 \to \mathcal{P}(\mathcal{D})$, then we define

- ▷ $[\![P]\!] := \varphi(P)$, (remember $\varphi(P) \subseteq \mathcal{D}$).
- ▷ $[\![\mathbf{A}\sqcap\mathbf{B}]\!] := [\![\mathbf{A}]\!] \cap [\![\mathbf{B}]\!]$ and $[\![\overline{\mathbf{A}}]\!] := \mathcal{D}\backslash[\![\mathbf{A}]\!]$ ...

We call this construction the set description semantics of PL^0.

▷ **Note:** $\langle \mathrm{PL}^0_{\mathrm{DL}}, \mathcal{S}, [\![\cdot]\!]\rangle$, where $\mathcal{S}$ is the class of possible domains forms a logical system.

The main use of the set-theoretic semantics for PL^0 is that we can use it to give meaning to concept axioms, which we use to describe the "world".

Concept Axioms

▷ **Observation:** Set-theoretic semantics of 'true' and 'false' ($\top := \varphi\sqcup\overline{\varphi}$ $\bot := \varphi\sqcap\overline{\varphi}$)

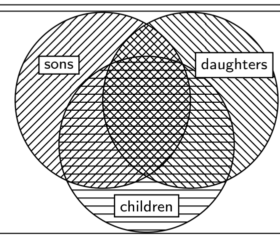
$$[\![\top]\!] = [\![p]\!] \cup [\![\overline{p}]\!] = [\![p]\!] \cup \mathcal{D}\backslash[\![p]\!] = \mathcal{D} \qquad \text{Analogously: } [\![\bot]\!] = \emptyset$$

▷ **Idea:** Use logical axioms to describe the world (Axioms restrict the class of admissible domain structures)

▷ **Definition 16.2.3.** A concept axiom is a $\mathrm{PL}^0_{\mathrm{DL}}$ formula $\mathbf{A}$ that is assumed to be true in the world.

▷ **Definition 16.2.4 (Set-Theoretic Semantics of Axioms).** $\mathbf{A}$ is true in domain of discourse $\mathcal{D}$ iff $[\![\mathbf{A}]\!] = \mathcal{D}$.

▷ **Example 16.2.5.** A world with three concepts and no concept axioms

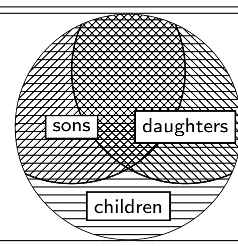
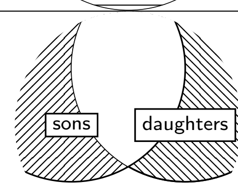
concepts	Set Semantics
child daughter son	

Concept axioms are used to restrict the set of admissible domains to the intended ones. In our situation, we require them to be true – as usual – which here means that they denote the whole domain $\mathcal{D}$.

Let us fortify our intuition about concept axioms with a simple example about the sibling relation. We give four concept axioms and study their effect on the admissible models by looking at the respective Venn diagrams. In the end we see that in all admissible models, the denotations of the concepts son and daughter are disjointq, and child is the union of the two – just as intended.

Effects of Axioms to Siblings

▷ **Example 16.2.6.** We can use concept axioms to describe the world from ??.

Axioms	Semantics
son $\sqsubseteq$ child iff $[\![\overline{\text{son}}]\!] \cup [\![\text{child}]\!] = \mathcal{D}$ iff $[\![\text{son}]\!] \subseteq [\![\text{child}]\!]$ daughter $\sqsubseteq$ child iff $\overline{[\![\text{daughter}]\!]} \cup [\![\text{child}]\!] = \mathcal{D}$ iff $[\![\text{daughter}]\!] \subseteq [\![\text{child}]\!]$	
$\overline{\text{son} \sqcap \text{daughter}}$ child $\sqsubseteq$ son $\sqcup$ daughter	

The set-theoretic semantics introduced above is compatible with the regular semantics of propositional logic, therefore we have the same propositional identities. Their validity can be established directly from the settings in **??**.

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	$\overline{\overline{\varphi}} = \varphi$	

Michael Kohlhase: Artificial Intelligence 1 500 2025-02-06

There is another way we can approach the set description interpretation of propositional logic: by translation into a logic that can express knowledge about sets – first-order logic.

Set-Theoretic Semantics and Predicate Logic

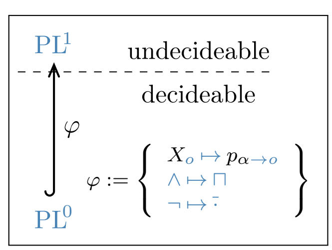
▷ **Definition 16.2.7.** Translation into PL^1 (borrow semantics from that)

- ▷ 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
$\overline{p}^{fo(x)} := p(x)$	
$\overline{\overline{\mathbf{A}}}^{fo(x)} := \neg\overline{\mathbf{A}}^{fo(x)}$	
$\overline{\mathbf{A} \sqcap \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \wedge \overline{\mathbf{B}}^{fo(x)}$	$\wedge$ vs. $\sqcap$
$\overline{\mathbf{A} \sqcup \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \vee \overline{\mathbf{B}}^{fo(x)}$	$\vee$ vs. $\sqcup$
$\overline{\mathbf{A} \sqsubseteq \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \Rightarrow \overline{\mathbf{B}}^{fo(x)}$	$\Rightarrow$ vs. $\sqsubseteq$
$\overline{\mathbf{A} = \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \Leftrightarrow \overline{\mathbf{B}}^{fo(x)}$	$\Leftrightarrow$ vs. $=$
$\overline{\mathbf{A}}^{fo} := (\forall x.\overline{\mathbf{A}}^{fo(x)})$	for formulae

Michael Kohlhase: Artificial Intelligence 1 501 2025-02-06

Normally, we embed PL^0 into PL^1 by mapping propositional variables to atomic first-order propositions and the connectives to themselves. The purpose of this embedding is to "talk about truth/falsity of assertions". For "talking about sets" we use a non-standard embedding: propositional variables in PL^0 are mapped to first-order predicates, and the connectives to corresponding set operations. This uses the convention that a set S is represented by a unary predicate p_S (its characteristic predicate), and set membership $a \in S$ as $p_S(a)$.



Translation Examples

- **Example 16.2.8.** We translate the concept axioms from ?? to fortify our intuition:

$$\begin{aligned}
\overline{\text{son} \sqsubseteq \text{child}}^{fo} &= \forall x.\text{son}(x) \Rightarrow \text{child}(x) \\
\overline{\text{daughter} \sqsubseteq \text{child}}^{fo} &= \forall x.\text{daughter}(x) \Rightarrow \text{child}(x) \\
\overline{\overline{\text{son} \sqcap \text{daughter}}}^{fo} &= \forall x.\overline{\text{son}(x) \wedge \text{daughter}(x)} \\
\overline{\text{child} \sqsubseteq \text{son} \sqcup \text{daughter}}^{fo} &= \forall x.\text{child}(x) \Rightarrow (\text{son}(x) \vee \text{daughter}(x))
\end{aligned}$$

- 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?)

Michael Kohlhase: Artificial Intelligence 1 502 2025-02-06

16.2.2 Ontologies and Description Logics

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27298`.
We have seen how sets of concept axioms can be used to describe the "world" by restricting the set of admissible models. We want to call such concept descriptions "ontologies" – formal descriptions of (classes of) objects and their relations.

Ontologies aka. "World Descriptions"

- **Definition 16.2.9 (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:** ?? 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 16.2.10.** An ontology consists of a formal system $\langle \mathcal{L}, \mathcal{C}, \mathcal{K}, \models \rangle$ 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 16.2.11.** Semantic networks are ontologies. (relatively informal)

- **Example 16.2.12.** $\mathrm{PL}^0_{\mathrm{DL}}$ is an ontology format. (formal, but relatively weak)

- **Example 16.2.13.** PL^1 is an ontology format as well. (formal, expressive)

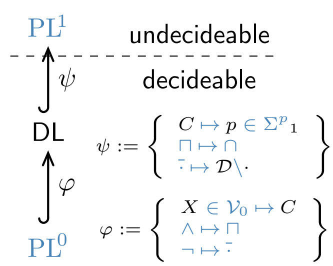
As we will see, the situation for $\mathrm{PL}^0_{\mathrm{DL}}$ is typical for formal ontologies (even though it only offers concepts), so we state the general description logic paradigm for ontologies. The important idea is that having a formal system as an ontology format allows us to capture, study, and implement ontological inference.

The Description Logic Paradigm

- **Idea:** Build a whole family of logics for describing sets and their relations. (tailor their expressivity and computational properties)
- **Definition 16.2.14.** 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.



- **Definition 16.2.15.** 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.

For convenience we add concept definitions as a mechanism for defining new concepts from old ones. The so-defined concepts inherit the properties from the concepts they are defined from.

TBoxes in Description Logics

- Let $\mathcal{D}$ be a description logic with concepts $\mathcal{C}$.
- **Definition 16.2.16.** A concept definition is a pair $c{=}\mathbf{C}$, where c is a new concept name and $\mathbf{C} \in \mathcal{C}$ is a $\mathcal{D}$-formula.
- **Example 16.2.17.** We can define mother=woman $\sqcap$ has_child.
- **Definition 16.2.18.** A concept definition $c{=}C$ is called recursive, iff c occurs in C.
- **Definition 16.2.19.** An TBox is a finite set of concept definitions and concept axioms. It is called acyclic, iff it does not contain recursive definitions.

▷ **Definition 16.2.20.** A formula $\mathbf{A}$ is called normalized wrt. an TBox $\mathcal{T}$, iff it does not contain concepts defined in $\mathcal{T}$. (convenient)

▷ **Definition 16.2.21 (Algorithm).** (for arbitrary DLs)
Input: A formula $\mathbf{A}$ and a TBox $\mathcal{T}$.

- ▷ While [$\mathbf{A}$ contains concept c and $\mathcal{T}$ a concept definition $c=\mathbf{C}$]
  - ▷ substitute c by $\mathbf{C}$ in $\mathbf{A}$.

▷ **Lemma 16.2.22.** *This algorithm terminates for acyclic TBoxes, but results can be exponentially large.*

As $\mathrm{PL}^0_{\mathrm{DL}}$ does not offer any guidance on this, we will leave the discussion of ABoxes to **??** when we have introduced our first proper description logic $\mathcal{ALC}$.

16.2.3 Description Logics and Inference

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27299`.

Now that we have established the description logic paradigm, we will have a look at the inference services that can be offered on this basis.

Before we go into details of particular description logics, we must ask ourselves what kind of inference support we would want for building systems that support knowledge workers in building, maintaining and using ontologies. An example of such a system is the Protégé system [Pro], which can serve for guiding our intuition.

Kinds of Inference in Description Logics

▷ **Definition 16.2.23.** 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

We will now through these inference-based tests separately.

The consistency test checks for concepts that do not/cannot have instances. We want to avoid such concepts in our ontologies, since they clutter the namespace and do not contribute any meaningful contribution.

Consistency Test

▷ **Definition 16.2.24.** 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 16.2.25.** A concept C is called inconsistent, iff $[\![C]\!] = \emptyset$ for all $\mathcal{D}$.

▷ **Example 16.2.26 (T-Box in $\mathrm{PL}^0_{\mathrm{DL}}$).**

man	=	person ⊓ has_Y	person with y-chromosome
woman	=	person ⊓ $\overline{\text{has\_Y}}$	person without y-chromosome
hermaphrodite	=	man ⊓ woman	man and woman

This specification is inconsistent, i.e. $[\![\text{hermaphrodite}]\!] = \emptyset$ for all $\mathcal{D}$.

▷ **Algorithm:** Satisfiability test (usually NP complete)
we know how to do this, e.g. tableaux, resolution, DPLL in $\mathrm{PL}^0_{\mathrm{DL}}$.

Even though consistency in our example seems trivial, large ontologies can make machine support necessary. This is even more true for ontologies that change over time. Say that an ontology initially has the concept definitions woman=person⊓long_hair and man=person⊓bearded, and then is modernized to a more biologically correct state. In the initial version the concept hermaphrodite is consistent, but becomes inconsistent after the renovation; the authors of the renovation should be made aware of this by the system.
The subsumption test determines whether the sets denoted by two concepts are in a subset relation. The main justification for this is that humans tend to be aware of concept subsumption, and tend to think in taxonomic hierarchies. To cater to this, the subsumption test is useful.

Subsumption Test

▷ **Example 16.2.27.** In this case trivial

axiom	entailed subsumption relation
man = person ⊓ has_Y	man ⊑ person
woman = person ⊓ $\overline{\text{has\_Y}}$	woman ⊑ person

▷ **Definition 16.2.28.** $\mathbf{A}$ subsumes $\mathbf{B}$ (modulo a set $\mathcal{A}$ of concept axioms), iff $[\![\mathbf{B}]\!] \subseteq [\![\mathbf{A}]\!]$ for all interpretations $\mathcal{D}$ that satisfy $\mathcal{A}$.

▷ **Observation:** Or equivalently, iff $\mathcal{A} \sqsubseteq \mathbf{B} \sqsubseteq \mathbf{A} = \top$

▷ **Reduction to consistency test:** (need to implement only one)
In PL^0, $\mathcal{A} \Rightarrow (\mathbf{A} \Rightarrow \mathbf{B})$ is valid iff $\mathcal{A} \wedge \mathbf{A} \wedge \neg\mathbf{B}$ is inconsistent.

▷ **In our example:** The concept person subsumes woman and man.

The good news is that we can reduce the subsumption test to the consistency test, so we can re-use our existing implementation.
The main user-visible service of the subsumption test is to compute the actual taxonomy induced by an ontology.

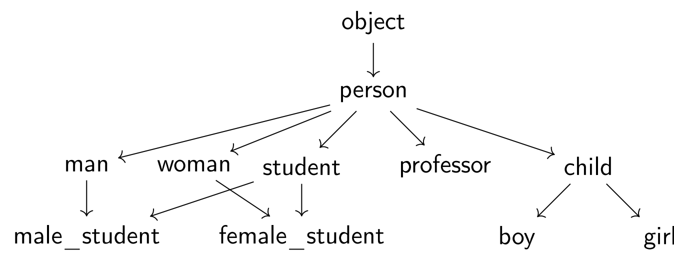
Classification

▷ The subsumption relation among all concepts (subsumption graph)

▷ Visualization of the subsumption graph for inspection (plausibility)

▷ **Definition 16.2.29.** Classification is the computation of the subsumption graph.

▷ **Example 16.2.30.** (not always so trivial)



Instance Test: Inferring Concept Membership

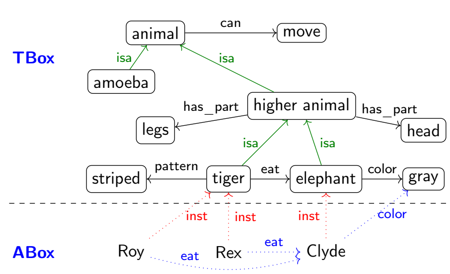
▷ **Definition 16.2.31.** An instance test computes whether given an ontology an individual is a member of a given concept.

▷ **Remark:** This is not something we can do in $\mathrm{PL}^0_{\mathrm{DL}}$, which is a TBox-only system. PL^1 (where concepts are predicate constants an assertions are atoms) suffices.

▷ **Example 16.2.32.** If we define a concept "mother" as "woman who has a child", and have the assertions "Mary is a woman" and "Jesus is a child of Mary", then we can infer that "Mary" is a "Mother", e.g. in the $\mathcal{ND}^1$:

$$\forall x.m(x) \Leftrightarrow w(x) \wedge (\exists y.hc(x,y)), w(M), hc(M,J) \vdash_{\mathcal{ND}^1} m(M)$$

▷ **Remark:** This only works in the presence of concept definitions, not in a purely descriptive framework like semantic networks:



If we take stock of what we have developed so far, then we can see $\mathrm{PL}^0_{\mathrm{DL}}$ as a rational reconstruction of semantic networks restricted to the "isa" relation. We relegate the "instance" relation to **??**.

This reconstruction can now be used as a basis on which we can extend the expressivity and inference procedures without running into problems.

16.3 A simple Description Logic: ALC

In this section, we instantiate the description-logic paradigm further with the prototypical logic $\mathcal{ALC}$, which we will introduce now.

16.3.1 Basic ALC: Concepts, Roles, and Quantification

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27300`.

In this subsection, we instantiate the description-logic paradigm with the prototypical logic $\mathcal{ALC}$, which we will develop now.

Motivation for $\mathcal{ALC}$ (Prototype Description Logic)

- ▷ Propositional logic (PL^0) is not expressive enough!
- ▷ **Example 16.3.1.** "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)

$$\forall x.mother(x) \Leftrightarrow woman(x) \wedge (\exists y.has\_child(x,y))$$

- ▷ **Problem:** Complex algorithms, non-termination (PL^1 is too expressive)
- ▷ **Idea:** Try to travel the middle ground
  More expressive than PL^0 (quantifiers) but weaker than PL^1. (still tractable)
- ▷ **Technique:** Allow only "restricted quantification", where quantified variables only range over values that can be reached via a binary relation like has_child.

FAU Michael Kohlhase: Artificial Intelligence 1 511 2025-02-06

$\mathcal{ALC}$ extends the concept operators of $\mathrm{PL}^0_{\mathrm{DL}}$ with binary relations (called "roles" in $\mathcal{ALC}$). This gives $\mathcal{ALC}$ the expressive power we had for the basic semantic networks from ??.

Syntax of $\mathcal{ALC}$

- ▷ **Definition 16.3.2 (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 16.3.3 (Special Concepts).** The top concept $\top$ (for "true" or "all") and the bottom concept $\bot$ (for "false" or "none").
- ▷ **Example 16.3.4.** person, woman, man, mother, professor, student, car, BMW, computer, computer program, heart attack risk, furniture, table, leg of a chair, ...
- ▷ **Definition 16.3.5.** Roles represent binary relations (like in PL^1)
- ▷ **Example 16.3.6.** has_child, has_son, has_daughter, loves, hates, gives_course, executes_computer_program, has_leg_of_table, has_wheel, has_motor, ...
- ▷ **Definition 16.3.7 (Grammar).** The formulae of $\mathcal{ALC}$ are given by the following grammar: $F_{\mathcal{ALC}} ::= C \mid \top \mid \bot \mid \overline{F_{\mathcal{ALC}}} \mid F_{\mathcal{ALC}} \sqcap F_{\mathcal{ALC}} \mid F_{\mathcal{ALC}} \sqcup F_{\mathcal{ALC}} \mid \exists \mathsf{R}.F_{\mathcal{ALC}} \mid \forall \mathsf{R}.F_{\mathcal{ALC}}$

FAU Michael Kohlhase: Artificial Intelligence 1 512 2025-02-06

$\mathcal{ALC}$ restricts the quantification to range all individuals reachable as role successor. The distinction between universal and existential quantifiers clarifies an implicit ambiguity in semantic networks.

Syntax of $\mathcal{ALC}$: Examples

- **Example 16.3.8.** person $\sqcap$ $\exists$has_child.student
  $\widehat{=}$ The set of persons that have a child which is a student
  $\widehat{=}$ parents of students
- **Example 16.3.9.** person $\sqcap$ $\exists$has_child.$\exists$has_child.student
  $\widehat{=}$ grandparents of students
- **Example 16.3.10.** person $\sqcap$ $\exists$has_child.$\exists$has_child.(student $\sqcup$ teacher)
  $\widehat{=}$ grandparents of students or teachers
- **Example 16.3.11.** person $\sqcap$ $\forall$has_child.student
  $\widehat{=}$ parents whose children are all students
- **Example 16.3.12.** 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 16.3.13.** car $\sqcap$ $\exists$has_part.$\exists$made_in.$\overline{\text{EU}}$
  $\widehat{=}$ cars that have at least one part that has not been made in the EU
- **Example 16.3.14.** student $\sqcap$ $\forall$audits_course.graduatelevelcourse
  $\widehat{=}$ students, that only audit graduate level courses
- **Example 16.3.15.** house$\sqcap$$\forall$has_parking.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 16.3.16.** student $\sqcap$ $\forall$audits_course.($\exists$hastutorial.$\top$ $\sqsubseteq$ $\forall$has_TA.woman)
  $\widehat{=}$ students that only audit courses that either have no tutorial or tutorials that are TAed by women

As before we allow concept definitions so that we can express new concepts from old ones, and obtain more concise descriptions.

$\mathcal{ALC}$ Concept Definitions

- **Idea:** Define new concepts from known ones.
- **Definition 16.3.17.** A concept definition is a pair consisting of a new concept name (the definiendum) and an $\mathcal{ALC}$ formula (the definiens). Concepts that are not

definienda are called primitive.

▷ We extend the $\mathcal{ALC}$ grammar from ?? by the production

$$CD_{\mathcal{ALC}} ::= C = F_{\mathcal{ALC}}$$

▷ **Example 16.3.18.**

Definition	rec?
man = person ⊓ ∃has_chrom.Y_chrom	-
woman = person ⊓ ∀has_chrom.$\overline{\text{Y\_chrom}}$	-
mother = woman ⊓ ∃has_child.person	-
father = man ⊓ ∃has_child.person	-
grandparent = person ⊓ ∃has_child.(mother ⊔ father)	-
german = person ⊓ ∃has_parents.german	+
number_list = empty_list ⊔ ∃is_first.number ⊓ ∃is_rest.number_list	+

As before, we can normalize a TBox by definition expansion if it is acyclic. With the introduction of roles and quantification, concept definitions in $\mathcal{ALC}$ have a more "interesting" way to be cyclic as ?? shows.

TBox Normalization in $\mathcal{ALC}$

▷ **Definition 16.3.19.** We call an $\mathcal{ALC}$ formula φ normalized wrt. a set of concept definitions, iff all concepts occurring in φ are primitive.

▷ **Definition 16.3.20.** 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 16.3.21 (Normalizing grandparent).**

grandparent
↦ person ⊓ ∃has_child.(mother ⊔ father)
↦ person ⊓ ∃has_child.(woman ⊓ ∃has_child.person ⊓ man ⊓ ∃has_child.person)
↦ person ⊓ ∃has_child.(person ⊓ ∃has_chrom.Y_chrom ⊓ ∃has_child.person ⊓ person ⊓ ∃has_chrom.Y_chrom ⊓ ∃has_child.person)

▷ **Observation 16.3.22.** *Normalization results can be exponential.* *(contain redundancies)*

▷ **Observation 16.3.23.** *Normalization need not terminate on cyclic TBoxes.*

▷ **Example 16.3.24.**

german ↦ person ⊓ ∃has_parents.german
↦ person ⊓ ∃has_parents.(person ⊓ ∃has_parents.german)
↦ ...

Now that we have motivated and fixed the syntax of $\mathcal{ALC}$, we will give it a formal semantics. The semantics of $\mathcal{ALC}$ is an extension of the set-theoretic semantics for PL^0, thus the interpretation $[\![\cdot]\!]$ assigns subsets of the domain of discourse to concepts and binary relations over the domain of discourse to roles.

Semantics of $\mathcal{ALC}$

- $\mathcal{ALC}$ semantics is an extension of the set-semantics of propositional logic.
- **Definition 16.3.25.** A model for $\mathcal{ALC}$ is a pair $\langle\mathcal{D},[[\cdot]]\rangle$, where $\mathcal{D}$ is a non-empty set called the domain of discourse and $[[\cdot]]$ a mapping called the interpretation, such that

Op.	formula semantics
	$[\![c]\!]\subseteq\mathcal{D}=[\![\top]\!]$ $\quad[\![\bot]\!]=\emptyset$ $\quad[\![r]\!]\subseteq\mathcal{D}\times\mathcal{D}$
$\overline{\cdot}$	$[\![\overline{\varphi}]\!]=\overline{[\![\varphi]\!]}=\mathcal{D}\backslash[\![\varphi]\!]$
$\sqcap$	$[\![\varphi\sqcap\psi]\!]=[\![\varphi]\!]\cap[\![\psi]\!]$
$\sqcup$	$[\![\varphi\sqcup\psi]\!]=[\![\varphi]\!]\cup[\![\psi]\!]$
$\exists\mathsf{R}.$	$[\![\exists\mathsf{R}.\varphi]\!]=\{x\in\mathcal{D}\,|\,\exists y.\langle x,y\rangle\in[\![\mathsf{R}]\!]\text{ and }y\in[\![\varphi]\!]\}$
$\forall\mathsf{R}.$	$[\![\forall\mathsf{R}.\varphi]\!]=\{x\in\mathcal{D}\,|\,\forall y.\text{if }\langle x,y\rangle\in[\![\mathsf{R}]\!]\text{ then }y\in[\![\varphi]\!]\}$

- Alternatively we can define the semantics of $\mathcal{ALC}$ by translation into PL^1.
- **Definition 16.3.26.** The translation of $\mathcal{ALC}$ into PL^1 extends the one from **??** by the following quantifier rules:

$$\overline{\forall\mathsf{R}.\varphi}^{fo(x)}:=(\forall y.\mathsf{R}(x,y)\Rightarrow\overline{\varphi}^{fo(y)})\qquad\overline{\exists\mathsf{R}.\varphi}^{fo(x)}:=(\exists y.\mathsf{R}(x,y)\wedge\overline{\varphi}^{fo(y)})$$

- **Observation 16.3.27.** *The set-theoretic semantics from* **??** *and the "semantics-by-translation" from* **??** *induce the same notion of satisfiability.*

We can now use the $\mathcal{ALC}$ identities above to establish a useful normal form for $\mathcal{ALC}$. This will play a role in the inference procedures we study next.
The following identitieswill be useful later on. They can be proven directly with the settings from **??**; we carry this out for one of them below.

$\mathcal{ALC}$ Identities

1	$\overline{\exists\mathsf{R}.\varphi}=\forall\mathsf{R}.\overline{\varphi}$	3	$\overline{\forall\mathsf{R}.\varphi}=\exists\mathsf{R}.\overline{\varphi}$
2	$\forall\mathsf{R}.(\varphi\sqcap\psi)=\forall\mathsf{R}.\varphi\sqcap\forall\mathsf{R}.\psi$	4	$\exists\mathsf{R}.(\varphi\sqcup\psi)=\exists\mathsf{R}.\varphi\sqcup\exists\mathsf{R}.\psi$

- Proof of 1

$$\begin{aligned}
[[\overline{\exists\mathsf{R}.\varphi}]]=\mathcal{D}\backslash[\![\exists\mathsf{R}.\varphi]\!] &= \mathcal{D}\backslash\{x\in\mathcal{D}\,|\,\exists y.(\langle x,y\rangle\in[\![\mathsf{R}]\!])\text{ and }(y\in[\![\varphi]\!])\}\\
&= \{x\in\mathcal{D}\,|\,\text{not }\exists y.(\langle x,y\rangle\in[\![\mathsf{R}]\!])\text{ and }(y\in[\![\varphi]\!])\}\\
&= \{x\in\mathcal{D}\,|\,\forall y.\text{if }(\langle x,y\rangle\in[\![\mathsf{R}]\!])\text{ then }(y\notin[\![\varphi]\!])\}\\
&= \{x\in\mathcal{D}\,|\,\forall y.\text{if }(\langle x,y\rangle\in[\![\mathsf{R}]\!])\text{ then }(y\in(\mathcal{D}\backslash[\![\varphi]\!]))\}\\
&= \{x\in\mathcal{D}\,|\,\forall y.\text{if }(\langle x,y\rangle\in[\![\mathsf{R}]\!])\text{ then }(y\in[\![\overline{\varphi}]\!])\}\\
&= [\![\forall\mathsf{R}.\overline{\varphi}]\!]
\end{aligned}$$

The form of the identities (interchanging quantification with connectives) is reminiscient of identities in PL¹; this is no coincidence as the "semantics by translation" of **??** shows.

Negation Normal Form

▷ **Definition 16.3.28 (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 16.3.29.**

example	by rule
$\overline{\exists R.(\forall S.e \sqcap \overline{\forall S.d})}$	
$\mapsto \forall R.\overline{\forall S.e \sqcap \overline{\forall S.d}}$	$\overline{\exists R.\varphi} \mapsto \forall R.\overline{\varphi}$
$\mapsto \forall R.(\overline{\forall S.e} \sqcup \overline{\overline{\forall S.d}})$	$\overline{\varphi \sqcap \psi} \mapsto \overline{\varphi} \sqcup \overline{\psi}$
$\mapsto \forall R.(\exists S.\overline{e} \sqcup \overline{\overline{\forall S.d}})$	$\overline{\forall R.\varphi} \mapsto \exists R.\overline{\varphi}$
$\mapsto \forall R.(\exists S.\overline{e} \sqcup \forall S.d)$	$\overline{\overline{\varphi}} \mapsto \varphi$

Finally, we extend $\mathcal{ALC}$ with an ABox component. This mainly means that we define two new assertions in $\mathcal{ALC}$ and specify their semantics and PL¹ translation.

$\mathcal{ALC}$ with Assertions about Individuals

▷ **Definition 16.3.30.** We define the ABox assertions for $\mathcal{ALC}$:

- ▷ Role assertions$a{:}\varphi$ (a is a φ)
- ▷ a R b (a stands in relation R to b)

assertions make up the ABox in $\mathcal{ALC}$.

▷ **Definition 16.3.31.** Let $\langle \mathcal{D}, [[\cdot]] \rangle$ be a model for $\mathcal{ALC}$, then we define

- ▷ $[\![a{:}\varphi]\!] = \mathsf{T}$, iff $[\![a]\!] \in [\![\varphi]\!]$, and
- ▷ $[\![a \mathrel{\mathsf{R}} b]\!] = \mathsf{T}$, iff $([\![a]\!], [\![b]\!]) \in [\![\mathsf{R}]\!]$.

▷ **Definition 16.3.32.** We extend the PL¹ translation of $\mathcal{ALC}$ to $\mathcal{ALC}$ assertions:

- ▷ $\overline{a{:}\varphi}^{fo} := \overline{\varphi}^{fo(a)}$, and
- ▷ $\overline{a \mathrel{\mathsf{R}} b}^{fo} := \mathsf{R}(a, b)$.

If we take stock of what we have developed so far, then we see that $\mathcal{ALC}$ as a rational reconstruction of semantic networks restricted to the "isa" and "instance" relations – which are the only ones that can really be given a denotational and operational semantics.

16.3.2 Inference for ALC

Video Nuggets covering this subsection can be found at `https://fau.tv/clip/id/27301` and `https://fau.tv/clip/id/27302`.

In this subsection we make good on the motivation from ?? that description logics enjoy tractable inference procedures: We present a tableau calculus for $\mathcal{ALC}$, show that it is a decision procedure, and study its complexity.

$\mathcal{T}_{\mathcal{ALC}}$: A Tableau-Calculus for $\mathcal{ALC}$

- **Recap Tableaux:** A tableau calculus develops an initial tableau in a tree-formed scheme using tableau extension rules.

  A saturated tableau (no rules applicable) constitutes a refutation, if all branches are closed (end in $\bot$).

- **Definition 16.3.33.** The $\mathcal{ALC}$ tableau calculus $\mathcal{T}_{\mathcal{ALC}}$ acts on assertions:
  - $x{:}\varphi$ (x inhabits concept φ)
  - $x \; \mathsf{R} \; y$ (x and y are in relation R)

  where φ is a normalized $\mathcal{ALC}$ concept in negation normal form with the following rules:

$$\frac{\begin{array}{c} x{:}c \\ x{:}\overline{c} \end{array}}{\bot}\,\mathcal{T}_\bot \qquad \frac{x{:}\varphi \sqcap \psi}{\begin{array}{c} x{:}\varphi \\ x{:}\psi \end{array}}\,\mathcal{T}_\sqcap \qquad \frac{x{:}\varphi \sqcup \psi}{x{:}\varphi \;\big|\; x{:}\psi}\,\mathcal{T}_\sqcup \qquad \frac{\begin{array}{c} x{:}\forall \mathsf{R}.\varphi \\ x \; \mathsf{R} \; y \end{array}}{y{:}\varphi}\,\mathcal{T}_\forall \qquad \frac{x{:}\exists \mathsf{R}.\varphi}{\begin{array}{c} x \; \mathsf{R} \; y \\ y{:}\varphi \end{array}}\,\mathcal{T}_\exists$$

- To test consistency of a concept φ, normalize φ to ψ, initialize the tableau with $x{:}\psi$, saturate. Open branches $\rightsquigarrow$ consistent. (x arbitrary)

FAU Michael Kohlhase: Artificial Intelligence 1 521 2025-02-06

In contrast to the tableau tableau calculi for theorem proving we have studied earlier, $\mathcal{T}_{\mathcal{ALC}}$ is run in "model generation mode". Instead of initializing the tableau with the axioms and the negated conjecture and hope that all branches will close, we initialize the $\mathcal{T}_{\mathcal{ALC}}$ tableau with axioms and the "membership-conjecture" that a given concept φ is satisfiable – i.e. φ h as a member x, and hope for branches that are open, i.e. that make the conjecture true (and at the same time give a model).
Let us now work through two very simple examples; one unsatisfiable, and a satisfiable one.

$\mathcal{T}_{\mathcal{ALC}}$ Examples

- **Example 16.3.34 (Tableau Proofs).** We have two similar conjectures about children.
  - $x{:}\forall \mathsf{has\_child}.\mathsf{man} \sqcap \exists \mathsf{has\_child}.\overline{\mathsf{man}}$ (all sons, but a daughter)

$x{:}\forall \mathsf{has\_child}.\mathsf{man} \sqcap \exists \mathsf{has\_child}.\overline{\mathsf{man}}$	initial
$x{:}\forall \mathsf{has\_child}.\mathsf{man}$	$\mathcal{T}_\sqcap$
$x{:}\exists \mathsf{has\_child}.\overline{\mathsf{man}}$	$\mathcal{T}_\sqcap$
$x \; \mathsf{has\_child} \; y$	$\mathcal{T}_\exists$
$y{:}\overline{\mathsf{man}}$	$\mathcal{T}_\exists$
$\bot$	$\mathcal{T}_\bot$
inconsistent	

  - $x{:}\forall \mathsf{has\_child}.\mathsf{man} \sqcap \exists \mathsf{has\_child}.\mathsf{man}$ (only sons, and at least one)

x:∀has_child.man ⊓ ∃has_child.man	initial
x:∀has_child.man	$\mathcal{T}_\sqcap$
x:∃has_child.man	$\mathcal{T}_\sqcap$
x has_child y	$\mathcal{T}_\exists$
y: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.

Michael Kohlhase: Artificial Intelligence 1 522 2025-02-06

Another example: this one is more complex, but the concept is satisfiable.

Another $\mathcal{T}_{\mathcal{ALC}}$ Example

- **Example 16.3.35.** ∀has_child.(ugrad ⊔ grad) ⊓ ∃has_child.$\overline{\text{ugrad}}$ is satisfiable.
  - Let's try it on the board
  - Tableau proof for the notes

1	x:∀has_child.(ugrad ⊔ grad) ⊓ ∃has_child.$\overline{\text{ugrad}}$		initial
2	x:∀has_child.(ugrad ⊔ grad)		$\mathcal{T}_\sqcap$
3	x:∃has_child.$\overline{\text{ugrad}}$		$\mathcal{T}_\sqcap$
4	x has_child y		$\mathcal{T}_\exists$
5	y:$\overline{\text{ugrad}}$		$\mathcal{T}_\exists$
6	y:ugrad ⊔ grad		$\mathcal{T}_\forall$
7	y:ugrad	y:grad	$\mathcal{T}_\sqcup$
8	⊥	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.

Michael Kohlhase: Artificial Intelligence 1 523 2025-02-06

After we got an intuition about $\mathcal{T}_{\mathcal{ALC}}$, we can now study the properties of the calculus to determine that it is a decision procedure for $\mathcal{ALC}$.

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)

The soundness result for $\mathcal{T}_{\mathcal{ALC}}$ is as usual: we start with a model of $x{:}\varphi$ and show that an $\mathcal{T}_{\mathcal{ALC}}$ tableau must have an open branch.

Correctness

▷ **Lemma 16.3.36.** *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 $\begin{array}{lll} \mathcal{M}\models(x{:}\psi) & \text{iff} & [\![x]\!] \in [\![\psi]\!] \\ \mathcal{M}\models x \mathrel{\mathsf{R}} y & \text{iff} & \langle x,y\rangle \in [\![\mathsf{R}]\!] \\ \mathcal{M}\models S & \text{iff} & \mathcal{I}\models c \text{ for all } c \in S \end{array}$
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)

We complete the proof by looking at all the $\mathcal{T}_{\mathcal{ALC}}$ inference rules in turn.

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 $\mathcal{M}\models(x{:}\forall\mathsf{R}.\varphi)$ and $\mathcal{M}\models x \mathrel{\mathsf{R}} y$, i.e. $[\![x]\!] \in [\![\forall\mathsf{R}.\varphi]\!]$ and $\langle x,y\rangle \in [\![\mathsf{R}]\!]$, so $[\![y]\!] \in [\![\varphi]\!]$

$\mathcal{T}_{\exists}$ **applies** then $\mathcal{M}\models(x{:}\exists\mathsf{R}.\varphi)$, i.e $[\![x]\!] \in [\![\exists\mathsf{R}.\varphi]\!]$,
so there is a $v \in D$ with $\langle [\![x]\!], v\rangle \in [\![\mathsf{R}]\!]$ and $v \in [\![\varphi]\!]$.
We define $[\![y]\!] := v$, then $\mathcal{M}\models x \mathrel{\mathsf{R}} y$ and $\mathcal{M}\models(y{:}\varphi)$

For the completeness result for $\mathcal{T}_{\mathcal{ALC}}$ we have to start with an open tableau branch and construct a model that satisfies all judgments in the branch. We proceed by building a Herbrand model, whose domain consists of all the individuals mentioned in the branch and which interprets all concepts and roles as specified in the branch. Not surprisingly, the model thus constructed satisfies (all judgments on) the branch.

Completeness of the Tableau Calculus

▷ **Lemma 16.3.37.** *Open saturated tableau branches for φ induce models for φ.*

▷ *Proof:* construct a model for the branch and verify for φ

1. Let $\mathcal{B}$ be an open, saturated branch

▷ we define

$$\begin{aligned}\mathcal{D} &:= \{x \mid x{:}\psi \in \mathcal{B} \text{ or } z \mathrel{\mathsf{R}} x \in \mathcal{B}\}\\ [\![c]\!] &:= \{x \mid x{:}c \in \mathcal{B}\}\\ [\![\mathsf{R}]\!] &:= \{\langle x,y\rangle \mid x \mathrel{\mathsf{R}} y \in \mathcal{B}\}\end{aligned}$$

▷ well-defined since never $x{:}c, x{:}\overline{c} \in \mathcal{B}$ (otherwise $\mathcal{T}_\bot$ applies)

▷ $\mathcal{M}$ satisfies all assertions $x{:}c$, $x{:}\overline{c}$ and $x \mathrel{\mathsf{R}} y$, (by construction)

2. $\mathcal{M}\models(y{:}\psi)$, for all $y{:}\psi \in \mathcal{B}$ (on $k = size(\psi)$ next slide)
3. $\mathcal{M}\models(x{:}\varphi)$.

We complete the proof by looking at all the $\mathcal{T}_{\mathcal{ALC}}$ inference rules in turn.

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 $y{:}\psi_1 \in \mathbf{B}$ or $y{:}\psi_2 \in \mathbf{B}$ ($\mathcal{T}_\sqcup$, 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 $\{y \mathrel{\mathsf{R}} z, z{:}\theta\} \subseteq \mathbf{B}$ (z new variable) ($\mathcal{T}_\exists$-rules, saturation)
so $\mathcal{M}\models(z{:}\theta)$ and $\mathcal{M}\models y \mathrel{\mathsf{R}} z$, thus $\mathcal{M}\models(y{:}\exists\mathsf{R}.\theta)$. (IH, Definition)

case $y{:}\psi = y{:}\forall\mathbf{R}.\theta$ Let $\langle [\![y]\!], v\rangle \in [\![\mathsf{R}]\!]$ for some $r \in \mathcal{D}$
then $v = z$ for some variable z with $y \mathrel{\mathsf{R}} z \in \mathbf{B}$ (construction of $[\![\mathsf{R}]\!]$)
So $z{:}\theta \in \mathcal{B}$ and $\mathcal{M}\models(z{:}\theta)$. ($\mathcal{T}_\forall$-rule, saturation, Def)
As v was arbitrary we have $\mathcal{M}\models(y{:}\forall\mathsf{R}.\theta)$.

Termination

▷ **Theorem 16.3.38.** *$\mathcal{T}_{\mathcal{ALC}}$ terminates.*

▷ To prove termination of a tableau algorithm, find a well-founded measure (function) that is decreased by all rules

$$\frac{\begin{matrix}x{:}c\\ x{:}\overline{c}\end{matrix}}{\bot}\mathcal{T}_\bot \qquad \frac{x{:}\varphi \sqcap \psi}{\begin{matrix}x{:}\varphi\\ x{:}\psi\end{matrix}}\mathcal{T}_\sqcap \qquad \frac{x{:}\varphi \sqcup \psi}{x{:}\varphi \mid x{:}\psi}\mathcal{T}_\sqcup \qquad \frac{\begin{matrix}x{:}\forall\mathsf{R}.\varphi\\ x \mathrel{\mathsf{R}} y\end{matrix}}{y{:}\varphi}\mathcal{T}_\forall \qquad \frac{x{:}\exists\mathsf{R}.\varphi}{\begin{matrix}x \mathrel{\mathsf{R}} y\\ y{:}\varphi\end{matrix}}\mathcal{T}_\exists$$

▷ *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 $x \mathrel{\mathsf{R}} y$ 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.

We can turn the termination result into a worst-case complexity result by examining the sizes of branches.

Complexity of $\mathcal{T}_{\mathcal{ALC}}$

- **Idea:** Work off tableau branches one after the other. (Branch size $\widehat{=}$ space complexity)
- **Observation 16.3.39.** *The size of the branches is polynomial in the size of the input formula:*

$$\text{branchsize} = \#(\textit{input formulae}) + \#(\exists\textit{-formulae}) \cdot \#(\forall\textit{-formulae})$$

- *Proof sketch:* Re-examine the termination proof and count: the first summand comes from ??, the second one from ?? and ??
- **Theorem 16.3.40.** *The satisfiability problem for $\mathcal{ALC}$ is in* **PSPACE**.
- **Theorem 16.3.41.** *The satisfiability problem for $\mathcal{ALC}$ is* **PSPACE***-Complete.*
- *Proof sketch:* Reduce a **PSPACE**-complete problem to $\mathcal{ALC}$-satisfiability
- **Theorem 16.3.42 (Time Complexity).** *The $\mathcal{ALC}$ satisfiability problem is in* **EXPTIME**.
- *Proof sketch:* There can be exponentially many branches (already for PL^0)

In summary, the theoretical complexity of $\mathcal{ALC}$ is the same as that for PL^0, but in practice $\mathcal{ALC}$ is much more expressive. So this is a clear win.
But the description of the tableau algorithm $\mathcal{T}_{\mathcal{ALC}}$ is still quite abstract, so we look at an exemplary implementation in a functional programming language.

The functional Algorithm for $\mathcal{ALC}$

- **Observation:** (leads to a better treatment for $\exists$)
  - the $\mathcal{T}_{\exists}$-rule generates the constraints $x \mathrel{\mathsf{R}} y$ and $y{:}\psi$ from $x{:}\exists \mathsf{R}.\psi$
  - this triggers the $\mathcal{T}_{\forall}$-rule for $x{:}\forall \mathsf{R}.\theta_i$, 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 16.3.43.** 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 '$\exists$R.ψ' $\in S$
    consistent($\{\psi\} \cup \{\theta \in \theta \mid$ '$\forall$R.θ' $\in S\}$)
  else true

- Relatively simple to implement. (good implementations optimized)
- **But:** This is restricted to $\mathcal{ALC}$. (extension to other DL difficult)

Note that we have (so far) only considered an empty TBox: we have initialized the tableau with a normalized concept; so we did not need to include the concept definitions. To cover "real" ontologies, we need to consider the case of concept axioms as well.
We now extend $\mathcal{T}_{\mathcal{ALC}}$ with concept axioms. The key idea here is to realize that the concept axioms apply to all individuals. As the individuals are generated by the $\mathcal{T}_\exists$ rule, we can simply extend that rule to apply all the concept axioms to the newly introduced individual.

Extending the Tableau Algorithm by Concept Axioms

- concept axioms, e.g. child $\sqsubseteq$ son $\sqcup$ daughter cannot be handled in $\mathcal{T}_{\mathcal{ALC}}$ yet.
- **Idea:** Whenever a new variable y is introduced (by $\mathcal{T}_\exists$-rule) add the information that axioms hold for y.
  - Initialize tableau with $\{x{:}\varphi\} \cup \mathcal{CA}$ ($\mathcal{CA}$: = set of concept axioms)
  - New rule for $\exists$: $$\frac{x{:}\exists \mathsf{R}.\varphi \quad \mathcal{CA} = \{\alpha_1, \ldots, \alpha_n\}}{\begin{array}{c} y{:}\varphi \\ x \;\mathsf{R}\; y \\ y{:}\alpha_1 \\ \vdots \\ y{:}\alpha_n \end{array}} \mathcal{T}^{\exists}_{\mathcal{CA}}$$ (instead of $\mathcal{T}_\exists$)
- **Problem:** $\mathcal{CA} := \{\exists \mathsf{R}.c\}$ and start tableau with $x{:}d$ (non-termination)

The problem of this approach is that it spoils termination, since we cannot control the number of rule applications by (fixed) properties of the input formulae. The example shows this very nicely. We only sketch a path towards a solution.

Non-Termination of $\mathcal{T}_{\mathcal{ALC}}$ with Concept Axioms

- **Problem:** $\mathcal{CA} := \{\exists \mathsf{R}.c\}$ and start tableau with $x{:}d$. (non-termination)

$x{:}d$	start
$x{:}\exists R.c$	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 16.3.44.** *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

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27303`.
Now that we have a decision problem for $\mathcal{ALC}$ with concept axioms, we can go the final step to the general case of inference in description logics: we add an ABox with assertional axioms that describe the individuals.
We will now extend the description logic $\mathcal{ALC}$ with assertions that can express concept membership.

Instance Test: Concept Membership

- **Definition 16.3.45.** An instance test computes whether given an $\mathcal{ALC}$ ontology an individual is a member of a given concept.
- **Example 16.3.46 (An Ontology).**

TBox (terminological Box)			ABox (assertional Box, data base)	
woman	=	person ⊓ has_Y	tony:person	Tony is a person
man	=	person ⊓ has_Y	tony:has_Y	Tony has a y-chrom

  This entails: tony:man (Tony is a man).

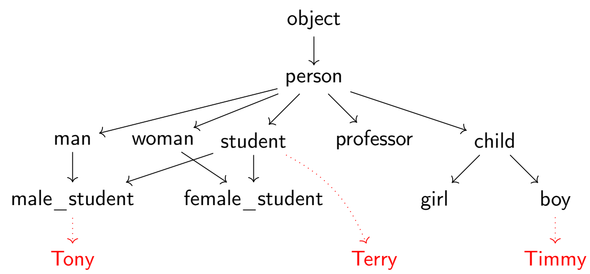
- **Problem:** Can we compute this?

If we combine classification with the instance test, then we get the full picture of how concepts and individuals relate to each other. We see that we get the full expressivity of semantic networks in $\mathcal{ALC}$.

Realization

- **Definition 16.3.47.** 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 16.3.48.** If tony:male_student is known, we do not need tony:man.

Michael Kohlhase: Artificial Intelligence 1 535 2025-02-06

Let us now get an intuition on what kinds of interactions between the various parts of an ontology.

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 16.3.49.**

property	example
internally inconsistent	tony:student, tony:$\overline{\text{student}}$
inconsistent with a TBox	TBox: student $\sqcap$ prof ABox: tony:student, tony:prof
implicit info that is not explicit	ABox: tony:$\forall$has_grad.genius tony has_grad mary $\models$ mary:genius
information that can be combined with TBox info	TBox: happy_prof $=$ prof $\sqcap$ $\forall$has_grad.genius ABox: tony:happy_prof, tony has_grad mary $\models$ mary:genius

Michael Kohlhase: Artificial Intelligence 1 536 2025-02-06

Again, we ask ourselves whether all of these are computable.
Fortunately, it is very simple to add assertions to $\mathcal{T}_{\mathcal{ALC}}$. In fact, we do not have to change anything, as the judgments used in the tableau are already of the form of ABox assertion.

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 16.3.50.**

Example: add mary:$\overline{\text{genius}}$ to determine $ABox, TBox \models$ mary:genius

TBox	happy_prof = prof ⊓ ∀has_grad.genius	tony:prof ⊓ ∀has_grad.genius	TBox
		tony has_grad mary	ABox
		mary:$\overline{\text{genius}}$	Query
		tony:prof	$\mathcal{T}_\sqcap$
ABox	tony:happy_prof tony has_grad mary	tony:∀has_grad.genius	$\mathcal{T}_\sqcap$
		mary:genius	$\mathcal{T}_\forall$
		⊥	$\mathcal{T}_\bot$

▷ **Note:** The instance test is the base for realization. (remember?)

▷ **Idea:** Extend to more complex ABox queries. (e.g. give me all instances of φ)

This completes our investigation of inference for $\mathcal{ALC}$. We summarize that $\mathcal{ALC}$ is a logic-based ontology language where the inference problems are all decidable/computable via $\mathcal{T}_{\mathcal{ALC}}$. But of course, while we have reached the expressivity of basic semantic networks, there are still things that we cannot express in $\mathcal{ALC}$, so we will try to extend $\mathcal{ALC}$ without losing decidability/computability.

16.4 Description Logics and the Semantic Web

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/27289`.

In this section we discuss how we can apply description logics in the real world, in particular, as a conceptual and algorithmic basis of the semantic web. That tries to transform the World Wide Web from a human-understandable web of multimedia documents into a "web of machine-understandable data". In this context, "machine-understandable" means that machines can draw inferences from data they have access to. Note that the discussion in this digression is not a full-blown introduction to RDF and OWL, we leave that to [SR14; Her+13a; Hit+12] and the respective W3C recommendations. Instead we introduce the ideas behind the mappings from a perspective of the description logics we have discussed above.

The most important component of the semantic web is a standardized language that can represent "data" about information on the Web in a machine-oriented way.

Resource Description Framework

▷ **Definition 16.4.1.** 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 16.4.2.** 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

Note that all these examples have in common that they are about "objects on the Web", which is an aspect we will come to now.

"Objects on the Web" are traditionally called "resources", rather than defining them by their intrinsic properties – which would be ambitious and prone to change – we take an external property to define them: everything that has a URI is a web resource. This has repercussions on the design of RDF.

Resources and URIs

- RDF describes resources with properties and property values.
- RDF uses Web identifiers (URIs) to identify resources.
- **Definition 16.4.3.** A resource is anything that can have a URI, such as `http://www.fau.de`.
- **Definition 16.4.4.** 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 16.4.5.** 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 16.4.6.** Statements: *[This slide]^{subj} has been [author]^{pred}ed by [Michael Kohlhase]^{obj}*

The crucial observation here is that if we map "subjects" and "objects" to "individuals", and "predicates" to "relations", the RDF triples are just relational ABox statements of description logics. As a consequence, the techniques we developed apply.

Note: Actually, a RDF graph is technically a labeled multigraph, which allows multiple edges between any two nodes (the resources) and where nodes and edges are labeled by URIs.

We now come to the concrete syntax of RDF. This is a relatively conventional XML syntax that combines RDF statements with a common subject into a single "description" of that resource.

XML Syntax for RDF

- RDF is a concrete XML vocabulary for writing statements
- **Example 16.4.7.** 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.

Michael Kohlhase: Artificial Intelligence 1 540 2025-02-06

Note that XML namespaces play a crucial role in using element to encode the predicate URIs. Recall that an element name is a qualified name that consists of a namespace URI and a proper element name (without a colon character). Concatenating them gives a URI in our example the predicate URI induced by the dc:creator element is `http://purl.org/dc/elements/1.1/creator`. Note that as URIs go RDF URIs do not have to be URLs, but this one is and it references (is redirected to) the relevant part of the Dublin Core elements specification [DCM12].

RDF was deliberately designed as a standoff markup format, where URIs are used to annotate web resources by pointing to them, so that it can be used to give information about web resources without having to change them. But this also creates maintenance problems, since web resources may change or be deleted without warning.

RDFa gives authors a way to embed RDF triples into web resources and make keeping RDF statements about them more in sync.

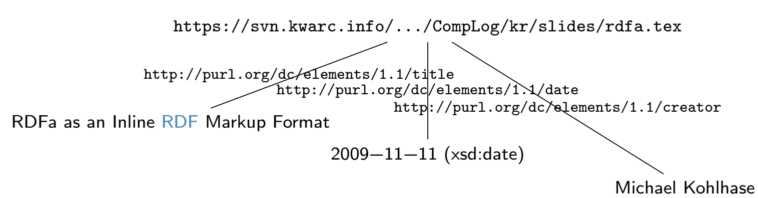
RDFa as an Inline RDF Markup Format

▷ **Problem:** RDF is a standoff markup format (annotate by URIs pointing into other files)

Definition 16.4.8. RDFa (RDF annotations) is a markup scheme for inline annotation (as XML attributes) of RDF triples.

▷ **Example 16.4.9.**

```
<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>
```



Michael Kohlhase: Artificial Intelligence 1 541 2025-02-06

In the example above, the about and property attributes are reserved by RDFa and specify the subject and predicate of the RDF statement. The object consists of the body of the element, unless otherwise specified e.g. by the content and datatype attributes for literals content.

Let us now come back to the fact that RDF is just an XML syntax for ABox statements.

RDF as an ABox Language for the Semantic Web

▷ **Idea:** RDF triples are ABox entries h R s or $h{:}\varphi$.

▷ **Example 16.4.10.** 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 $\mathcal{ALC}$)

In this situation, we want a standardized representation language for TBox information; OWL does just that: it standardizes a set of knowledge representation primitives and specifies a variety of concrete syntaxes for them. OWL is designed to be compatible with RDF, so that the two together can form an ontology language for the web.

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 16.4.11.** OWL (the ontology web language) is a language for encoding TBox information about RDF classes.

▷ **Example 16.4.12 (A concept definition for "Mother").** Mother=Woman ⊓ Parent is represented as

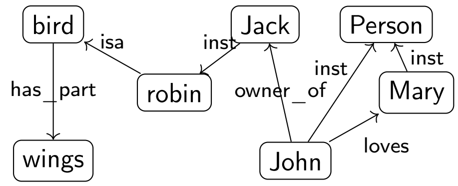
XML Syntax	Functional Syntax

<EquivalentClasses>
  <Class IRI="Mother"/>
  <ObjectIntersectionOf>
    <Class IRI="Woman"/>
    <Class IRI="Parent"/>
  </ObjectIntersectionOf>
</EquivalentClasses>
	EquivalentClasses(
	  :Mother
	  ObjectIntersectionOf(
	    :Woman
	    :Parent
	  )
	)

But there are also other syntaxes in regular use. We show the functional syntax which is inspired by the mathematical notation of relations.

Extended OWL Example in Functional Syntax

▷ **Example 16.4.13.** The semantic network from ?? 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*.

We have introduced the ideas behind using description logics as the basis of a "machine-oriented web of data". While the first OWL specification (2004) had three sublanguages "OWL Lite", "OWL DL" and "OWL Full", of which only the middle was based on description logics, with the OWL2 Recommendation from 2009, the foundation in description logics was nearly universally accepted.

The semantic web hype is by now nearly over, the technology has reached the "plateau of productivity" with many applications being pursued in academia and industry. We will not go into these, but briefly instroduce one of the tools that make this work.

SPARQL an RDF Query language

- ▷ **Definition 16.4.14.** 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 16.4.15.** A system is called a SPARQL endpoint, iff it answers SPARQL queries.
- ▷ **Example 16.4.16.** 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.
}
```

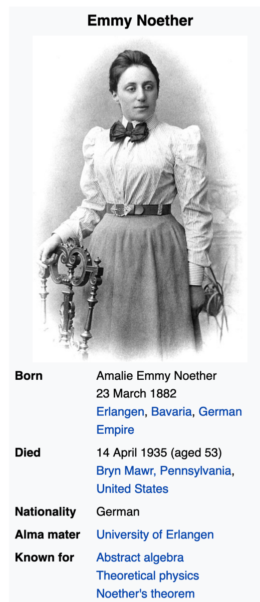

SPARQL end-points can be used to build interesting applications, if fed with the appropriate data. An interesting – and by now paradigmatic – example is the DBPedia project, which builds a large ontology by analyzing Wikipedia fact boxes. These are in a standard HTML form which can be analyzed e.g. by regular expressions, and their entries are essentially already in triple form: The subject is the Wikipedia page they are on, the predicate is the key, and the object is either the URI on the object value (if it carries a link) or the value itself.

SPARQL Applications: DBPedia

- ▷ **Typical Application:** DBPedia screen-scrapes Wikipedia fact boxes for RDF triples and uses SPARQL for querying the induced triplestore.
- ▷ **Example 16.4.17 (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
:Emilian_Dolha	:Romania	:Lech_Poznań	:Poland	43269
:Eusebio_Acasuzo	:Peru	:Club_Bolívar	:Bolivia	42000
:Faryd_Mondragón	:Colombia	:Real_Zaragoza	:Spain	34596
:Faryd_Mondragón	:Colombia	:Club_Atlético_Independiente	:Argentina	48069
:Federico_Vilar	:Argentina	:Club_Atlas	:Mexico	54500
:Fernando_Martinuzzi	:Argentina	:Real_Garcilaso	:Peru	45000
:Fábio_André_da_Silva	:Portugal	:Servette_FC	:Switzerland	30084
:Gerhard_Tremmel	:Germany	:FC_Red_Bull_Salzburg	:Austria	31000
:Gift_Muzadzi	:United_Kingdom	:Lech_Poznań	:Poland	43269
:Günay_Güvenç	:Germany	:Beşiktaş_J.K.	:Turkey	41903
:Hugo_Marques	:Portugal	:C.D._Primeiro_de_Agosto	:Angola	48500
:Héctor_Landazuri	:Colombia	:La_Paz_F.C.	:Bolivia	42000

We conclude our survey of the semantic web technology stack with the notion of a triplestore, which refers to the database component, which stores vast collections of ABox triples.

Triple Stores: the Semantic Web Databases

- ▷ **Definition 16.4.18.** 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 16.4.19.** 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 IV

Planning & Acting

This part covers the AI subfield of "planning", i.e. search-based problem solving with a structured representation language for environment state and actions — in planning, the focus is on the latter.

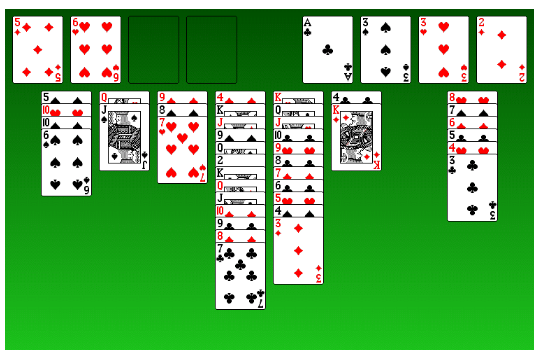
We first introduce the framework of planning (structured representation languages for problems and actions) and then present algorithms and complexity results. Finally, we lift some of the simplifying assumptions – deterministic, fully observable environments – we made in the previous parts of the course.

Chapter 17

Planning I: Framework

Reminder: Classical Search Problems

▷ **Example 17.0.1 (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 17.0.2.** Let Π be a search problem (see ??)

- ▷ The blackbox description of Π is an API providing functionality allowing to construct the state space: $\text{InitialState}()$, $\text{GoalTest}(s)$, ...

- ▷ "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

Before we go into the planning framework and its particular methods, let us see what we would do with the methods from **??** if we were to develop a "logic-based language" for describing states and actions. We will use the Wumpus world from **??** as a running example.

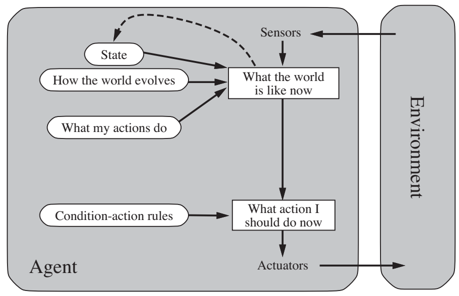
Fluents: Time-Dependent Knowledge in Planning

- ▷ **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 17.1.1.** If the agent moves and a cell with a draft at (a perceived breeze) is followed by one without.
- ▷ **Obvious Idea:** Make representations of percepts time-dependent
- ▷ **Example 17.1.2.** D^t for $t \in \mathbb{N}$ for PL^0 and $\mathrm{draft}(t)$ in PL^1 and $\mathrm{PL}^{\mathrm{nq}}$.
- ▷ **Definition 17.1.3.** We use the word fluent to refer an aspect of the world that changes, all others we call atemporal.

Let us recall the agent-based setting we were using for the inference procedures from **??**. We will elaborate this further in this section.

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)

Now that we have the notion of fluents to represent the percepts at a given time point, let us try to model how they influence the agent's world model.

Fluents: Modeling the Agent's Sensors

▷ **Idea:** Relate percept fluents to atemporal cell attributes.

▷ **Example 17.1.4.** E.g., if the agent perceives a draft at at time t, when it is in cell $[x, y]$, then there must be a breeze there:

$$\forall t, x, y.\mathrm{Ag@}(t, x, y) \Rightarrow (\mathrm{draft}(t) \Leftrightarrow \mathrm{breeze}(x, y))$$

▷ Axioms like these model the agent's sensors – here that they are totally reliable: there is a breeze, iff the agent feels a draft at.

▷ **Definition 17.1.5.** We call fluents that describe the agent's sensors sensor axioms.

▷ **Problem:** Where do fluents like $\mathrm{Ag@}(t, x, y)$ come from?

You may have noticed that for the sensor axioms we have only used first-order logic. There is a general story to tell here: If we have finite domains (as we do in the Wumpus cave) we can always "compile first-order logic into propositional logic"; if domains are infinite, we usually cannot.

We will develop this here before we go on with the Wumpus models.

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 ?? are best represented in first-order logic. In PL^0 and $\mathrm{PL}^{\mathrm{nq}}$ we would have to use concrete instances like $\mathrm{Ag@}(7,2,1) \Rightarrow (\mathrm{draft}(7) \Leftrightarrow \mathrm{breeze}(2,1))$ for all suitable t, x, and y.
- **Problem:** Unless we restrict ourselves to finite domains and an end time t_{end} we have infinitely many axioms. Even then, formalization in PL^0 and $\mathrm{PL}^{\mathrm{nq}}$ is very tedious.
- **Solution:** Formalize in first-order logic and then compile down:
  1. enumerate ranges of bound variables, instantiate body, ($\rightsquigarrow \mathrm{PL}^{\mathrm{nq}}$)
  2. translate $\mathrm{PL}^{\mathrm{nq}}$ atoms to propositional variables. ($\rightsquigarrow \mathrm{PL}^0$)
- **In Practice:** The choice of domain, end time, and logic is up to agent designer, weighing expressivity vs. efficiency of inference.
- WLOG: We will use PL^1 in the following. (easier to read)

We now continue to our logic-based agent models: Now we focus on effect axioms to model the effects of an agent's actions.

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 17.1.6.** The action of "going forward" at time t is captured by the fluent $\mathrm{forw}(t)$.
- **Definition 17.1.7.** Effect axioms describe how the environment changes under an agent's actions.
- **Example 17.1.8.** 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.

Unfortunately, the percept fluents, sensor axioms, and effect axioms are not enough, as we will show in ??. We will see that this is a more general problem – the famous frame problem that needs to be considered whenever we deal with change in environments.

Frames and Frame Axioms

- **Problem:** Effect axioms are not enough.
- **Example 17.1.9.** Say that the agent has an arrow at time 0, and then moves forward at into $[2,1]$, perceives a glitter, and knows that the Wumpus is ahead.

  To evaluate the action $\text{shoot}(1)$ the corresponding effect axiom needs to know $\text{havarrow}(1)$, but cannot prove it from $\text{havarrow}(0)$.

  Problem: The information of having an arrow has been lost in the move forward.
- **Definition 17.1.10.** 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.

We conclude our discussion with a relatively complete implementation of a logic-based Wumpus agent, building on the schema from slide 552.

A Hybrid Agent for the Wumpus World

- **Example 17.1.11 (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.

Now look at the "special code" we have promised.

A Hybrid Agent: Custom Action Selection

▷ **Example 17.1.12 (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.

And finally the route planning part of the code. This is essentially just A^* search.

A Hybrid Agent: Custom Action Selection

▷ **Example 17.1.13 (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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26892`.

How does a planning language describe a problem?

▷ **Definition 17.2.1.** 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 $Eq(.,.)$.)
- ▷ initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)
- ▷ goal states G (vs. a goal test). (E.g.: $Eq(x,2)$.)
- ▷ set A of actions in terms of preconditions and effects (vs. functions returning applicable actions and successor states). (E.g.: "increment x: pre $Eq(x,1)$, iff $Eq(x \wedge 2) \wedge \neg Eq(x,1)$".)

A logical description of all of these is called a planning task.

▷ **Definition 17.2.2.** 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.

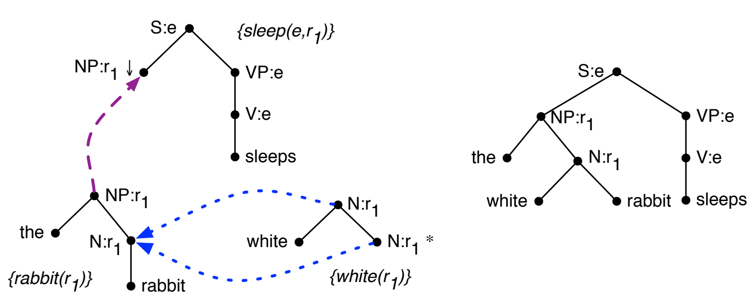
Planning Language Overview

▷ **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



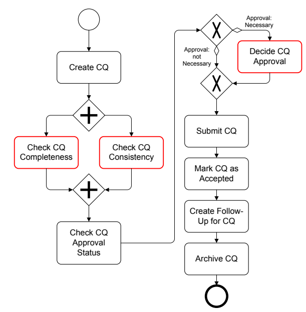
▷ **Input**: Tree-adjoining grammar, intended meaning.

▷ **Output**: Sentence expressing that meaning.

FAU Michael Kohlhase: Artificial Intelligence 1 562 2025-02-06

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

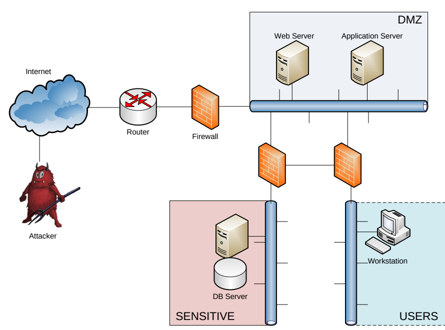


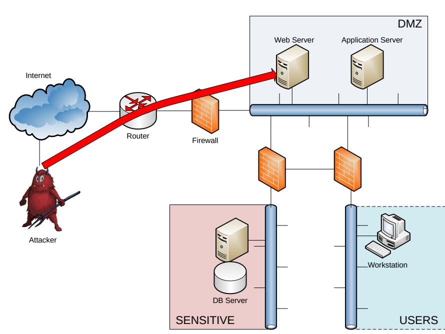
▷ **Input**: model of behavior of activities on business objects, process endpoint.

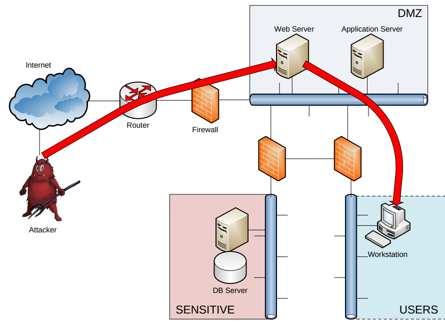
▷ **Output**: Process template leading to this point.

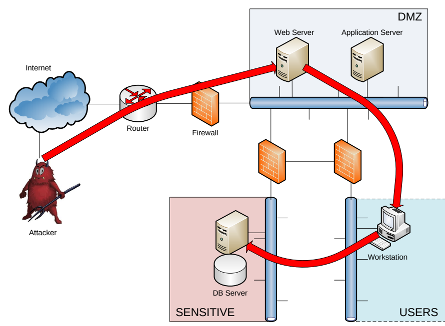
FAU Michael Kohlhase: Artificial Intelligence 1 563 2025-02-06

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 . . .

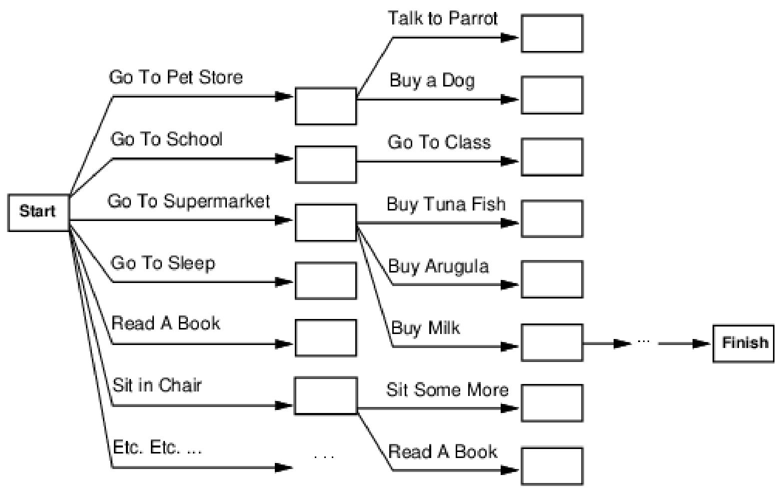
  ▷ Trade-off between "automatic and general" vs. "manual work but efficient".

▷ **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

Search vs. planning (cont.)

▷ Planning systems do the following:

1. open up action and goal representation to allow selection
2. divide-and-conquer by subgoaling

▷ relax requirement for sequential construction of solutions

	Search	**Planning**
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

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^*

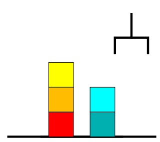
- order $frontier$ by $g + h$ instead of h (line 4)
- insert $g(n') + h(n')$ instead of $h(n')$ to $frontier$ (last line)

▷ 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 17.2.3.**



- 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 17.2.4.** *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 17.2.5.** 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 17.2.6.** 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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26894`.

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 $move(A, B)$; $\rightsquigarrow open = \{at(A)\}$ ...
  - **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. $at(B)_1{}^{F} \vee move(A,B)_0{}^{T}$; unit clauses to encode initial state $at(A)_0{}^{T}$, $at(B)_0{}^{T}$; unit clauses to encode goal $at(B)_1{}^{T}$.
- **Popular when**: 1996 – today.
- **Approach**: *From planning task description, generate propositional CNF formula φ_k that is satisfiable iff there exists a plan with k steps; use a SAT solver on φ_k, for different values of k.*
- **Keywords/cites**: [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 17.3.1.** 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:**

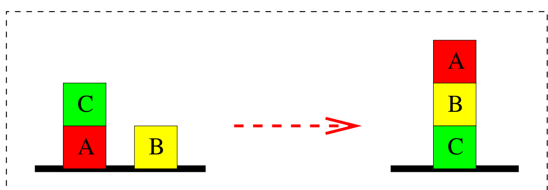
- Standard representation language: PDDL [McD+98; FL03; HE05; Ger+09]
- Problem Corpus: ≈ 50 **domains**, $\gg 1000$ **instances**, 74 (!!) planners in 2011

International Planning Competition

- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?
- **Answer:** reserved for the plenary sessions $\rightsquigarrow$ be there!
- **Generally:** reserved for the plenary sessions $\rightsquigarrow$ be there!

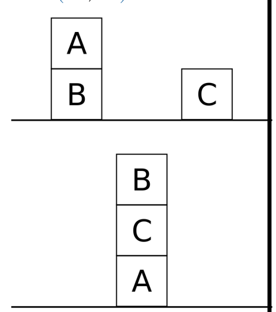
Planning History, p.s.: Planning is Non-Trivial!

- **Example 17.3.2.** 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:

- 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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26896`.

STRIPS Planning

- **Definition 17.4.1.** 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.

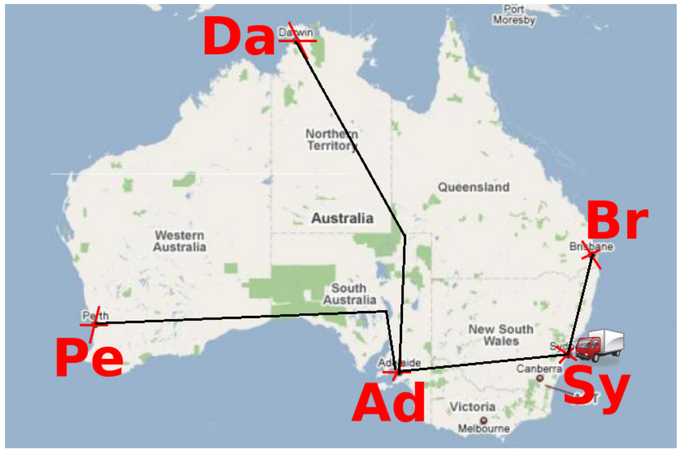
STRIPS Planning: Syntax

- ▷ **Definition 17.4.2.** 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

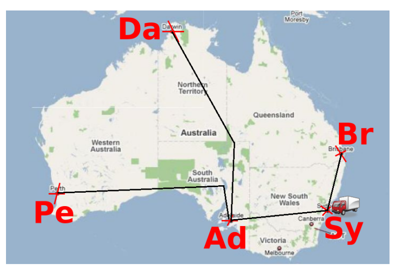
- ▷ **Example 17.4.3 (Salesman Travelling in Australia).**



Strictly speaking, this is not actually a **TSP** problem instance; simplified/adapted for illustration.

STRIPS Encoding of "TSP"

▷ **Example 17.4.4 (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 17.4.5.** 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 $\{s \xrightarrow{a} \mathrm{apply}(s,a) \mid \mathrm{pre}_a \subseteq s\}$.
  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 $S_G = \{s \in S_P \mid G \subseteq s\}$ 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.

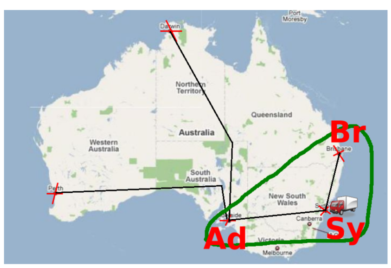
▷ **Definition 17.4.6.** For a plan $a = \langle a_1, \ldots, a_n\rangle$, we define

$$\mathrm{apply}(s,a) := \mathrm{apply}(\ldots\mathrm{apply}(\mathrm{apply}(s,a_1),a_2)\ldots,a_n)$$

if each a_i is applicable in the respective state; else, $\mathrm{apply}(s,a)$ is undefined.

STRIPS Encoding of Simplified TSP

▷ **Example 17.4.7 (Simplified traveling salesman problem in Australia).**

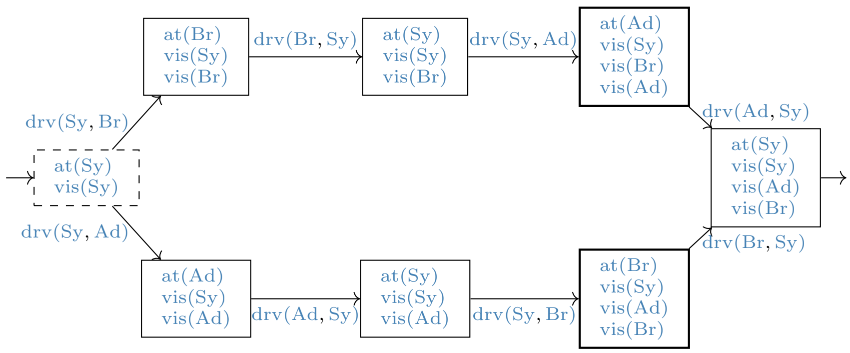


Let TSP_ be the STRIPS task, $\langle P, A, I, G\rangle$, where

- ▷ Facts P: $\{\mathrm{at}(x), \mathrm{vis}(x) \mid x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}\}\}$.
- ▷ Initial state state I: $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
- ▷ Goal state G: $\{\mathrm{vis}(x) \mid x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}\}\}$ (note: no $\mathrm{at}(\mathrm{Sy})$)
- ▷ 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 ?? 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)

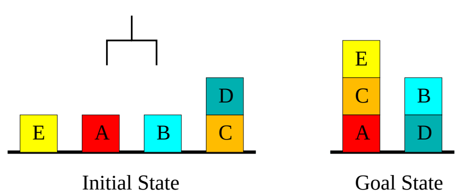
▷ **Question:** Is the graph above actually the state space induced by ?

▷ **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 17.4.8.** 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 17.4.9.**



- ▷ 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: $\mathrm{stack}(x,y)$, $\mathrm{unstack}(x,y)$, $\mathrm{putdown}(x)$, $\mathrm{pickup}(x)$.
- ▷ $\mathrm{stack}(x,y)$?

pre : $\{\text{holding}(x), \text{clear}(y)\}$
add : $\{\text{on}(x,y), \text{armEmpty}, clear x\}$
del : $\{\text{holding}(x), \text{clear}(y)\}$.

STRIPS for the Blocksworld

▷ **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\text{pickup}(x)$ action schema?

(A)	$\{\text{onTable}(x), \text{clear}(x), \text{armEmpty}\}$ $\{\text{holding}(x)\}$ $\{\text{onTable}(x)\}$	(B)	$\{\text{onTable}(x), \text{clear}(x), \text{armEmpty}\}$ $\{\text{holding}(x)\}$ $\{\text{armEmpty}\}$
(C)	$\{\text{onTable}(x), \text{clear}(x), \text{armEmpty}\}$ $\{\text{holding}(x)\}$ $\{\text{onTable}(x), \text{armEmpty}, \text{clear}(x)\}$	(D)	$\{\text{onTable}(x), \text{clear}(x), \text{armEmpty}\}$ $\{\text{holding}(x)\}$ $\{\text{onTable}(x), \text{armEmpty}\}$

Recall: an actions a represented by a tuple $\langle \text{pre}_a, \text{add}_a, \text{del}_a \rangle$ of lists of facts.

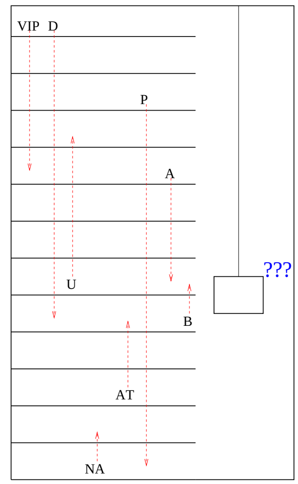
▷ **Hint:** The only differences between them are the delete lists

▷ **Answer:** reserved for the plenary sessions $\rightsquigarrow$ be there!

The next example for a planning task is not obvious at first sight, but has been quite influential, showing that many industry problems can be specified declaratively by formalizing the domain and the particular planning tasks in PDDL and then using off-the-shelf planners to solve them. [KS00] reports that this has significantly reduced labor costs and increased maintainability of the implementation.

Miconic-10: A Real-World Example

▷ **Example 17.4.10.** 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

Michael Kohlhase: Artificial Intelligence 1 589 2025-02-06

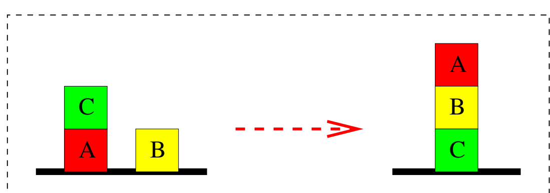
17.5 Partial Order Planning

In this section we introduce a new and different planning algorithm: partial order planning that works on several subgoals independently without having to specify in which order they will be pursued and later combines them into a global plan. **A Video Nugget** covering this section can be found at `https://fau.tv/clip/id/28843`.

To fortify our intuitions about partial order planning let us have another look at the Sussman anomaly, where pursuing two subgoals independently and then reconciling them is a prerequisite.

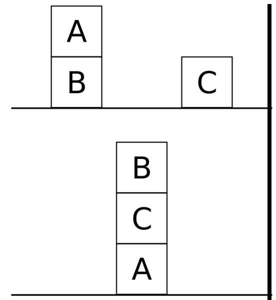
Planning History, p.s.: Planning is Non-Trivial!

▷ **Example 17.5.1.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\text{on}(A, B)$ and $\text{on}(B, C)$ fail:

- If we pursue $\mathrm{on}(A,B)$ 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.



Before we go into the details, let us try to understand the main ideas of partial order planning.

Partial Order Planning

- **Definition 17.5.2.** 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.
  - 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.

We now make the ideas discussed above concrete by giving a mathematical formulation. It is advantageous to cast a partially ordered plan as a labeled DAG rather than a partial ordering since it draws the attention to the difference between actions and steps.

Partially Ordered Plans

- **Definition 17.5.3.** 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 17.5.4.** 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 17.5.5.** A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it.
- **Definition 17.5.6.** A partially ordered plan Π is called complete iff every precondition is achieved.
- **Definition 17.5.7.** Partial order planning is the process of computing complete and acyclic partially ordered plans for a given planning task.

Michael Kohlhase: Artificial Intelligence 1 592 2025-02-06

A Notation for STRIPS Actions

- **Definition 17.5.8 (Notation).** In diagrams, we often write STRIPS actions into boxes with preconditions above and effects below.
- **Example 17.5.9.**
  - Actions: $Buy(x)$
  - Preconditions: $At(p)$, $Sells(p,x)$
  - Effects: $Have(x)$

$At(p)\ Sells(p,x)$

$Buy(x)$

$Have(x)$

- **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.

Michael Kohlhase: Artificial Intelligence 1 593 2025-02-06

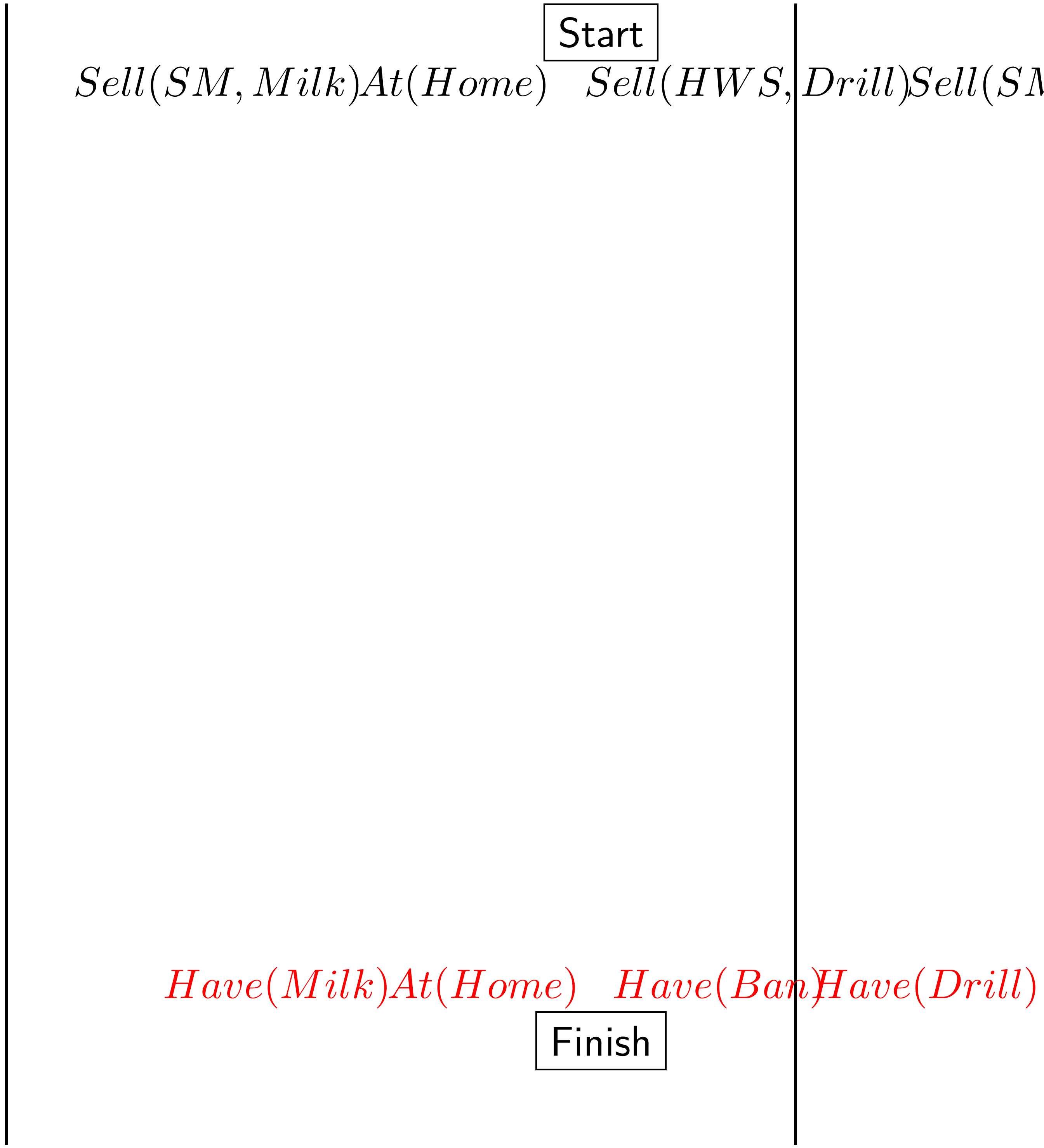
Planning Process

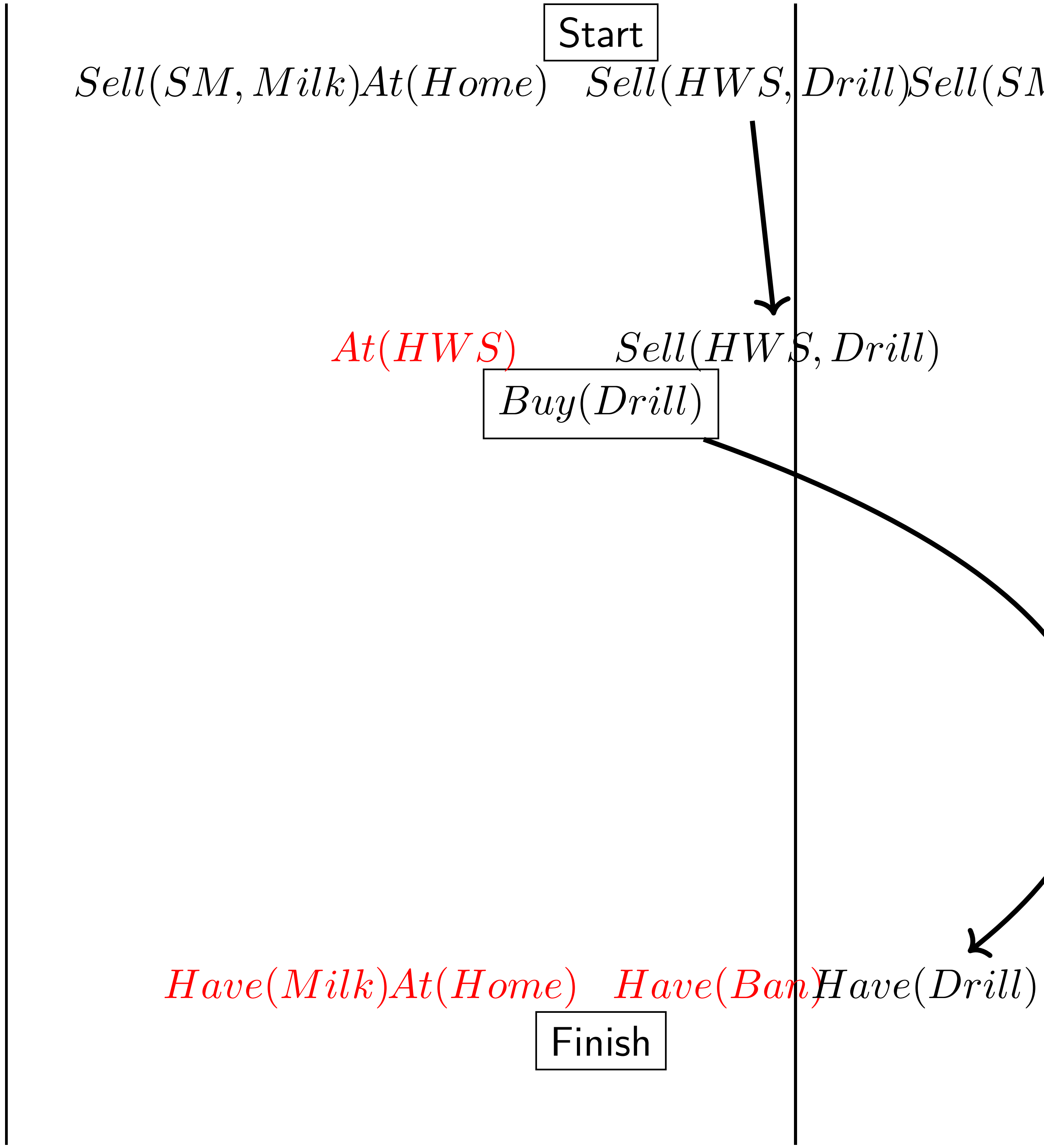
- **Definition 17.5.10.** Partial order planning is search in the space of partial plans via the following operations:
  - add link from an existing action to an open precondition,
  - add step (an action with links to other steps) to fulfil an open precondition,
  - order one step wrt. another (by adding temporal constraints) to remove possible conflicts.

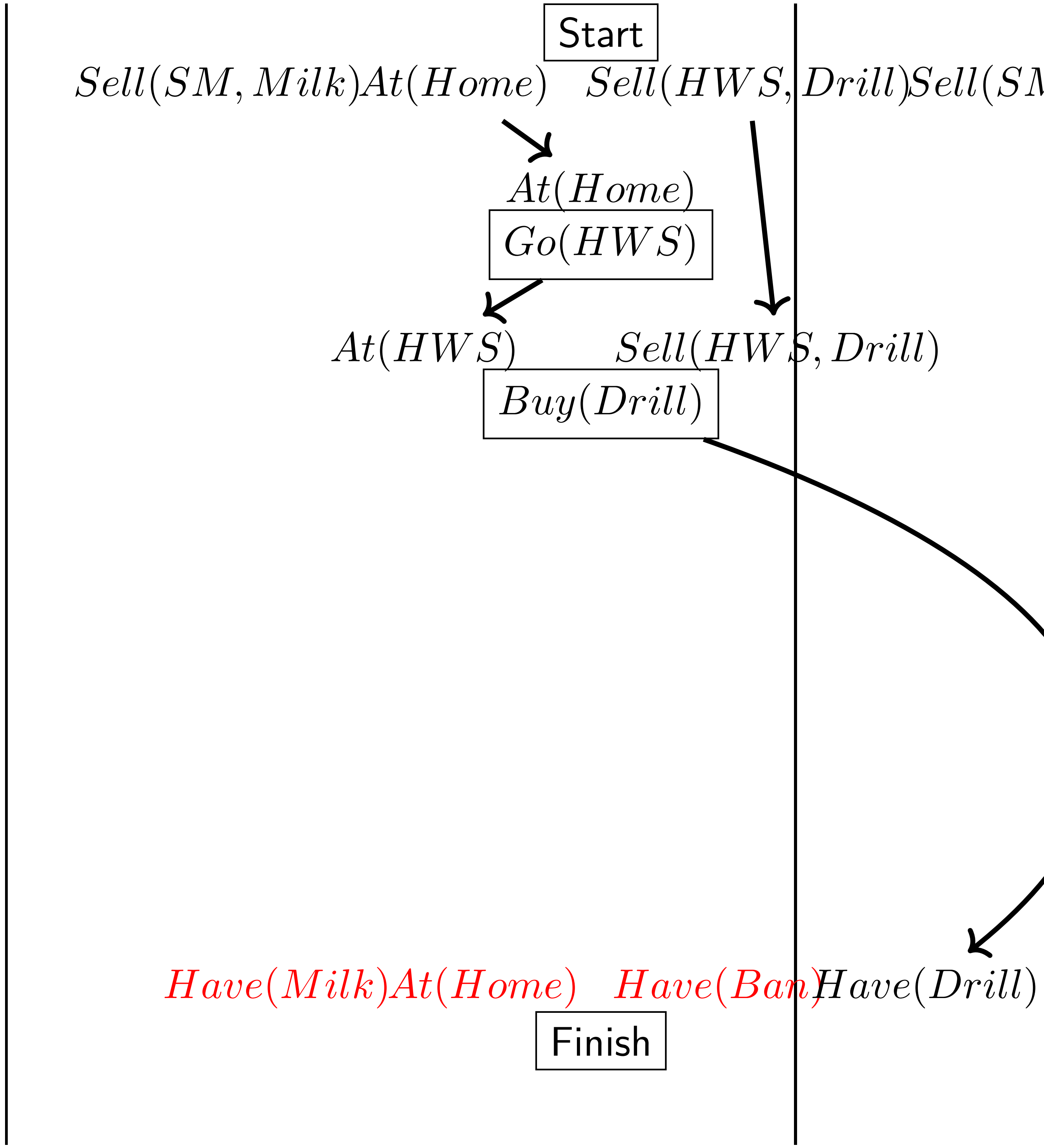
▷ **Idea:** Gradually move from incomplete/vague plans to complete, correct plans. backtrack if an open condition is unachievable or if a conflict is unresolvable.

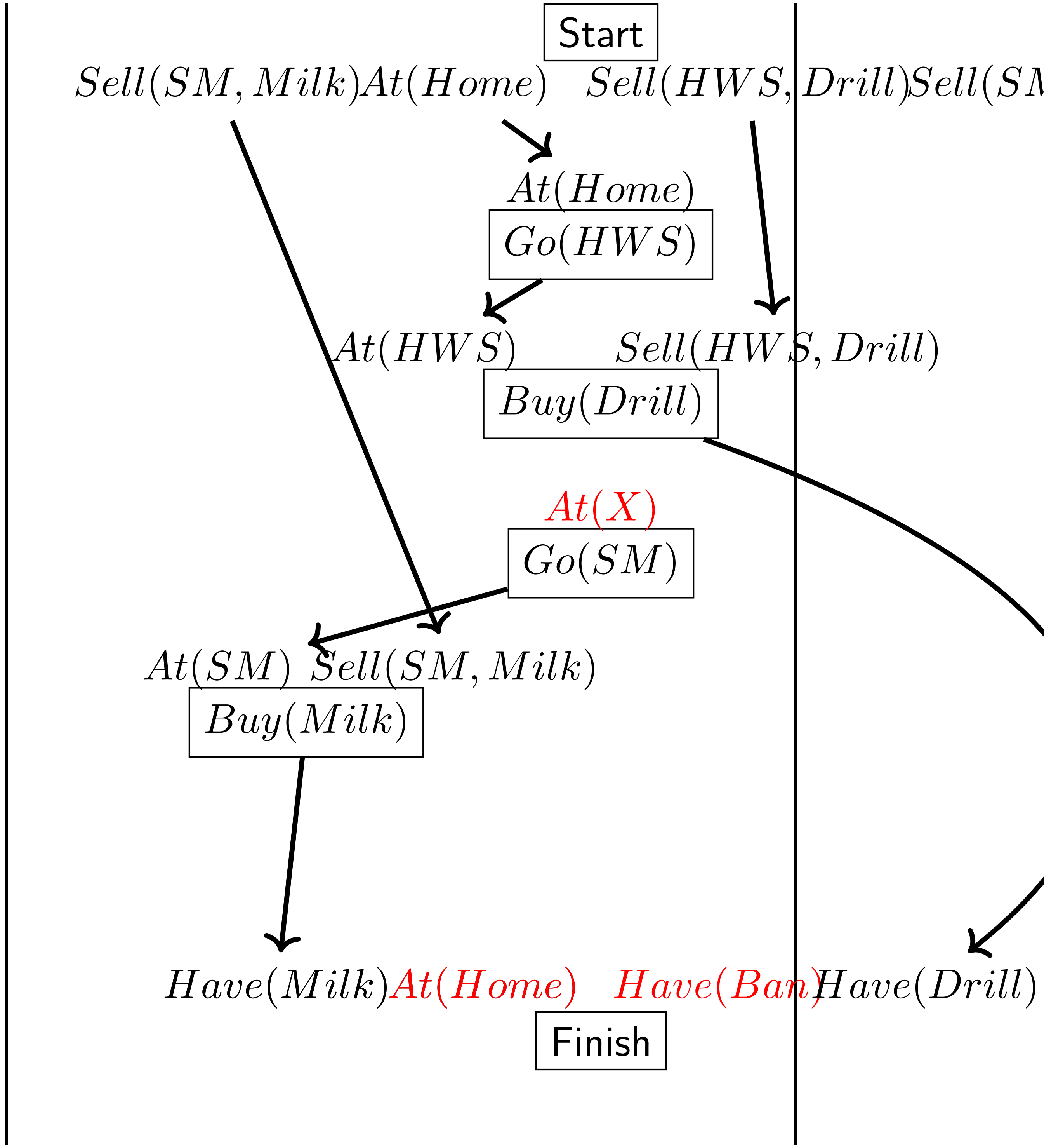
Example: Shopping for Bananas, Milk, and a Cordless Drill

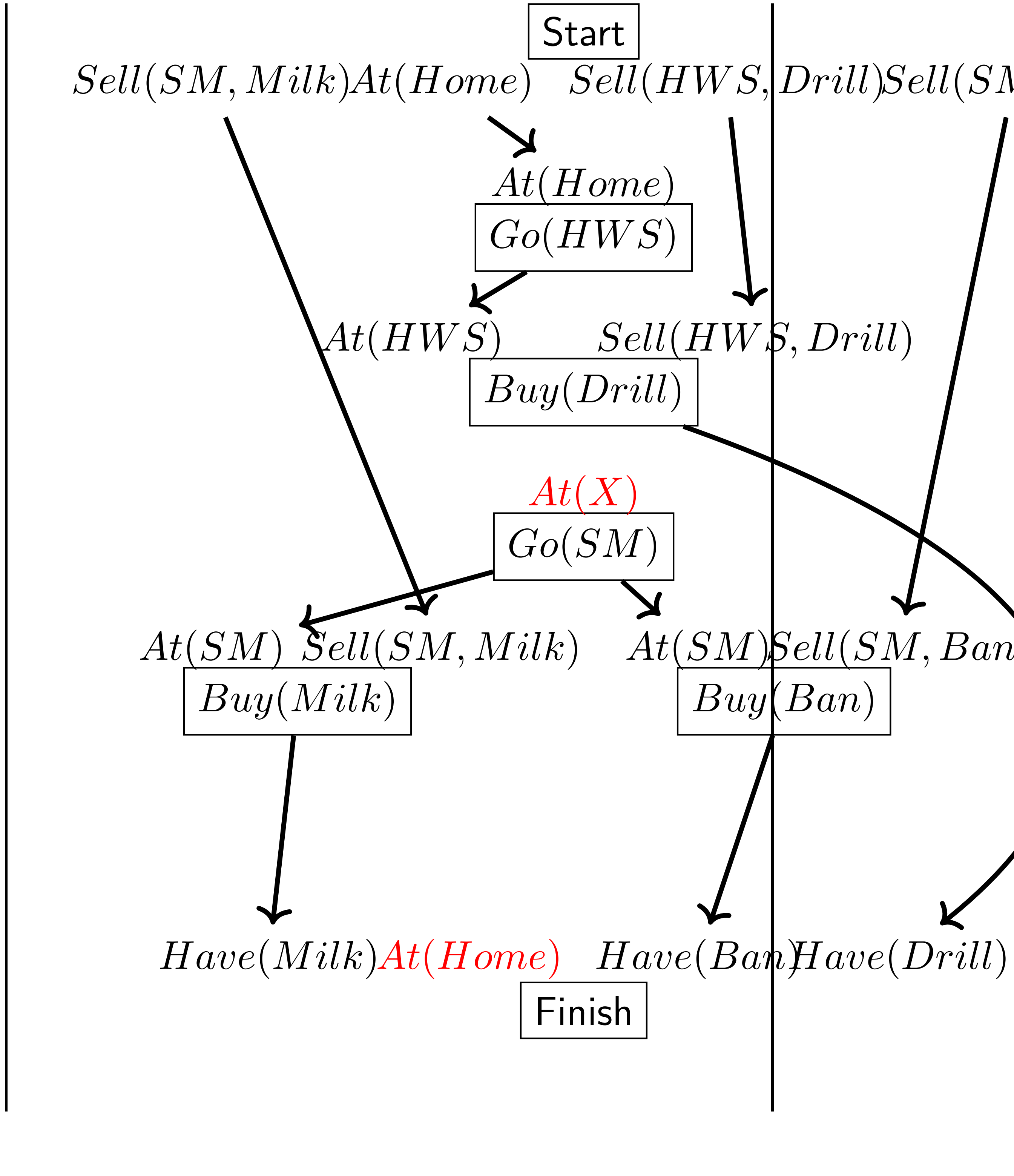
▷ **Example 17.5.11.**

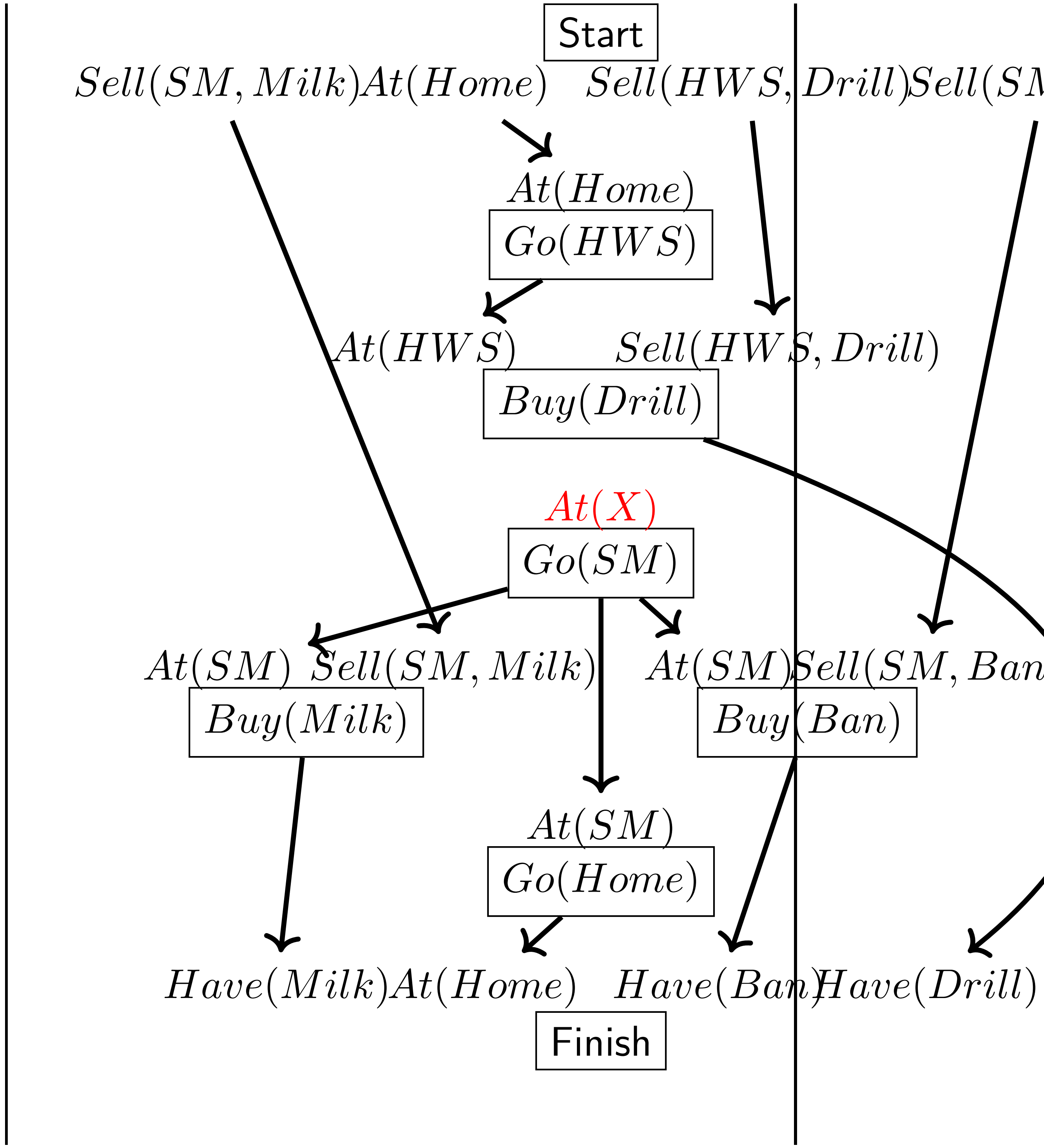

Start
Sell(SM, Milk)
At(Home)
Sell(HWS, Drill)
Have(Milk)
At(Home)
Have(Ban
Have(Drill)
Finish

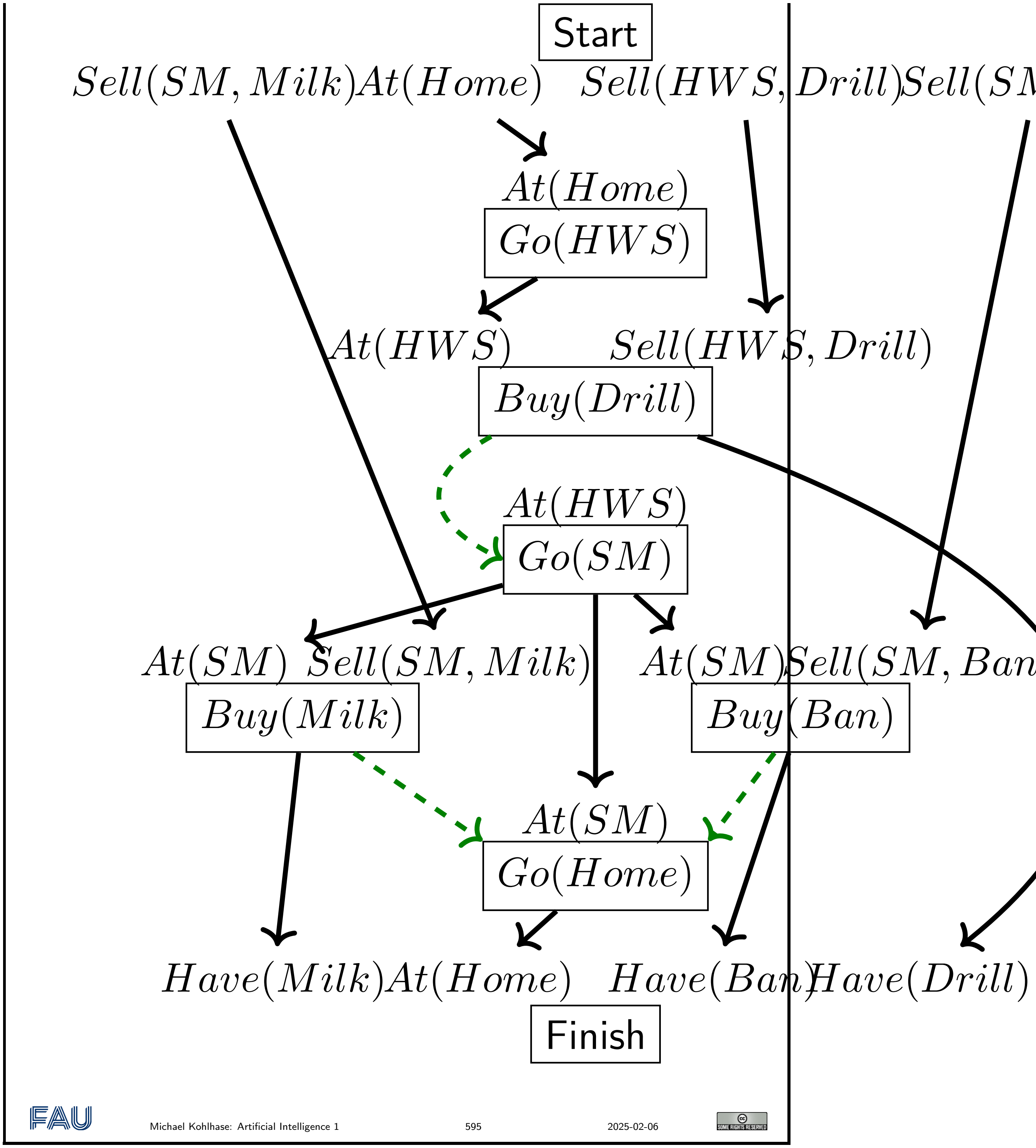

Start
Sell(SM, Milk)
At(Home)
Sell(HWS, Drill)
Sell(SM
At(HWS)
Sell(HWS, Drill)
Buy(Drill)
Have(Milk)
At(Home)
Have(Ban
Have(Drill)
Finish


Start
Sell(SM, Milk) At(Home) Sell(HWS, Drill) Sell(SM
At(Home)
Go(HWS)
At(HWS) Sell(HWS, Drill)
Buy(Drill)
Have(Milk) At(Home) Have(Ban) Have(Drill)
Finish


Start
Sell(SM, Milk)
At(Home)
Sell(HWS, Drill)
At(Home)
Go(HWS)
At(HWS)
Sell(HWS, Drill)
Buy(Drill)
At(X)
Go(SM)
At(SM)
Sell(SM, Milk)
Buy(Milk)
Have(Milk)
At(Home)
Have(Drill)
Finish


Start
Sell(SM, Milk)At(Home)
Sell(HWS, Drill)Sell(SM
At(Home)
Go(HWS)
At(HWS)
Sell(HWS, Drill)
Buy(Drill)
At(X)
Go(SM)
At(SM)
Sell(SM, Milk)
At(SM)Sell(SM, Ban
Buy(Milk)
Buy(Ban)
Have(Milk)At(Home)
Have(Ban)Have(Drill)
Finish


Start
Sell(SM, Milk)
At(Home)
Sell(HWS, Drill)
At(Home)
Go(HWS)
At(HWS)
Sell(HWS, Drill)
Buy(Drill)
At(X)
Go(SM)
At(SM)
Sell(SM, Milk)
Buy(Milk)
At(SM)
Buy(Ban)
At(SM)
Go(Home)
Have(Milk)
At(Home)
Have(Drill)
Finish



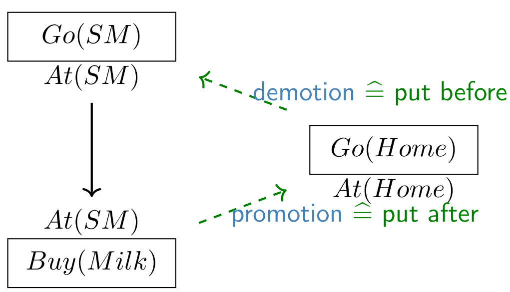
Here we show a successful search for a partially ordered plan. We start out by initializing the plan by with the respective start and finish steps. Then we consecutively add steps to fulfill the open preconditions – marked in red – starting with those of the finish step.

In the end we add three temporal constraints that complete the partially ordered plan.

The search process for the links and steps is relatively plausible and standard in this example, but we do not have any idea where the temporal constraints should systematically come from. We look at this next.

Clobbering and Promotion/Demotion

- **Definition 17.5.12.** 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 17.5.13.** 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 17.5.14.** $Go(Home)$ clobbers $At(Supermarket)$:



POP algorithm sketch

- **Definition 17.5.15.** 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 17.5.16.** The missing parts for the POP algorithm.

function Choose—Operator (plan, operators, S_{need}, c)
  choose a step S_{add} from operators or Steps(plan) that has c as an effect
  if there is no such step **then fail**
  add the causal—link $S_{add} \xrightarrow{c} S_{need}$ **to** Links(plan)
  add the temporal—constraint $S_{add} \prec S_{need}$ **to** Orderings(plan)
  if S_{add} is a newly added \step from operators **then**
    add S_{add} **to** Steps(plan)
    add $Start \prec S_{add} \prec Finish$ **to** Orderings(plan)

function Resolve—Threats (plan)
  for each S_{threat} that threatens a causal—link $S_i \xrightarrow{c} S_j$ **in** Links(plan) **do**
    choose either
      demotion: Add $S_{threat} \prec S_i$ **to** Orderings(plan)
      promotion: Add $S_j \prec S_{threat}$ **to** Orderings(plan)
    if not Consistent(plan) **then fail**

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.

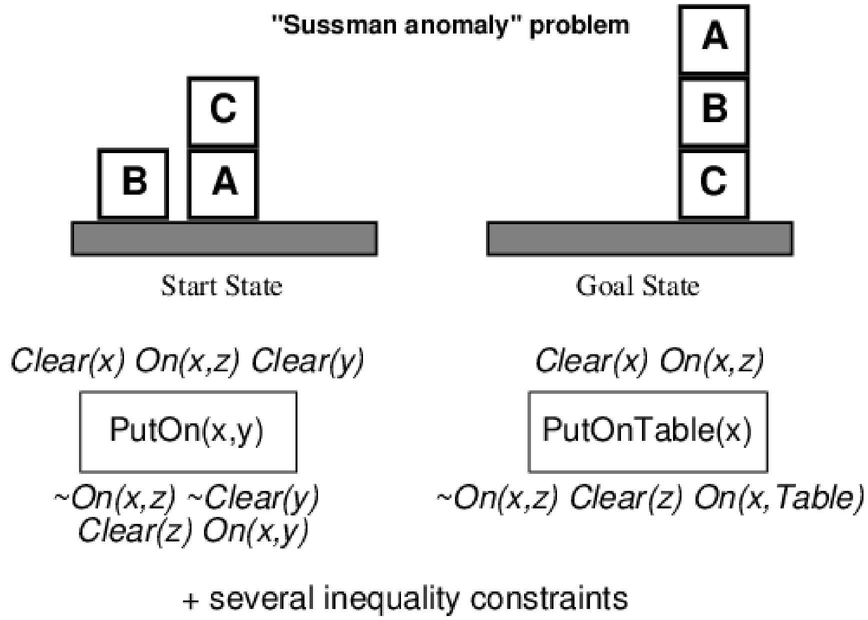
▷ **Observation 17.5.17.** *POP is sound, complete, and systematic i.e. no repetition*

▷ 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 17.5.18.** Solving the Sussman anomaly

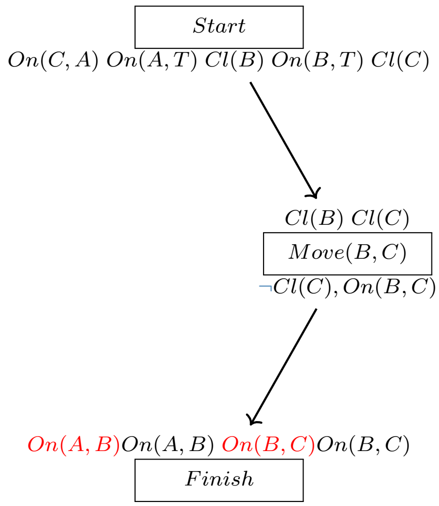
Start

$On(C,A)$ $On(A,T)$ $Cl(B)$ $On(B,T)$ $Cl(C)$

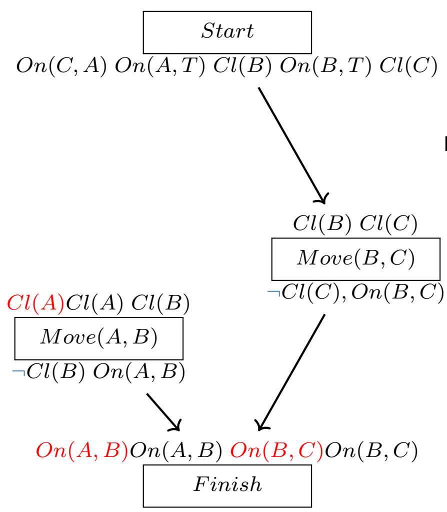
Initializing the partial order plan with with Start and Finish.

$On(A,B)On(A,B)$ $On(B,C)On(B,C)$

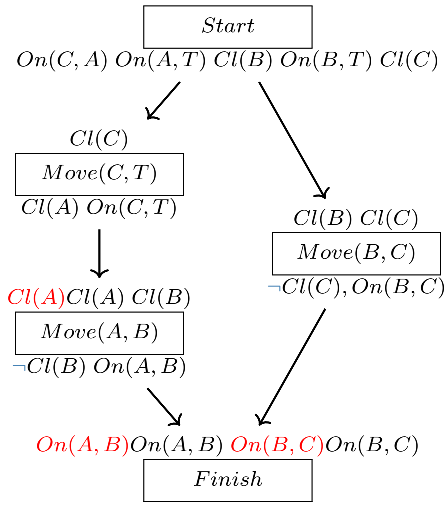
Finish



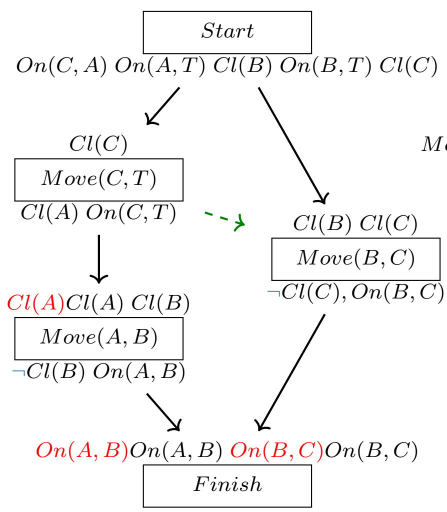
Refining for the subgoal $On(B, C)$.



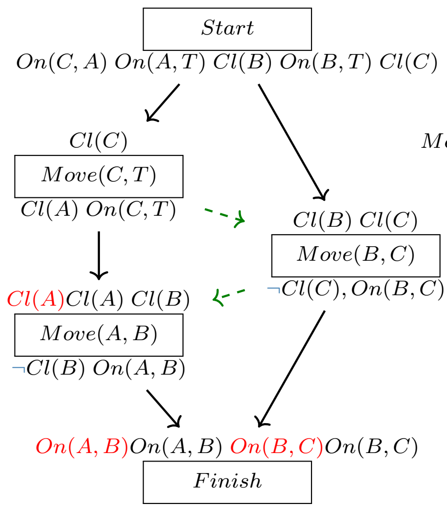
Refining for the subgoal $ON(A, C)$.



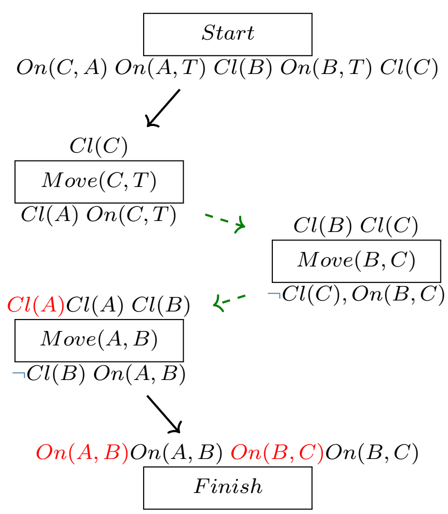
Refining for the subgoal $Cl(A)$.



$Move(A, B)$ clobbers $Cl(B)$ $\rightsquigarrow$ demote.



$Move(B, C)$ clobbers $Cl(C)$ $\rightsquigarrow$ demote.



A totally ordered plan.

17.6 The PDDL Language

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26897`.

PDDL: Planning Domain Description Language

▷ **Definition 17.6.1.** The Planning Domain Description Language (PDDL) is a standardized representation language for planning benchmarks in various extensions of the STRIPS formalism.

▷ **Definition 17.6.2.** 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.

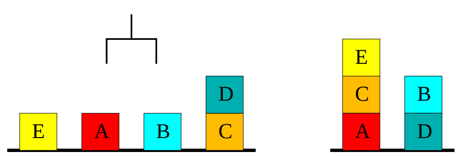
▷ **Definition 17.6.3.** A PDDL planning task comes in two pieces

- ▷ The problem file gives the objects, the initial state, and the goal state.
- ▷ The domain file gives the predicates and the actions.

History and Versions:

- Used in the International Planning Competition (IPC).
- 1998: PDDL [McD+98].
- 2000: "PDDL subset for the 2000 competition" [Bac00].
- 2002: PDDL2.1, Levels 1-3 [FL03].
- 2004: PDDL2.2 [HE05].
- 2006: PDDL3 [Ger+09].

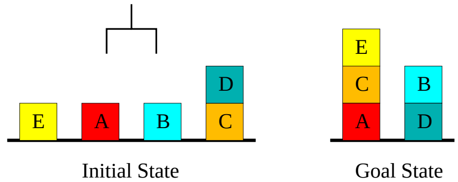
The Blocksworld in PDDL: Domain File



Initial State   Goal State

```
(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))))
```

Michael Kohlhase: Artificial Intelligence 1 604 2025-02-06

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)))))
)
```

Michael Kohlhase: Artificial Intelligence 1 605 2025-02-06

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:** reserved for the plenary sessions ⇝ be there!

17.7 Conclusion

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26900`.

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.

Suggested Reading:

- Chapters 10: *Classical Planning* and 11: *Planning and Acting in the Real World* in [RN09].
  - Although the book is named "*A Modern Approach*", the planning section was written long before the IPC was even dreamt of, before PDDL was conceived, and several years before heuristic search hit the scene. As such, what we have right now is the attempt of two outsiders trying in vain to catch up with the dramatic changes in planning since 1995.
  - Chapter 10 is Ok as a background read. Some issues are, imho, misrepresented, and it's far from being an up-to-date account. But it's Ok to get some additional intuitions in words different from my own.
  - Chapter 11 is useful in our context here because we don't cover any of it. If you're interested in extended/alternative planning paradigms, do read it.
- A good source for modern information (some of which we covered in the course) is Jörg Hoffmann's *Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)* [Hof11] which is available online at `http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf`

Chapter 18

Planning II: Algorithms

18.1 Introduction

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26901`.

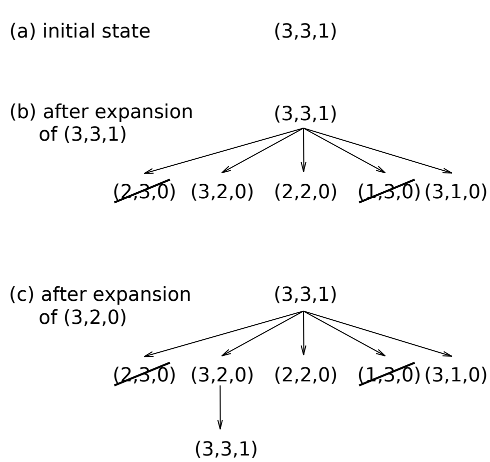
Reminder: Our Agenda for This Topic

- ▷ **??**: 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.

Michael Kohlhase: Artificial Intelligence 1 608 2025-02-06

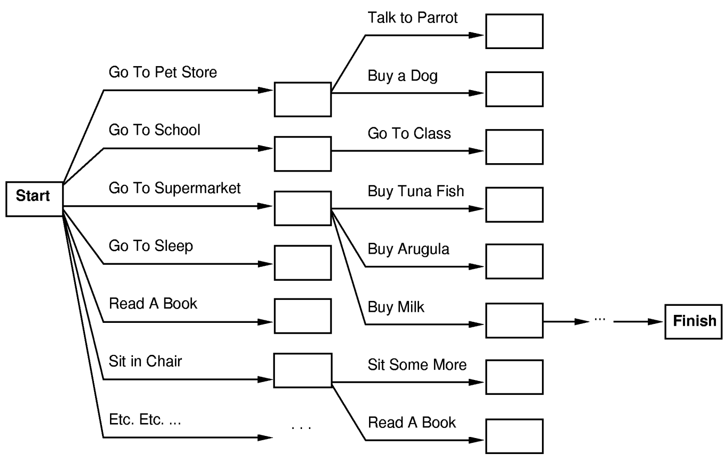
Reminder: Search

- ▷ Starting at initial state, produce all successor states step by step:



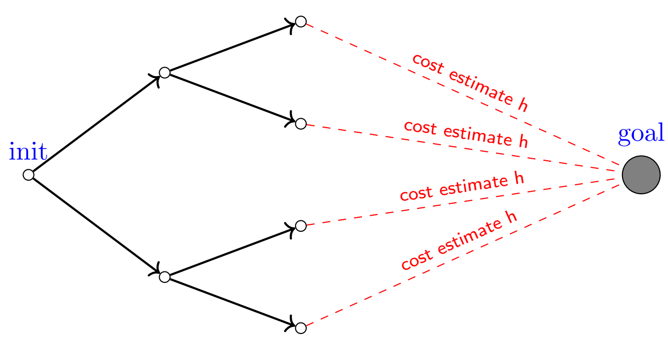
In planning, this is referred to as forward search, or forward state-space search.

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 18.1.1.** 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 18.1.2.** Let Π be a STRIPS task with states S. The perfect heuristic h^* assigns every $s \in S$ the length of a shortest path from s to a goal state, or ∞ if no such path exists. A heuristic h for Π is admissible if, for all $s \in S$, we have $h(s) \leq h^*(s)$.
- Exactly like our definition from ??, except for path *length* instead of path *cost* (cf. above).
- In all cases, we attempt to approximate $h^*(s)$, the length of an optimal plan for s. Some algorithms guarantee to lower bound $h^*(s)$.

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

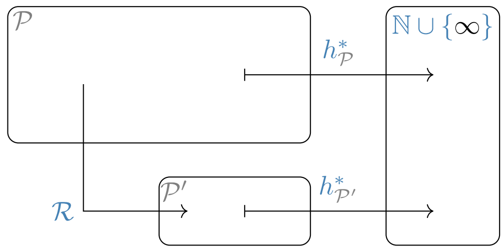
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26902`.
We will now instantiate our general knowledge about heuristic search to the planning domain. As always, the main problem is to find good heuristics. We will follow the intuitions of our discussion in ?? and consider full solutions to relaxed problems as a source for heuristics.

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 18.2.1 (The General Case).**



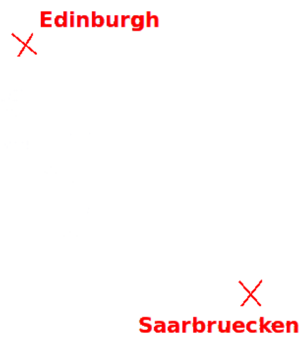
1. You have a class $\mathcal{P}$ of problems, whose perfect heuristic $h^*_{\mathcal{P}}$ you wish to estimate.
2. You define a class $\mathcal{P}'$ of *simpler problems*, whose perfect heuristic $h^*_{\mathcal{P}'}$ can be used to estimate $h^*_{\mathcal{P}}$.
3. You define a transformation – the relaxation 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 18.2.2.** For planning tasks, we speak of relaxed planning.

Reminder: Heuristic Functions from Relaxed Problems

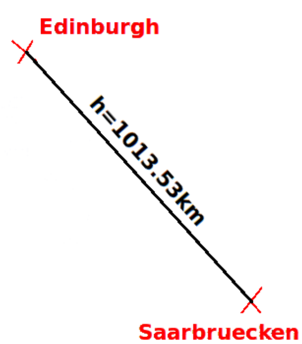
▷ Problem Π: Find a route from Saarbrücken to Edinburgh.

Reminder: Heuristic Functions from Relaxed Problems



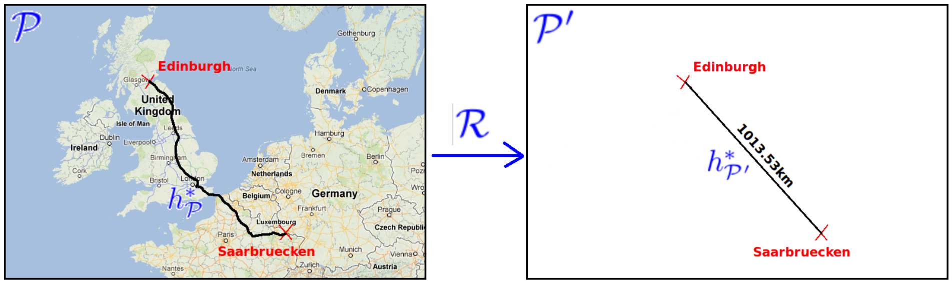
▷ Relaxed Problem Π': Throw away the map.

Reminder: Heuristic Functions from Relaxed Problems



▷ Heuristic function h: Straight line distance.

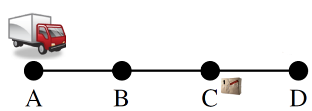
Relaxation in Route-Finding



- **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 18.2.3 (Logistics).**

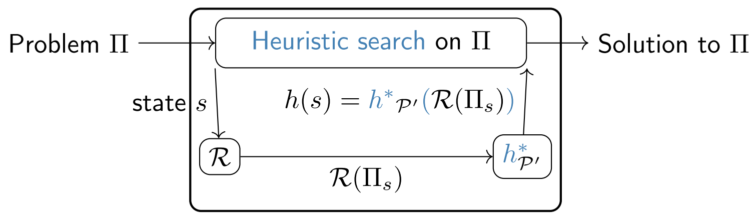


  - facts P: $\{\text{truck}(x)\,|\,x \in \{A,B,C,D\}\} \cup \{\text{pack}(x)\,|\,x \in \{A,B,C,D,T\}\}$.
  - initial state I: $\{\text{truck}(A), \text{pack}(C)\}$.
  - goal state G: $\{\text{truck}(A), \text{pack}(D)\}$.
  - actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
    - $\text{drive}(x,y)$, where x and y have a road: "$\text{truck}(x) \Rightarrow \text{truck}(y), \neg\text{truck}(x)$".
    - $\text{load}(x)$: "$\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T), \neg\text{pack}(x)$".
    - $\text{unload}(x)$: "$\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x), \neg\text{pack}(T)$".
- **Example 18.2.4 ("Only-Adds" Relaxation).** Drop the preconditions and deletes.
  - "$\text{drive}(x,y)$: $\Rightarrow \text{truck}(y)$";
  - "$\text{load}(x)$: $\Rightarrow \text{pack}(T)$";
  - "$\text{unload}(x)$: $\Rightarrow \text{pack}(x)$".
- Heuristics value for I is?
- $h^{\mathcal{R}}(I) = 1$: A plan for the relaxed task is $\langle\text{unload}(D)\rangle$.

We will start with a very simple relaxation, which could be termed "positive thinking": we do not consider preconditions of actions and leave out the delete lists as well.

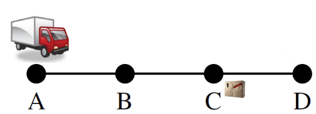
How to Relax During Search: Overview

- **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 $h(s)$!!!**



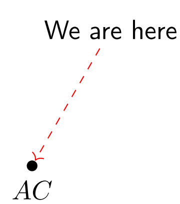
  - Here, Π_s is Π with initial state replaced by s, i.e., $\Pi := \langle P, A, I, G \rangle$ changed to $\Pi^s := \langle P, A, \{s\}, G \rangle$: The task of finding a plan for search state s.
  - A common student error is to instead apply the relaxation once to the whole problem, then doing the whole search "within the relaxation".
  - The next slide illustrates the correct search process in detail.

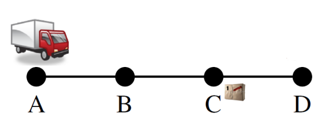
How to Relax During Search: Only-Adds



Real problem:
- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

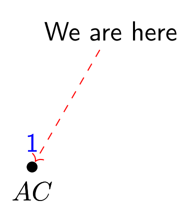
Greedy best-first search: (tie-breaking: alphabetic)

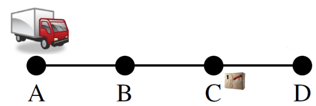




Relaxed problem:
- State s: AC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 1$: $\langle ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

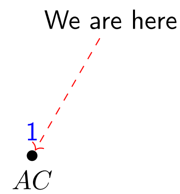


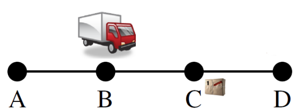


Relaxed problem:

- ▷ State s: AC; goal G: AD.
- ▷ Actions A: add.
- ▷ $h^{\mathcal{R}}(s)$ =1: $\langle ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

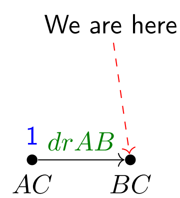


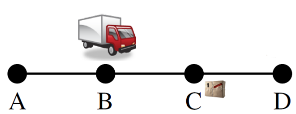


Real problem:

- ▷ State s: BC; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ $AC \xrightarrow{drAB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)

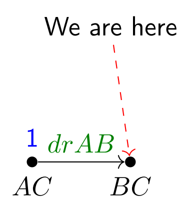


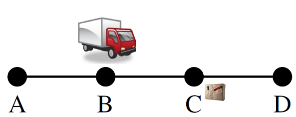


Relaxed problem:

- ▷ 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)

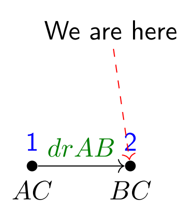


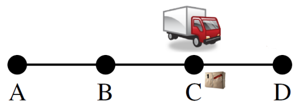


Relaxed problem:

- ▷ 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)

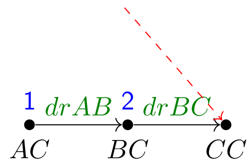


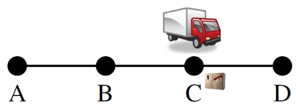


Real problem:
- ▷ State s: CC; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ $BC \xrightarrow{drBC} CC$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here

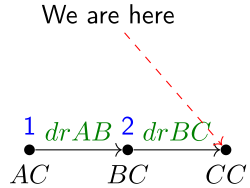


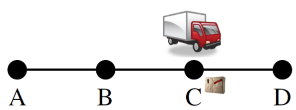


Relaxed problem:
- ▷ State s: CC; goal G: AD.
- ▷ Actions A: add.
- ▷ $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here

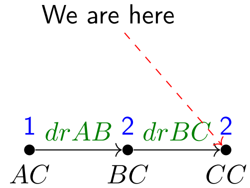


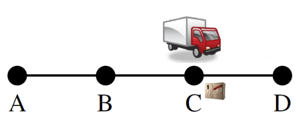


Relaxed problem:
- ▷ State s: CC; goal G: AD.
- ▷ Actions A: add.
- ▷ $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here

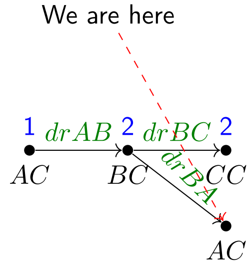


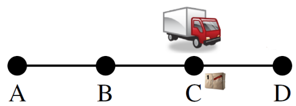


Real problem:
- ▷ State s: AC; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ $BC \xrightarrow{drBA} AC$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here

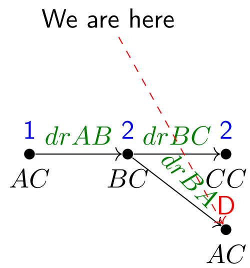


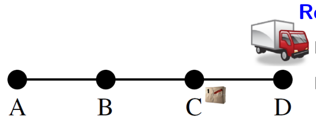


Real problem:

- ▷ State s: AC; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)

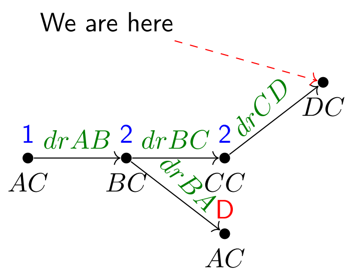


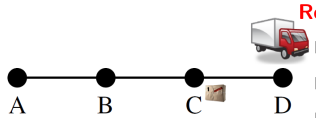


Real problem:

- ▷ State s: DC; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ $CC \xrightarrow{drCD} DC$.

Greedy best-first search: (tie-breaking: alphabetic)

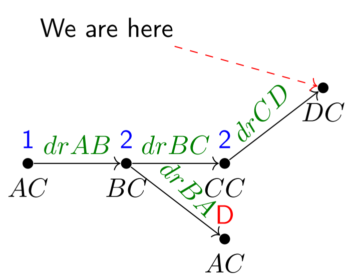


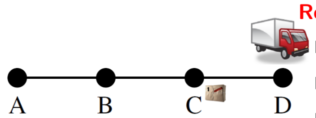


Relaxed problem:

- ▷ 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)

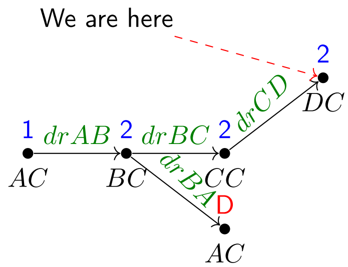


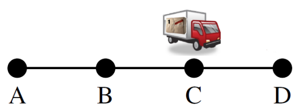


Relaxed problem:

- ▷ 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)

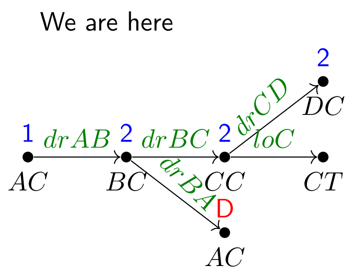


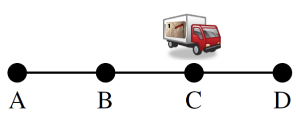


Real problem:

- ▷ State s: CT; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ $CC \xrightarrow{loC} CT$.

Greedy best-first search: (tie-breaking: alphabetic)

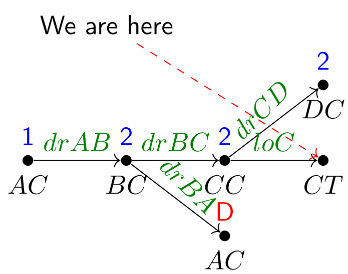


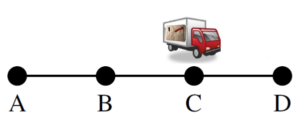


Relaxed problem:

- ▷ 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)

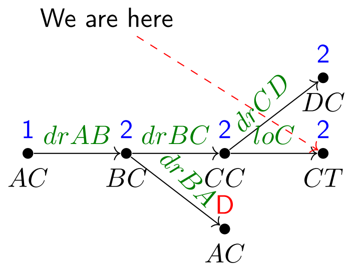


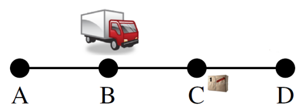


Relaxed problem:

- ▷ 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)

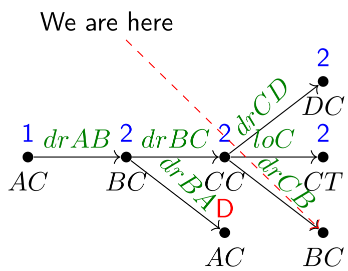


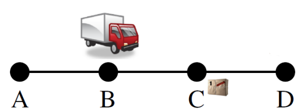


Real problem:

- ▷ State s: BC; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ $CC \xrightarrow{drCB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)

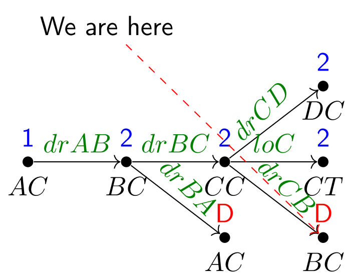


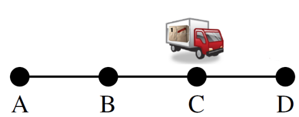


Real problem:

- ▷ State s: BC; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)

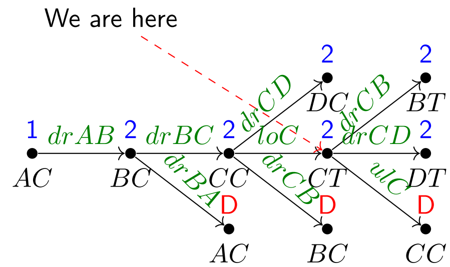


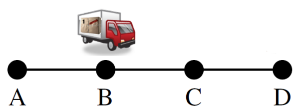


Real problem:

- ▷ State s: CT; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Successors: BT, DT, CC.

Greedy best-first search: (tie-breaking: alphabetic)

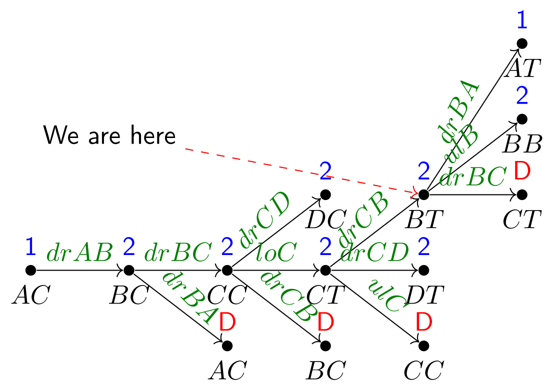


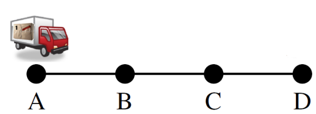


Real problem:

- ▷ State s: BT; goal G: AD.
- ▷ Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- ▷ Successors: AT, BB, CT.

Greedy best-first search: (tie-breaking: alphabetic)

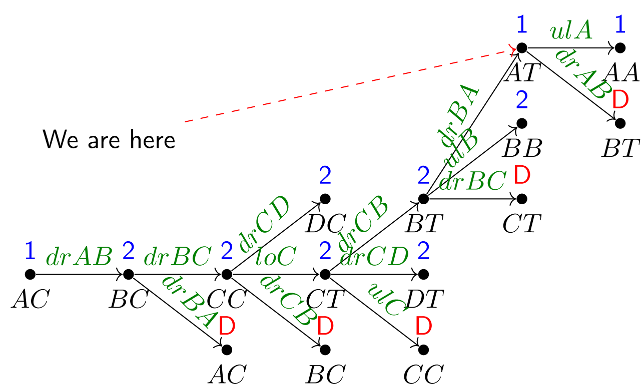


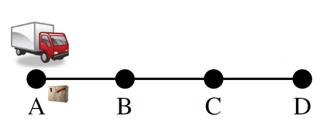


Real problem:

- ▷ State s: AT; goal G: AD.
- ▷ Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- ▷ Successors: AA, BT.

Greedy best-first search: (tie-breaking: alphabetic)

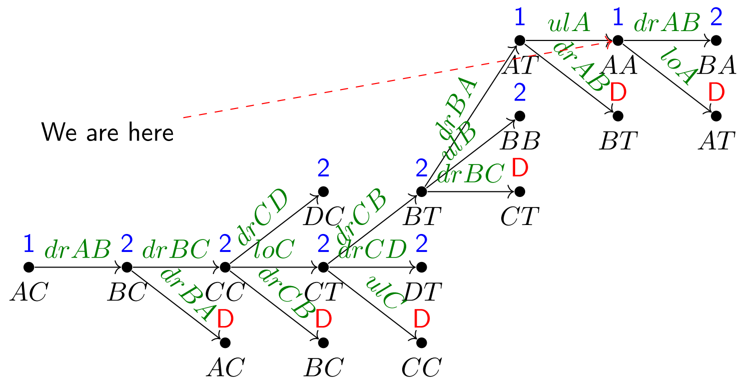


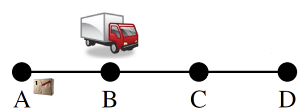


Real problem:

- ▷ State s: AA; goal G: AD.
- ▷ Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- ▷ Successors: BA, AT.

Greedy best-first search: (tie-breaking: alphabetic)

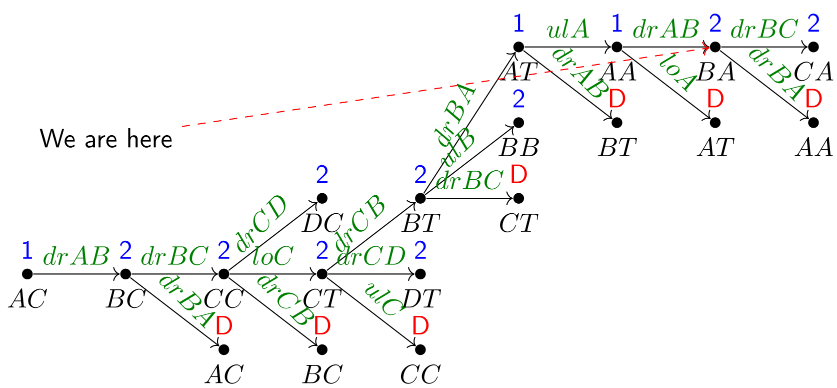


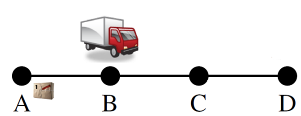


Real problem:

- ▷ State s: BA; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Successors: CA, AA.

Greedy best-first search: (tie-breaking: alphabetic)

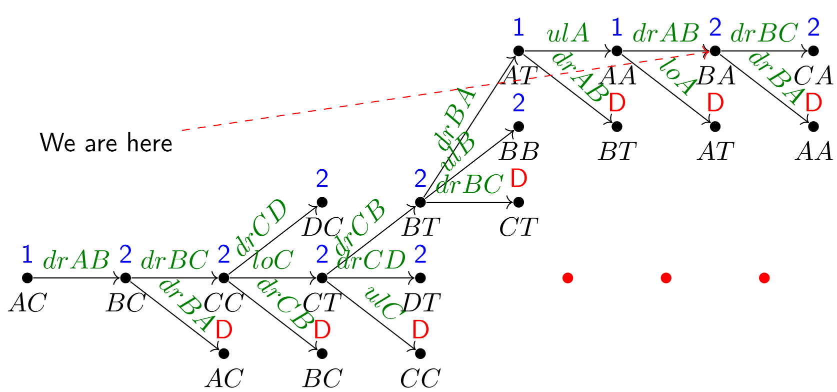




Real problem:

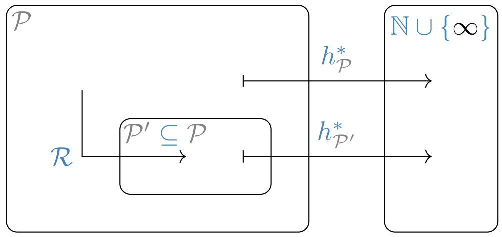
- ▷ State s: BA; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Successors: CA, AA.

Greedy best-first search: (tie-breaking: alphabetic)



Only-Adds is a "Native" Relaxation

▷ **Definition 18.2.5 (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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26903`.

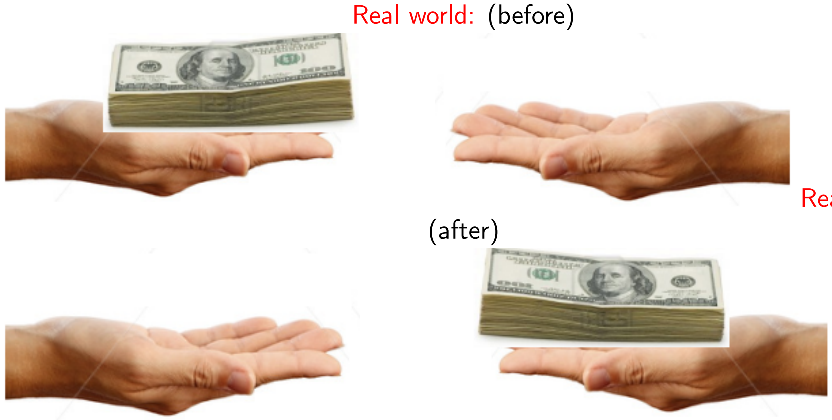
We turn to a more realistic relaxation, where we only disregard the delete list.

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)



Real world: (after)

Relaxed world: (before)


Relaxed

world: (after)


How the Delete Relaxation Changes the World (II)

▷ Relaxation mapping $\mathcal{R}$ saying that:

Real world: (before)

Real world: (after)

Relaxed world: (before)

Relaxed world: (after)

How the Delete Relaxation Changes the World (III)

▷ Relaxation mapping $\mathcal{R}$ saying that:

Real world:

Relaxed world:

The Delete Relaxation

▷ **Definition 18.3.1 (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 $A^+ := \{a^+ \mid a \in A\}$ with $\mathrm{pre}_{a^+} := \mathrm{pre}_a$, $\mathrm{add}_{a^+} := \mathrm{add}_a$, and $\mathrm{del}_{a^+} := \emptyset$.

▷ In other words, the class of simpler problems $\mathcal{P}'$ is the set of all STRIPS tasks with empty delete lists, and the relaxation mapping $\mathcal{R}$ drops the delete lists.

▷ **Definition 18.3.2 (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.

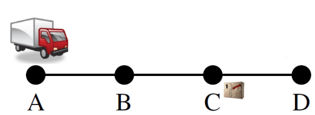
▷ A relaxed plan for s is an action sequence that solves s when pretending that all delete lists are empty.

▷ 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:** $\{\text{at}(\text{Sy}), \text{vis}(\text{Sy})\}$.
2. $\text{drv}(\text{Sy}, \text{Br})^+$: $\{\text{at}(\text{Br}), \text{vis}(\text{Br}), \text{at}(\text{Sy}), \text{vis}(\text{Sy})\}$.
3. $\text{drv}(\text{Sy}, \text{Ad})^+$: $\{\text{at}(\text{Ad}), \text{vis}(\text{Ad}), \text{at}(\text{Br}), \text{vis}(\text{Br}), \text{at}(\text{Sy}), \text{vis}(\text{Sy})\}$.
4. $\text{drv}(\text{Ad}, \text{Pe})^+$: $\{\text{at}(\text{Pe}), \text{vis}(\text{Pe}), \text{at}(\text{Ad}), \text{vis}(\text{Ad}), \text{at}(\text{Br}), \text{vis}(\text{Br}), \text{at}(\text{Sy}), \text{vis}(\text{Sy})\}$.
5. $\text{drv}(\text{Ad}, \text{Da})^+$: $\{\text{at}(\text{Da}), \text{vis}(\text{Da}), \text{at}(\text{Pe}), \text{vis}(\text{Pe}), \text{at}(\text{Ad}), \text{vis}(\text{Ad}), \text{at}(\text{Br}), \text{vis}(\text{Br}), \text{at}(\text{Sy}), \text{vis}(\text{Sy})\}$.

A Relaxed Plan for "Logistics"



▷ Facts P: $\{\text{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\text{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.

▷ Initial state I: $\{\text{truck}(A), \text{pack}(C)\}$.

▷ Goal G: $\{\text{truck}(A), \text{pack}(D)\}$.

▷ Relaxed actions A^+: (Notated as "precondition $\Rightarrow$ adds")

- ▷ $\text{drive}(x, y)^+$: "$\text{truck}(x) \Rightarrow \text{truck}(y)$".

- ▷ $\text{load}(x)^+$: "$\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T)$".
- ▷ $\text{unload}(x)^+$: "$\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x)$".

Relaxed plan:

$$\langle \text{drive}(A,B)^+, \text{drive}(B,C)^+, \text{load}(C)^+, \text{drive}(C,D)^+, \text{unload}(D)^+ \rangle$$

▷ We don't need to drive the truck back, because "it is still at A".

$\mathbf{PlanEx}^+$

▷ **Definition 18.3.3 (Relaxed Plan Existence Problem).** By $\mathbf{PlanEx}^+$, 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 $\mathbf{PlanEx}$ for general STRIPS!

▷ $\mathbf{PlanEx}^+$ is in $\mathbf{P}$.

▷ *Proof:* The following algorithm decides $\mathbf{PlanEx}^+$

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 632

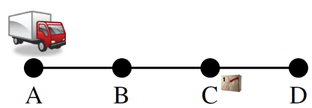
Deciding $\mathbf{PlanEx}^+$ in "TSP" in Australia

Iterations on F:

1. $\{\mathrm{at(Sy)}, \mathrm{vis(Sy)}\}$
2. $\cup$ $\{\mathrm{at(Ad)}, \mathrm{vis(Ad)}, \mathrm{at(Br)}, \mathrm{vis(Br)}\}$
3. $\cup$ $\{\mathrm{at(Da)}, \mathrm{vis(Da)}, \mathrm{at(Pe)}, \mathrm{vis(Pe)}\}$

Deciding PlanEx^+ in "Logistics"

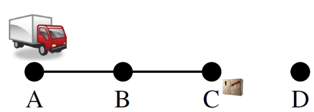
▷ **Example 18.3.4 (The solvable Case).**



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 18.3.5 (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 $\text{apply}(I, \langle a_0^+, \ldots, a_{i-1}^+\rangle) \subseteq F_i$, for all i.

5.3. We get $G \subseteq \text{apply}(I, \langle a_0^+, \ldots, a_{n-1}^+\rangle) \subseteq F_n$, and the algorithm returns "solvable" as desired.

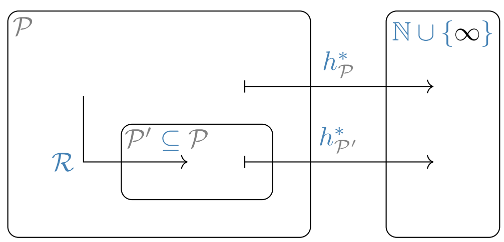
5.4. Assume to the contrary of the claim that, in an iteration $i < n$, (*) fires. Then $G \not\subseteq F$ and $F = F'$. But, with $F = F'$, $F = F_j$ for all $j > i$, and we get $G \not\subseteq F_n$ in contradiction.

FAU Michael Kohlhase: Artificial Intelligence 1 632 2025-02-06

18.4 The h^+Heuristic

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26905`.

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* $h^* \circ \mathcal{R}$.

FAU Michael Kohlhase: Artificial Intelligence 1 633 2025-02-06

h^+: The Ideal Delete Relaxation Heuristic

▷ **Definition 18.4.1 (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 626, just adding the word "optimal".

▷ Here's what we're looking for:

▷ **Definition 18.4.2.** 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 $h^+ : S \to \mathbb{N} \cup \{\infty\}$ where $h^+(s)$ is the length of an optimal relaxed plan for s if a relaxed plan for s exists, and $h^+(s) = \infty$ otherwise.

▷ In other words, $h^+ = h^* \circ \mathcal{R}$, cf. previous slide.

h^+ is Admissible

▷ **Lemma 18.4.3.** *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 $\mathrm{apply}(s, \langle a_1, \ldots, a_i\rangle) \subseteq \mathrm{apply}(s, \langle a_1^+, \ldots, a_i^+\rangle)$.

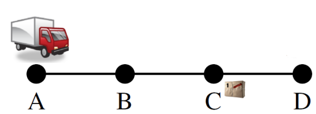
▷ If we ignore deletes, the states along the plan can only get bigger.

▷ **Theorem 18.4.4.** 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



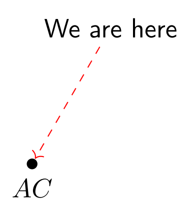
Real problem:

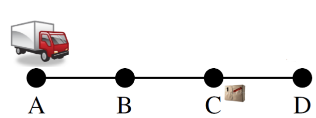
▷ Initial state I: AC; goal G: AD.

▷ Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.

▷ $drXY, loX, ulX$.

Greedy best-first search: (tie-breaking: alphabetic)





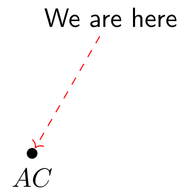
Relaxed problem:

▷ State s: AC; goal G: AD.

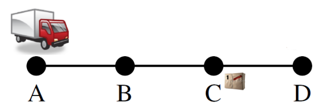
▷ Actions A: $\mathrm{pre}, \mathrm{add}$.

▷ $h^+(s)$ =5: e.g. $\langle drAB, drBC, drCD, loC, ulD\rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

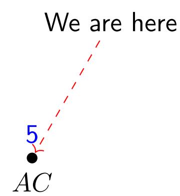


Relaxed problem:

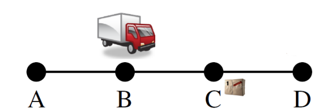


- ▷ State s: AC; goal G: AD.
- ▷ Actions A: pre, add.
- ▷ $h^+(s)$ =5: e.g. $\langle drAB, drBC, drCD, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

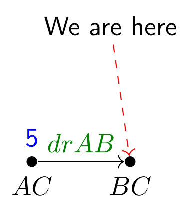


Real problem:

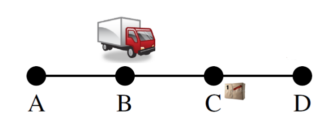


- ▷ State s: BC; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ $AC \xrightarrow{drAB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)

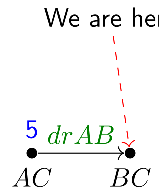


Relaxed problem:

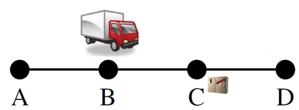


- ▷ State s: BC; goal G: AD.
- ▷ Actions A: pre, add.
- ▷ $h^+(s)$ =5: e.g. $\langle drBA, drBC, drCD, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



Relaxed problem:

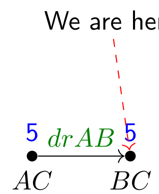


▷ State s: BC; goal G: AD.

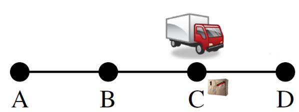
▷ Actions A: $\mathrm{pre}, \mathrm{add}$.

▷ $h^+(s)$ =5: e.g. $\langle drBA, drBC, drCD, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



Real problem:

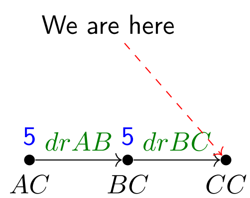


▷ State s: CC; goal G: AD.

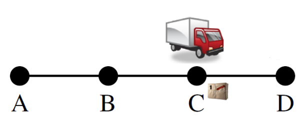
▷ Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.

▷ $BC \xrightarrow{drBC} CC$.

Greedy best-first search: (tie-breaking: alphabetic)



Relaxed problem:

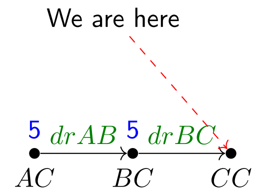


▷ State s: CC; goal G: AD.

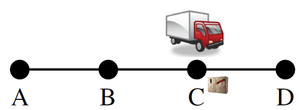
▷ Actions A: $\mathrm{pre}, \mathrm{add}$.

▷ $h^+(s)$ =5: e.g. $\langle drCB, drBA, drCD, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

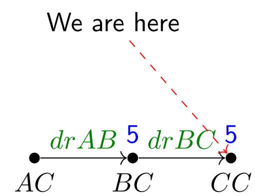


Relaxed problem:

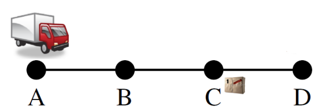


- State s: CC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drCB, drBA, drCD, loC, ulD\rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

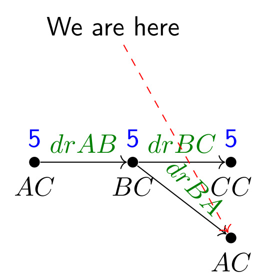


Real problem:

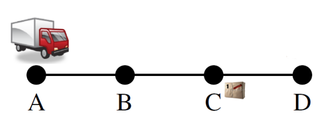


- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBA} AC$.

Greedy best-first search: (tie-breaking: alphabetic)

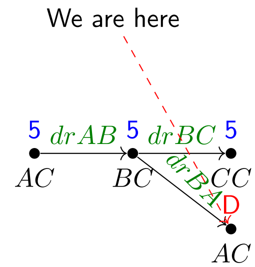


Real problem:

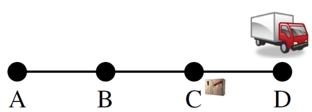


- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)

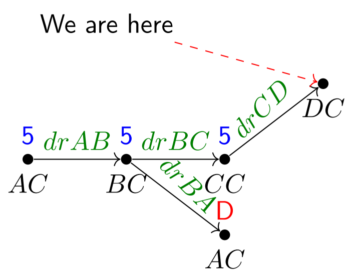


Real problem:

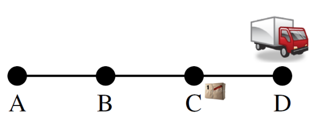


- State s: DC; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search: (tie-breaking: alphabetic)

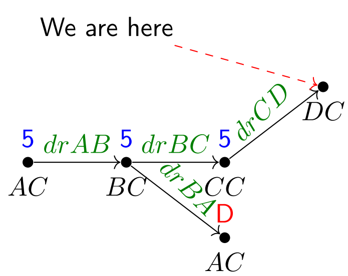


Relaxed problem:

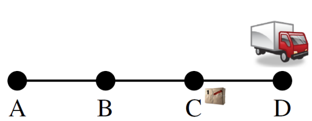


- State s: DC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drDC, drCB, drBA, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

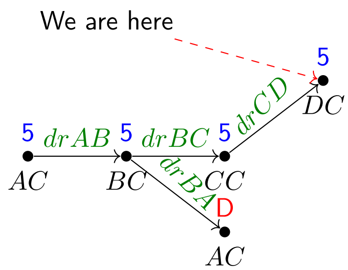


Relaxed problem:

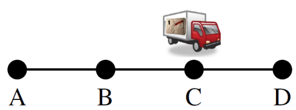


- State s: DC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drDC, drCB, drBA, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

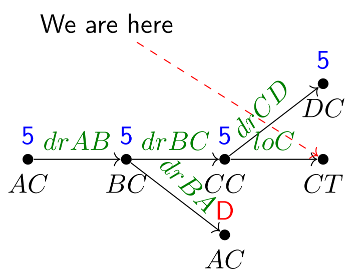


Real problem:

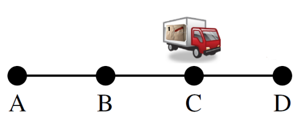


- ▷ State s: CT; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ $CC \xrightarrow{loC} CT$.

Greedy best-first search: (tie-breaking: alphabetic)

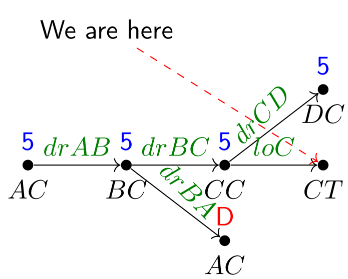


Relaxed problem:

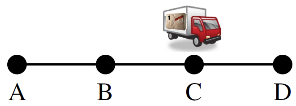


- ▷ State s: CT; goal G: AD.
- ▷ Actions A: pre, add.
- ▷ $h^+(s)$ =4: e.g. $\langle drCB, drBA, drCD, ulD\rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

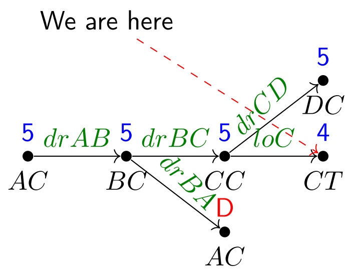


Relaxed problem:

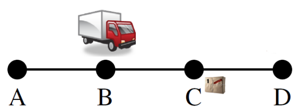


- State s: CT; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s)$ =4: e.g. $\langle drCB, drBA, drCD, ulD\rangle$.

Greedy best-first search: (tie-breaking: alphabetic)

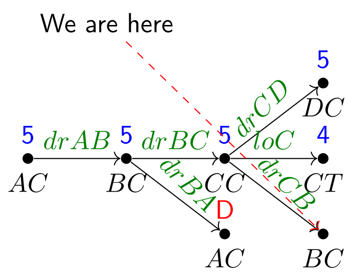


Real problem:

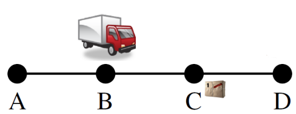


- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{drCB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)

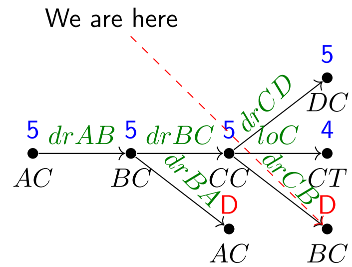


Real problem:

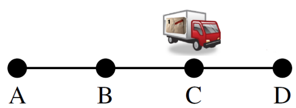


- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)

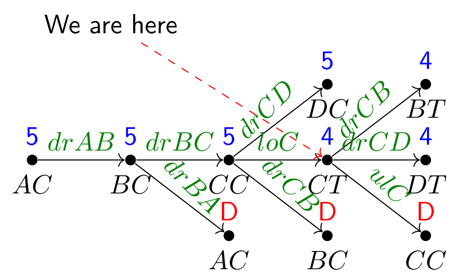


Real problem:

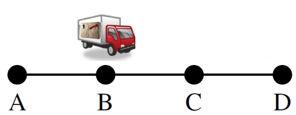


- ▷ State s: CT; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Successors: BT, DT, CC.

Greedy best-first search: (tie-breaking: alphabetic)

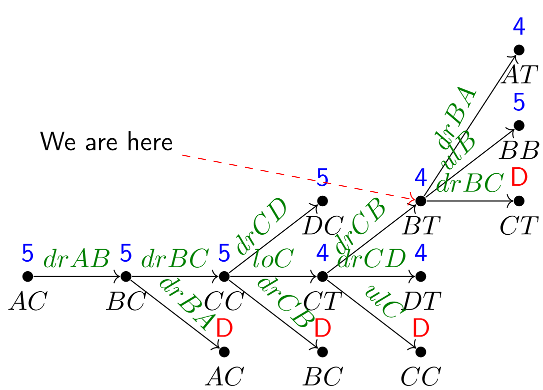


Real problem:

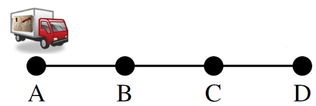


- ▷ State s: BT; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Successors: AT, BB, CT.

Greedy best-first search: (tie-breaking: alphabetic)



Real problem:

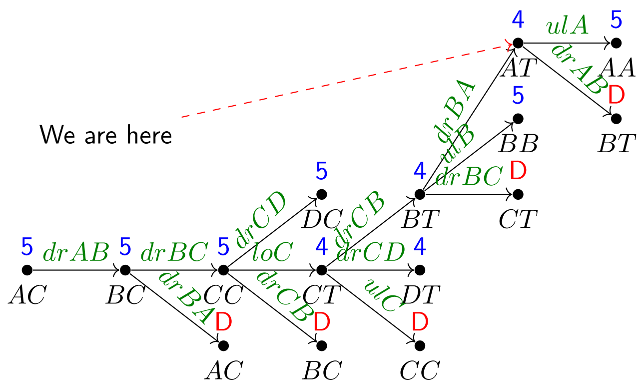


▷ State s: AT; goal G: AD.

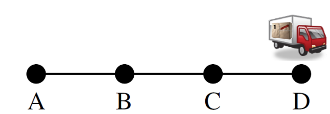
▷ Actions A: pre, add, del.

▷ Successors: AA, BT.

Greedy best-first search: (tie-breaking: alphabetic)



Real problem:

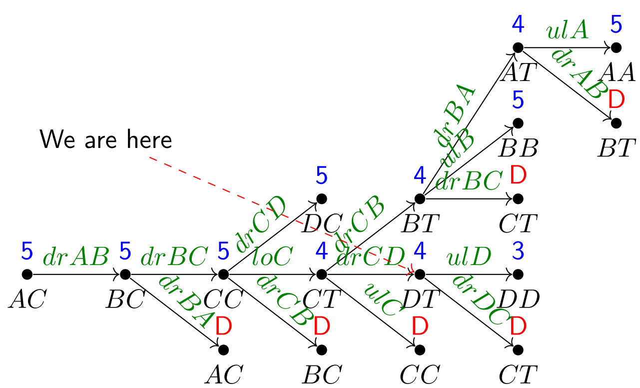


▷ State s: DT; goal G: AD.

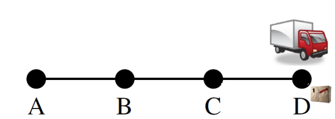
▷ Actions A: pre, add, del.

▷ Successors: DD, CT.

Greedy best-first search: (tie-breaking: alphabetic)



Real problem:

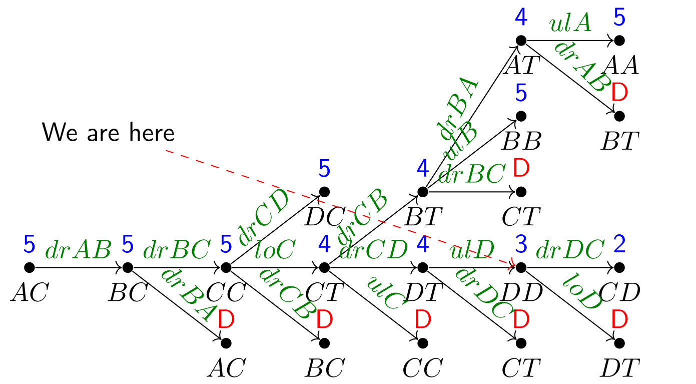


▷ State s: DD; goal G: AD.

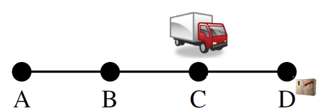
▷ Actions A: pre, add, del.

▷ Successors: CD, DT.

Greedy best-first search: (tie-breaking: alphabetic)

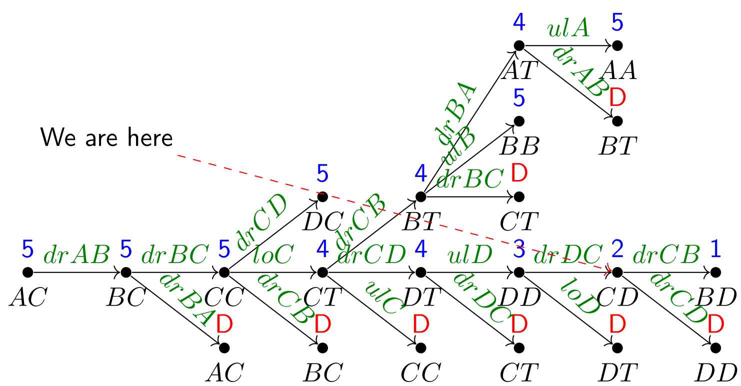


Real problem:

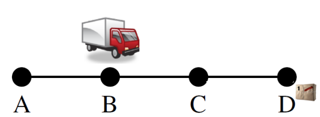


- ▷ State s: CD; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Successors: BD, DD.

Greedy best-first search: (tie-breaking: alphabetic)

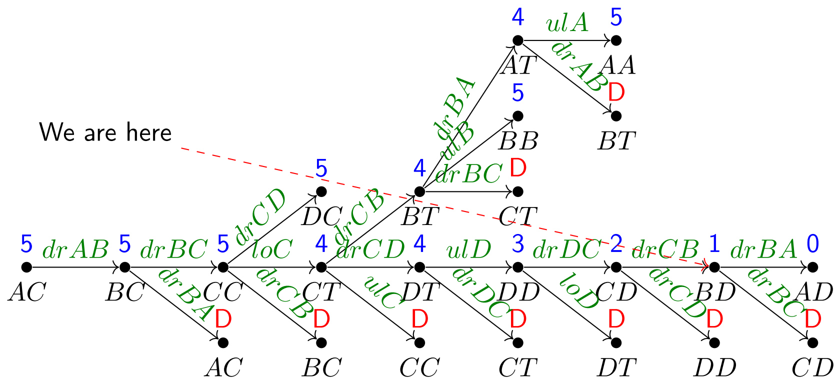


Real problem:

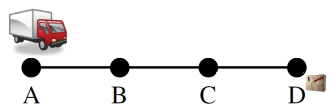


- ▷ State s: BD; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Successors: AD, CD.

Greedy best-first search: (tie-breaking: alphabetic)

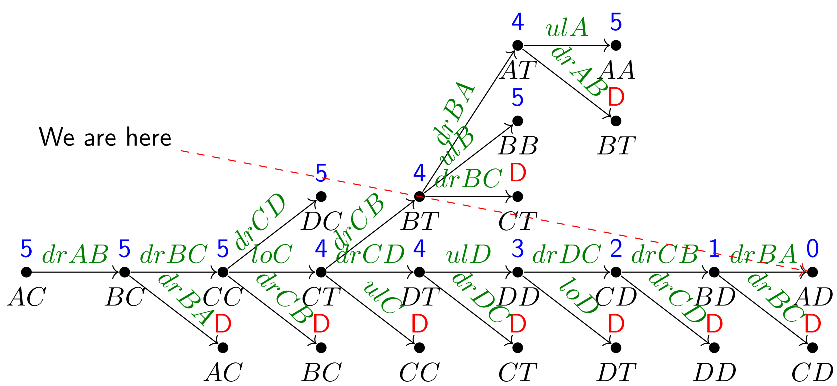


Real problem:



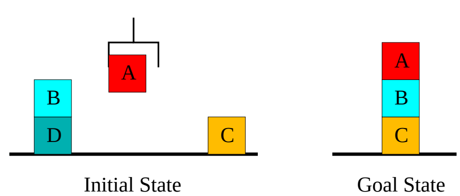
- ▷ State s: AD; goal G: AD.
- ▷ Actions A: pre, add, del.
- ▷ Goal state!

Greedy best-first search: (tie-breaking: alphabetic)



Of course there are also bad cases. Here is one.

h^+ in the Blocksworld

▷


- ▷ **Optimal plan**: $\langle$putdown(A), unstack(B,D), stack(B,C), pickup(A), stack$(A,B)\rangle$.
- ▷ **Optimal relaxed plan**: $\langle$stack(A,B), unstack(B,D), 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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26906`.

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 $\mathbf{NP}$-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** ...

Suggested Reading (RN: Same As Previous Chapter):

- Chapters 10: *Classical Planning* and 11: *Planning and Acting in the Real World* in [RN09].
  - Although the book is named "*A Modern Approach*", the planning section was written long before the IPC was even dreamt of, before PDDL was conceived, and several years before heuristic search hit the scene. As such, what we have right now is the attempt of two outsiders trying in vain to catch up with the dramatic changes in planning since 1995.
  - Chapter 10 is Ok as a background read. Some issues are, imho, misrepresented, and it's far from being an up-to-date account. But it's Ok to get some additional intuitions in words different from my own.
  - Chapter 11 is useful in our context here because we don't cover any of it. If you're interested in extended/alternative planning paradigms, do read it.
- A good source for modern information (some of which we covered in the course) is Jörg Hoffmann's *Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)* [Hof11] which is available online at `http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf`

Chapter 19

Searching, Planning, and Acting in the Real World

Outline

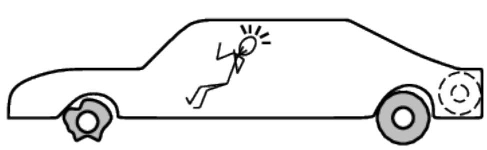
- ▷ **So Far:** we made idealizing/simplifying assumptions:
  The environment is fully observable and deterministic.
- ▷ **Outline:** In this chapter we will lift some of them
  - ▷ The real world (things go wrong)
  - ▷ Agents and Belief States
  - ▷ Conditional planning
  - ▷ Monitoring and replanning
- ▷ **Note:** The considerations in this chapter apply to both search and planning.

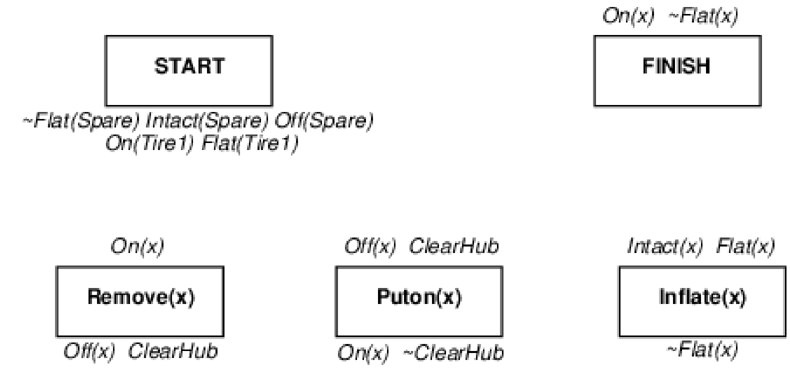
19.1 Introduction

A Video Nugget covering this section can be found at https://fau.tv/clip/id/26908.

The real world

- ▷ **Example 19.1.1.** We have a flat tire – what to do?





Generally: Things go wrong (in the real world)

- **Example 19.1.2 (Incomplete Information).**
  - Unknown preconditions, e.g., $Intact(Spare)$?
  - Disjunctive effects, e.g., $Inflate(x)$ causes $Inflated(x) \vee SlowHiss(x) \vee Burst(x) \vee BrokenPump \vee \ldots$
- **Example 19.1.3 (Incorrect Information).**
  - Current state incorrect, e.g., spare NOT intact
  - Missing/incorrect effects in actions.
- **Definition 19.1.4.** 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 19.1.5 (A conditional Plan).** (AAA $\widehat{=}$ ADAC)
$[Check(T1), \text{if } Intact(T1) \text{ then } Inflate(T1) \text{ else } CallAAA \text{ fi}]$

▷ **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 19.1.6.** *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

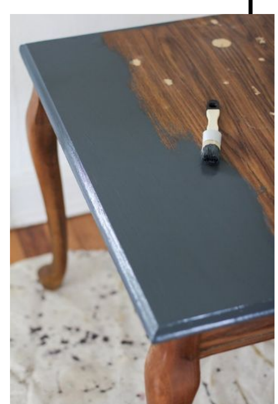
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/29180`.
We now introduce a planning example that shows off the various features.

The Furniture-Coloring Example: Specification

▷ **Example 19.2.1 (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.



We formalize the example in PDDL for simplicity. Note that the :percept scheme is not part of the official PDDL, but fits in well with the design.

The Furniture-Coloring Example: PDDL

▷ **Example 19.2.2 (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 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)))
```

- 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.
- **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.

- 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))))
```

- **Contingent Plan**:
  1. look at furniture to determine color, if same ⇝ done.
  2. else, look at open and look at paint in cans
  3. if paint in one can is the same as an object, paint the other with this color
  4. else paint both in any color

19.3 Searching/Planning with Non-Deterministic Actions

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/29181`.

Conditional Plans

▷ **Definition 19.3.1.** 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 19.3.2.** Conditional plans can contain

- ▹ conditional step: $[\ldots, \text{if } C \text{ then } Plan_A \text{ else } Plan_B \text{ fi}, \ldots]$,
- ▹ while step: $[\ldots, \text{while } C \text{ do } Plan \text{ done}, \ldots]$, and
- ▹ the empty plan $\emptyset$ to make modeling easier.

▷ **Definition 19.3.3.** 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.

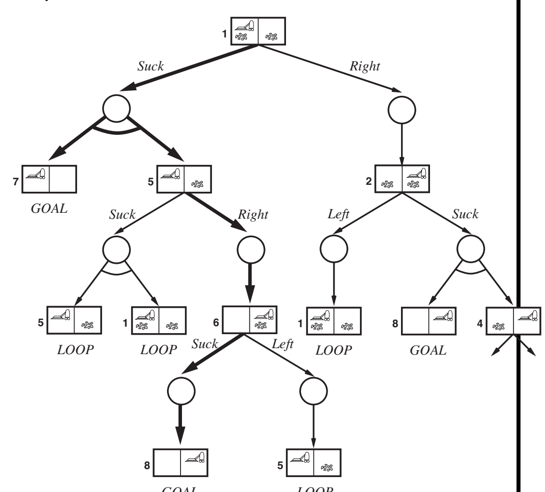
▷ **Idea:** Use an AND–OR tree search (very similar to backward chaining algorithm)

Contingency Planning: The Erratic Vacuum Cleaner

▷ **Example 19.3.4 (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: $[suck, \text{if } State = 5 \text{ then } [right, suck] \text{ else } [] \text{ fi}]$

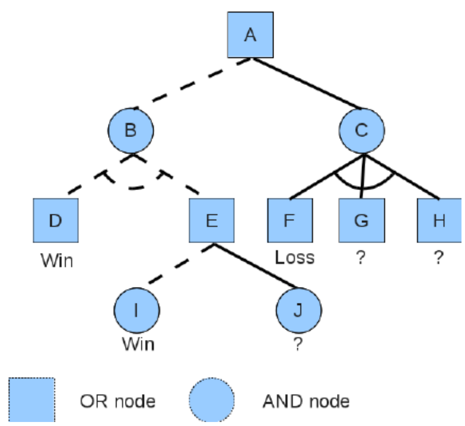
Conditional AND-OR Search (Data Structure)

- **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 19.3.5.** 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 19.3.6.** 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 19.3.7 (An AND-OR-tree).**



Conditional AND-OR Search (Algorithm)

- **Definition 19.3.8.** 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**

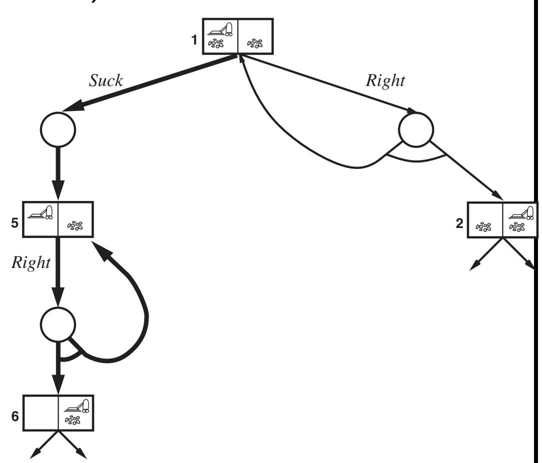
▷ **fail** does not mean *there is no solution*, but

▷ *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 19.3.9 (Slippery Vacuum World).**

Moving sometimes fails $\rightsquigarrow$ AND-OR graph



Two possible solutions (depending on what our plan language allows)

▷ $[L_1 : \ left, \text{if } AtR \text{ then } L_1 \text{ else } [\text{if } CleanL \text{ then } \emptyset \text{ else } suck \text{ fi}] \text{ fi}]$ or

▷ $[\text{while } AtR \text{ do } [left] \text{ done}, \text{if } CleanL \text{ then } \emptyset \text{ else } suck \text{ fi}]$

▷ We have an infinite loop but plan eventually works unless action always fails.

19.4 Agent Architectures based on Belief States

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/29182`.

We are now ready to proceed to environments which can only partially observed and where actions are non deterministic. Both sources of uncertainty conspire to allow us only partial knowledge about the world, so that we can only optimize "expected utility" instead of "actual utility" of our 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 19.4.1.** 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.

That is exactly what we have been doing until now: we have been studying methods that build on descriptions of the "actual" world, and have been concentrating on the progression from atomic to factored and ultimately structured representations. Tellingly, we spoke of "world states" instead of "belief states"; we have now justified this practice in the brave new belief-based world models by the (re-) definition of "world states" above. To fortify our intuitions, let us recap from a belief-state-model perspective.

World Models by Agent Type in AI-1

- **Search-based Agents:** In a fully observable, deterministic environment
  - goal-based agent with world state $\widehat{=}$ "current state"
  - no inference. (goal $\widehat{=}$ goal state from search problem)
- **CSP-based Agents:** In a fully observable, deterministic environment
  - goal-based agent withworld state $\widehat{=}$ constraint network,
  - inference $\widehat{=}$ constraint propagation. (goal $\widehat{=}$ satisfying assignment)
- **Logic-based Agents:** In a fully observable, deterministic environment
  - model-based agent with world state $\widehat{=}$ logical formula
  - inference $\widehat{=}$ e.g. DPLL or resolution.
- **Planning Agents:** In a fully observable, deterministic, environment
  - goal-based agent with world state $\widehat{=}$ PL0, transition model $\widehat{=}$ STRIPS,
  - inference $\widehat{=}$ state/plan space search. (goal: complete plan/execution)

Let us now see what happens when we lift the restrictions of total observability and determinism.

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

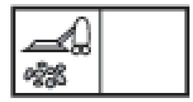
- ▷ **Probabilistic Agents:** In a partially observable environment
  - ▷ belief state $\widehat{=}$ Bayesian networks,
  - ▷ inference $\widehat{=}$ probabilistic inference.
- ▷ **Decision-Theoretic Agents:** In a partially observable, stochastic environment
  - ▷ belief state + transition model $\widehat{=}$ decision networks,
  - ▷ inference $\widehat{=}$ maximizing expected utility.
- ▷ We will study them in detail in the coming semester.

19.5 Searching/Planning without Observations

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/29183`.

Conformant/Sensorless Planning

- ▷ **Definition 19.5.1.** Conformant or sensorless planning tries to find plans that work without any sensing. (not even the initial state)



▷ **Example 19.5.2 (Sensorless Vacuum Cleaner World).**

States	integer dirt and robot locations
Actions	$left, right, suck, noOp$
Goal states	$notdirty?$

▷ **Observation 19.5.3.** *In a sensorless world we do not know the initial state. (or any state after)*

▷ **Observation 19.5.4.** *Sensorless planning must search in the space of belief states (sets of possible actual states).*

▷ **Example 19.5.5 (Searching the Belief State Space).**

- ▷ Start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
- ▷ Solution: $[right, suck, left, suck]$ $right \rightarrow \{2, 4, 6, 8\}$, $suck \rightarrow \{4, 8\}$, $left \rightarrow \{3, 7\}$, $suck \rightarrow \{7\}$

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 $\mathcal{T} : \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, 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 $\mathcal{G}^b := \{S \in \mathcal{S}^b \,|\, S \subseteq \mathcal{G}\}$ (all possible states must be physical goal states)
- ▷ Actions $\mathcal{A}^b$: we just take $\mathcal{A}$. (that's the point!)
- ▷ Transition model $\mathcal{T}^b : \mathcal{A}^b \times \mathcal{S}^b \rightarrow \mathcal{P}(\mathcal{A}^b)$: 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 19.5.6.** *In belief-state space the problem is always fully observable!*

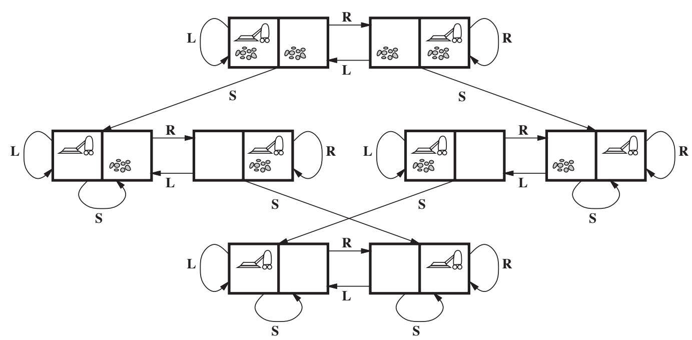
Let us see if we can understand the options for $\mathcal{T}^b(a, S)$ a bit better. The first question is when we want an action a to be applicable to a belief state $S \subseteq \mathcal{S}$, i.e. when should $\mathcal{T}^b(a, S)$ be non-empty.

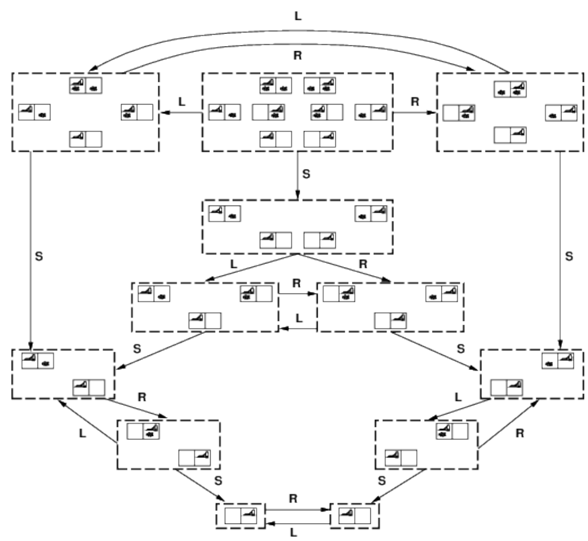
In the first case, a^b would be applicable iff a is applicable to some $s \in S$, in the second case if a is applicable to all $s \in S$. So we only want to choose the first case if actions are harmless.

The second question we ask ourselves is what should be the results of applying a to $S \subseteq \mathcal{S}$?, again, if actions are harmless, we can just collect the results, otherwise, we need to make sure that all members of the result a^b are reached for all possible states in S.

State Space vs. Belief State Space

▷ **Example 19.5.7 (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

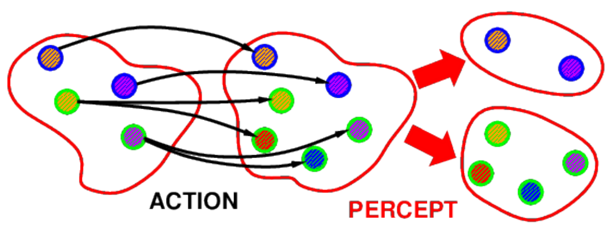
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/29184`.

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 657 to include partial observability.

▷ **Definition 19.6.1.** 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: $\mathrm{PossPERC}(\widehat{b}) = \{\mathrm{PERC}(s) \mid s \in$

$\widehat{b}\}$.

- ▷ The update stage determines, for each possible percept, the resulting belief state: $\mathrm{UPDATE}(\widehat{b}, o) := \{s \,|\, o = \mathrm{PERC}(s) \text{ and } s \in \widehat{b}\}$

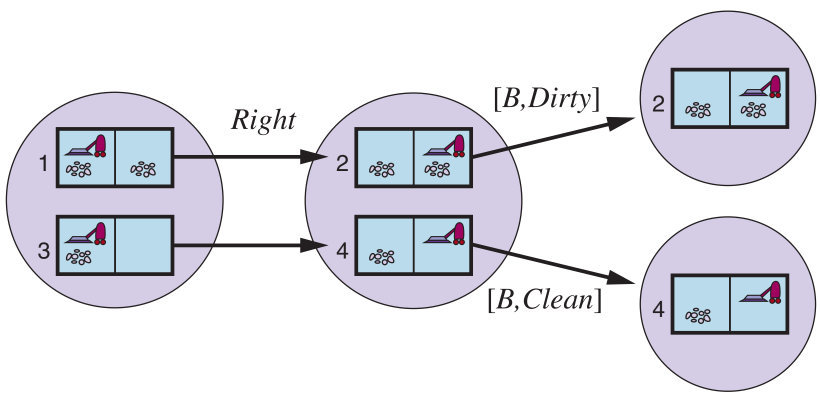
The functions PRED and PERC are the main parameters of this model. We define $\mathrm{RESULT}(b, a){:=}\{\mathrm{UPDATE}(\mathrm{PRED}(b, a), o) \,|\, \mathrm{PossPERC}(\mathrm{PRED}(b, a))\}$

▷ **Observation 19.6.2.** *We always have* $\mathrm{UPDATE}(\widehat{b}, o) \subseteq \widehat{b}$.

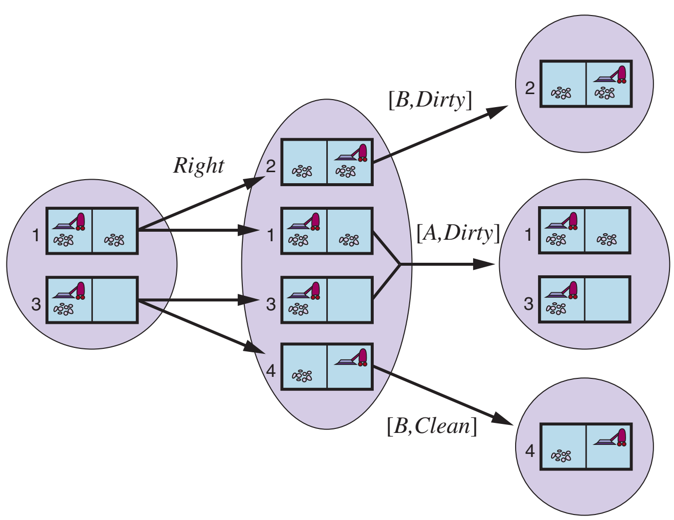
▷ **Observation 19.6.3.** *If sensing is deterministic, belief states for different possible percepts are disjoint, forming a partition of the original predicted belief state.*

Example: Local Sensing Vacuum Worlds

▷ **Example 19.6.4 (Transitions in the Vacuum World).** Deterministic World:



The action *Right* is deterministic, sensing disambiguates to singletons Slippery World:



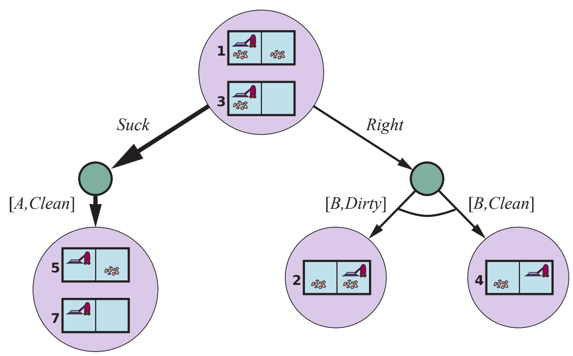
The action $Right$ is non-deterministic, sensing disambiguates somewhat

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 19.6.5.** AND-OR graph for initial percept $[A, Dirty]$.



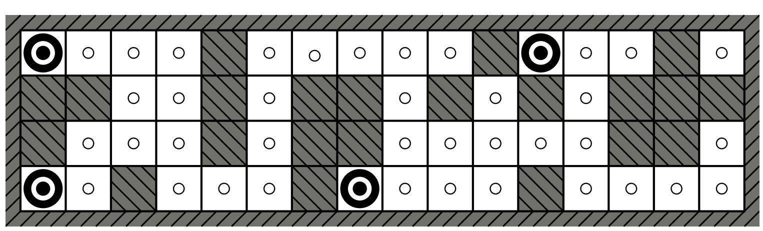
Solution: $[Suck, Right, \text{if } Bstate = \{6\} \text{ then } Suck \text{ else } [] \text{ fi}]$

▷ **Note:** Belief-state-problem $\leadsto$ conditional step tests on belief-state percept (plan would not be executable in a partially observable environment otherwise)

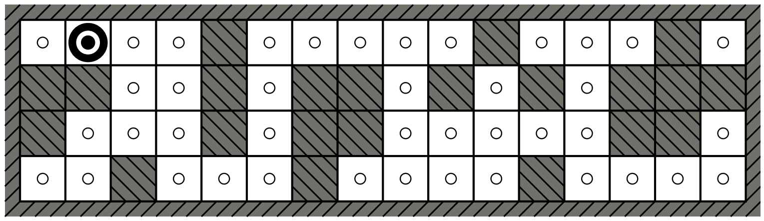
Example: Agent Localization

▷ **Example 19.6.6.** 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) $\leadsto \widehat{b}_2 = \text{UPDATE}(\widehat{b}_1, NWS)$



3. Agent executes $Move \leadsto \widehat{b}_3 = \text{PRED}(\widehat{b}_2, Move) = \textit{one step away from these}$.
4. Next percept: $NS \leadsto \widehat{b}_4 = \text{UPDATE}(\widehat{b}_3, NS)$



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 19.6.7.** 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 19.6.8.** Continuing ??.
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 19.6.9.** 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

- $\widehat{b} = b \backslash \text{del}_a \cup \text{add}_a$ (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.

AI-1 Survey on ALeA

▷ Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)

▷ Works on all common devices (mobile phone, notebook, etc.)

▷ Is in English; takes about 10 - 20 min
depending on proficiency in english and using ALeA

▷ Questions about how ALeA is used, what it is like usig ALeA, and questions about demography

▷ Token is generated at the end of the survey (SAVE THIS CODE!)

- Completed survey count as a successfull prepquiz in AI1!
- Look for Quiz 15 in the usual place (single question)
- just submit the token to get full points
- The token can also be used to exercise the rights of the GDPR.

▷ Survey has no timelimit and is free, anonymous, can be paused and continued later on and can be cancelled.

Find the Survey Here

`https://ddi-survey.cs.fau.de/limesurvey/index.php/667123?lang=en`

This URL will also be posted on the forum tonight.

FAU Michael Kohlhase: Artificial Intelligence 1 668 2025-02-06

19.7 Online Search

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/29185`.

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.

FAU Michael Kohlhase: Artificial Intelligence 1 669 2025-02-06

Online Search Problems

- **Observation:** Online problem solving even makes sense in deterministic, fully observable environments.
- **Definition 19.7.1.** A online search problem consists of a set S of states, and

- ▷ a function $\text{Actions}(s)$ 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 Goal Test.

▷ **Note:** We can only determine $\text{RESULT}(s, a)$ by being in s and executing a.

▷ **Definition 19.7.2.** The competitive ratio of an online problem solving agent is the quotient of

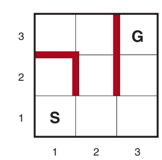
- ▷ offline performance, i.e. cost of optimal solutions with full information and
- ▷ online performance, i.e. the actual cost induced by online problem solving.

Online Search Problems (Example)

▷ **Example 19.7.3 (A simple maze problem).**

The agent starts at S and must reach G but knows nothing of the environment. In particular not that

- ▷ $\text{Up}(1, 1)$ results in $(1,2)$ and
- ▷ $\text{Down}(1, 1)$ results in $(1,1)$ (i.e. back)



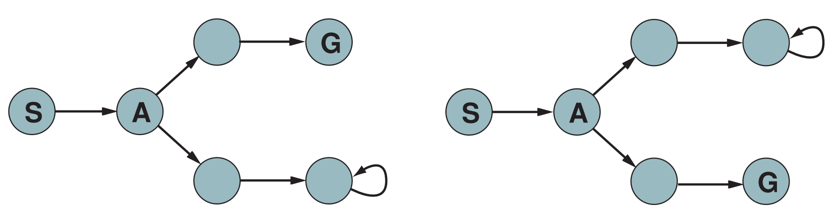
Online Search Obstacles (Dead Ends)

▷ **Definition 19.7.4.** 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.

▷ **Observation 19.7.5.** *No online algorithm can avoid dead ends in all state spaces.*

▷ **Example 19.7.6.** Two state spaces that lead an online agent into dead ends:

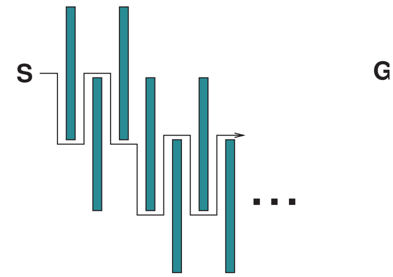


Any agent will fail in at least one of the spaces.

▷ **Definition 19.7.7.** We call ?? an adversary argument.

▷ **Example 19.7.8.** Forcing an online agent into an arbitrarily inefficient route:

Whichever choice the agent makes the adversary can block with a long, thin wall



▷ **Observation:** Dead ends are a real problem for robots: ramps, stairs, cliffs, ...

▷ **Definition 19.7.9.** 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 19.7.10.** The online depth first search algorithm:

function ONLINE–DFS–AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: $result$, a table mapping (s,a) **to** s', initially empty
    $untried$, a table mapping s **to** a list of untried actions
    $unbacktracked$, a table mapping s **to** a list backtracks not tried
    s, a, the previous state and action, initially null
  if Goal Test(s') **then return** stop
  if $s' \notin untried$ **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

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/29186`.

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 19.8.1.** The plan $[\mathrm{RePlan}(g)]$ is a (trivially) complete plan for any goal g. (not helpful)

▷ **Example 19.8.2.** A plan with sub-plans for every contingency (e.g. what to do if a meteor strikes) may be too costly/large. (wasted effort)

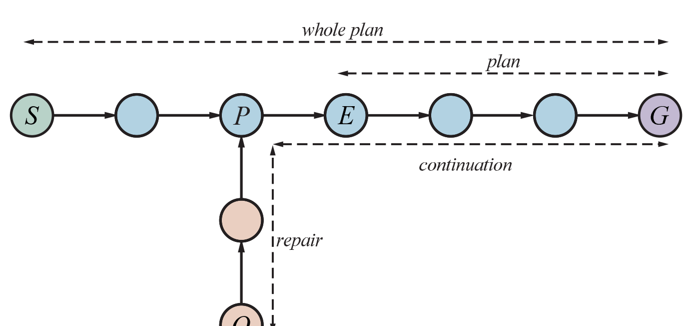
▷ **Example 19.8.3.** But when a tire blows while driving into the desert, we want to have water pre-planned. (due diligence against catastrophies)

▷ **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 19.8.4 (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 ⇝ Monitoring

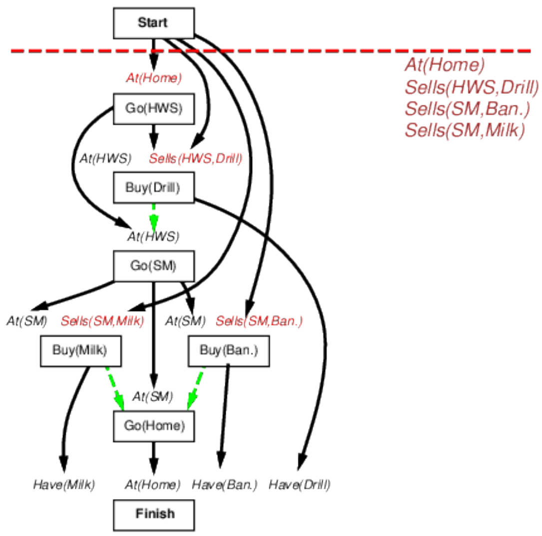
- **Generally:** The agent's world model can be incorrect, because
  - an action has a missing precondition (need a screwdriver for remove—lid)
  - an action misses an effect (painting a table gets paint on the floor)
  - it is missing a state variable (amount of paint in a can: no paint ⇝ no color)
  - no provisions for exogenous events (someone knocks over a paint can)
- **Observation:** Without a way for monitoring for these, planning is very brittle.
- **Definition 19.8.5.** 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:** ?? was a case of action monitoring leading to replanning.

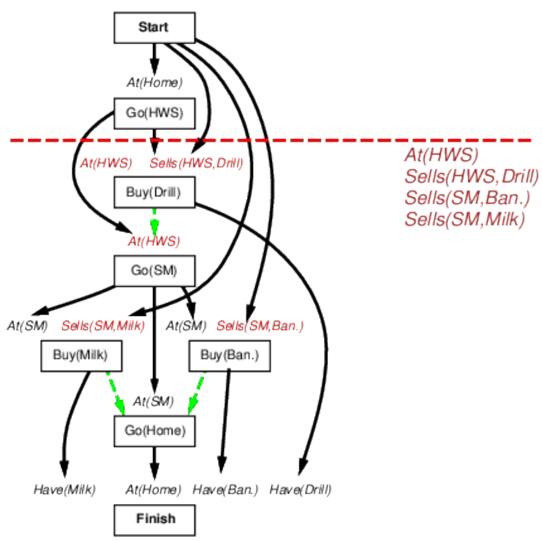
Integrated Execution Monitoring and Planning

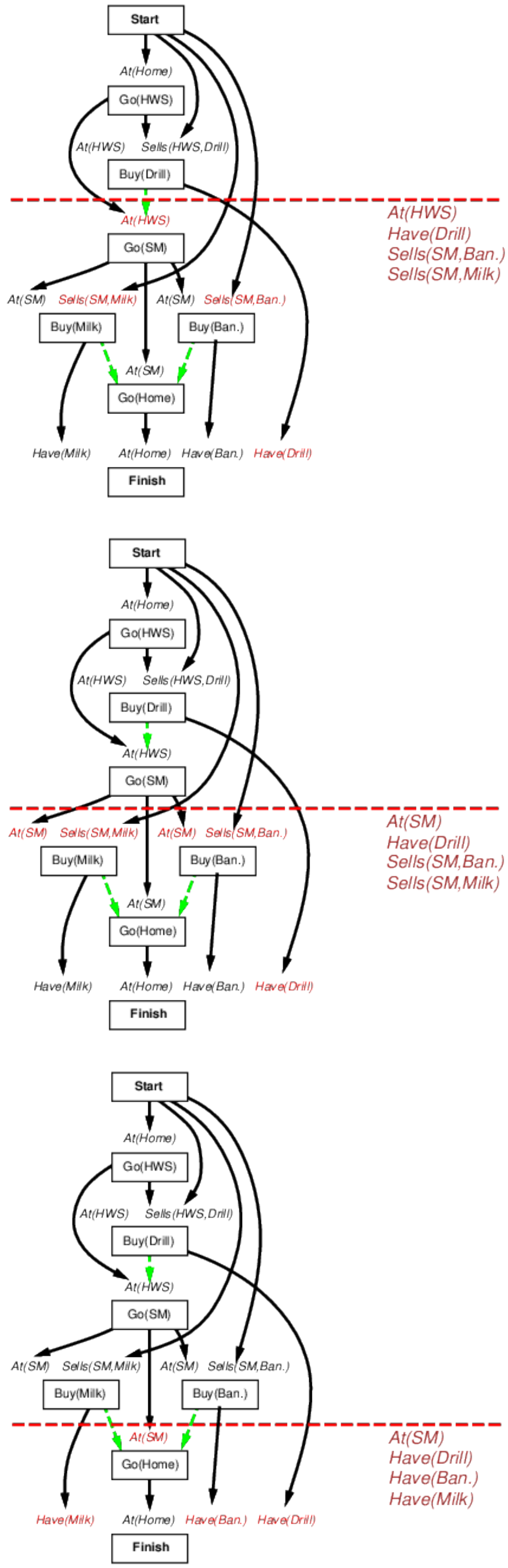
- **Problem:** Need to upgrade planing data structures by bookkeeping for execution monitoring.
- **Observation:** With their causal links, partially ordered plans already have most of the infrastructure for action monitoring:
  Preconditions of remaining plan
  $\widehat{=}$ all preconditions of remaining 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 19.8.6.** IPEM (Integrated Planning, Execution, and Monitoring):
  - keep updating $Start$ to match current state
  - links from actions replaced by links from $Start$ when done

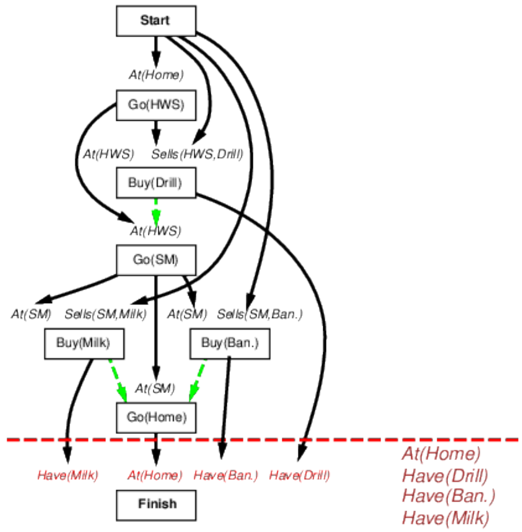
Execution Monitoring Example

▷ **Example 19.8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.






Start
At(Home)
Go(HWS)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM)
Sells(SM,Milk)
At(SM)
Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish
At(HWS)
Have(Drill)
Sells(SM,Ban.)
Sells(SM,Milk)
Start
At(Home)
Go(HWS)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM)
Sells(SM,Milk)
At(SM)
Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish
At(SM)
Have(Drill)
Sells(SM,Ban.)
Sells(SM,Milk)
Start
At(Home)
Go(HWS)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM)
Sells(SM,Milk)
At(SM)
Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish
At(SM)
Have(Drill)
Have(Ban.)
Have(Milk)


Start
At(Home)
Go(HWS)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM)
Sells(SM,Milk)
At(SM)
Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish
At(Home)
Have(Drill)
Have(Ban.)
Have(Milk)

Part V

What did we learn in AI 1?

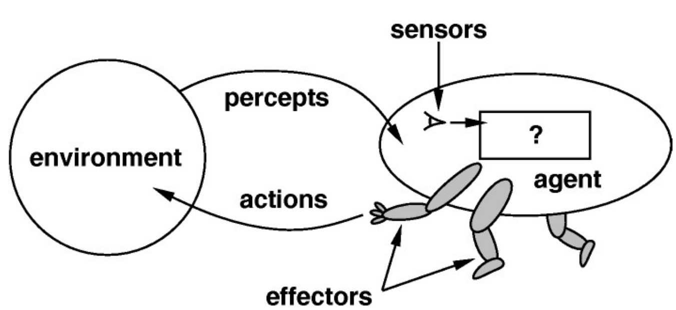
A Video Nugget covering this part can be found at `https://fau.tv/clip/id/26916`.

Topics of AI-1 (Winter Semester)

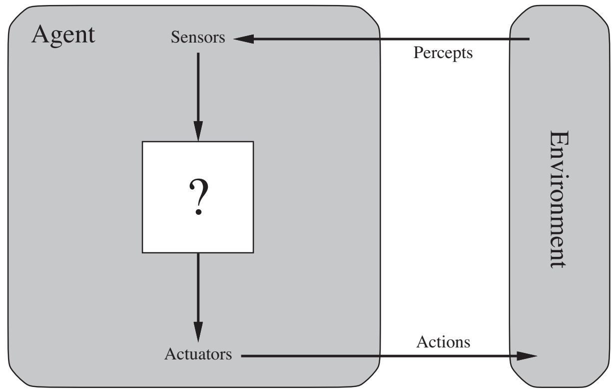
- Getting Started
  - What is Artificial Intelligence? (situating ourselves)
  - Logic programming in Prolog (An influential paradigm)
  - Intelligent Agents (a unifying framework)
- Problem Solving
  - Problem Solving and search (Black Box World States and Actions)
  - Adversarial search (Game playing)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

Rational Agents as an Evaluation Framework for AI

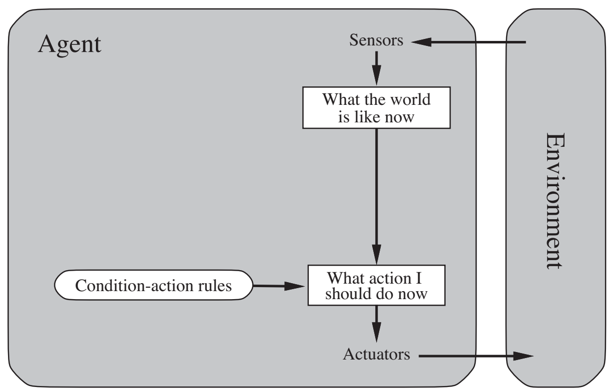
- Agents interact with the environment



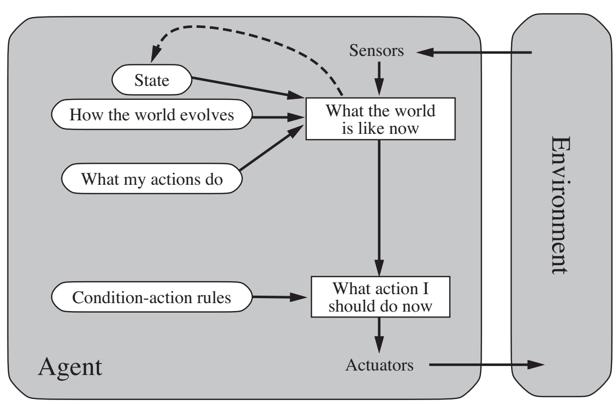
General agent schema



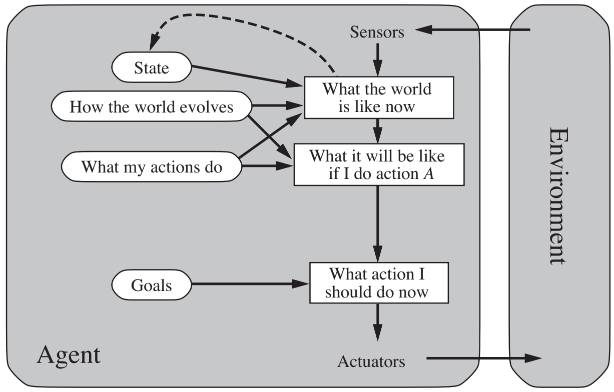
Simple Reflex Agents



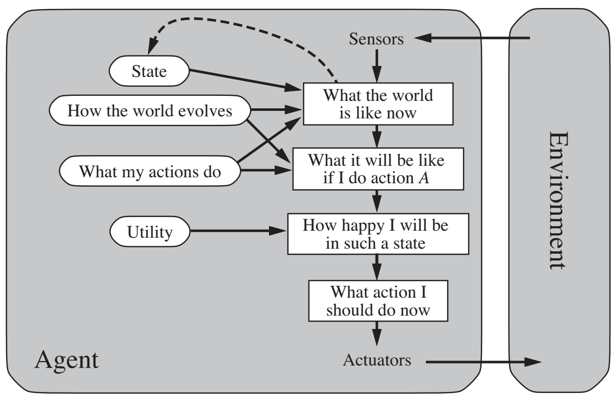
Reflex Agents with State



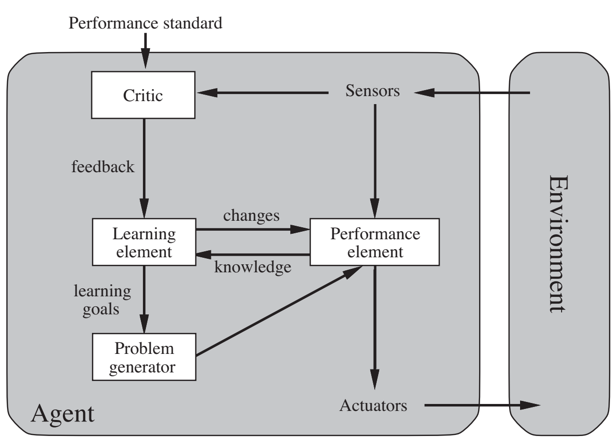
Goal-Based Agents



Utility-Based Agent



Learning Agents



Rational Agent

▷ **Idea:** Try to design agents that are successful (do the right thing)

- **Definition 19.8.8.** 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 ≠ omniscient)
    - need not predict e.g. very unlikely but catastrophic events in the future
  - percepts may not supply all relevant information (Rational ≠ 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 ≠ successful)
    - but we may need to take action to ensure that they do (more often) (learning)
- Rational ⇝ 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)
- 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

Bibliography

[Bac00] Fahiem Bacchus. *Subset of PDDL for the AIPS2000 Planning Competition.* The AIPS-00 Planning Competition Comitee. 2000.

[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.

[Bon+12] Blai Bonet et al., eds. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS’12).* AAAI Press, 2012.

[Bro90] Rodney Brooks. In: *Robotics and Autonomous Systems* 6.1–2 (1990), pp. 3–15. DOI: `10.1016/S0921-8890(05)80025-9`.

[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 12^{th} 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`.

[DCM12] DCMI Usage Board. *DCMI Metadata Terms.* DCMI Recommendation. Dublin Core Metadata Initiative, June 14, 2012. URL: `http://dublincore.org/documents/2012/06/14/dcmi-terms/`.

[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://saksagan.ceng.metu.edu.tr/courses/ceng242/documents/prolog/jrfisher/contents.html` (visited on 10/29/2024).

[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.

[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.

[Glo] *Grundlagen der Logik in der Informatik*. Course notes at `https://www8.cs.fau.de/_media/ws16:gloin:skript.pdf`. URL: `https://www8.cs.fau.de/_media/ws16:gloin:skript.pdf` (visited on 10/13/2017).

[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.

[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+13a] Ivan Herman et al. *RDF 1.1 Primer (Second Edition). Rich Structured Data Markup for Web Documents*. W3C Working Group Note. World Wide Web Consortium (W3C), 2013. URL: `http://www.w3.org/TR/rdfa-primer`.

[Her+13b] 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.

[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.

[Hit+12] Pascal Hitzler et al. *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), 2012. URL: `http://www.w3.org/TR/owl-primer`.

[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.

[Hof11] Jörg Hoffmann. "Every806thing You Always Wanted to Know about Planning (But Were Afraid to Ask)". In: *Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI'11)*. Ed. by Joscha Bach and Stefan Edelkamp. Vol. 7006. Lecture Notes in Computer Science. Springer, 2011, pp. 1–13. URL: `http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf`.

[ILD] *7. Constraints: Interpreting Line Drawings*. URL: `https://www.youtube.com/watch?v=l-tzjenXrvI&t=2037s` (visited on 11/19/2019).

[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/`.

[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.

[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`.

[Koh08] Michael Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: https://kwarc.info/kohlhase/papers/mcs08-stex.pdf.

[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.

[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.

[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.

[NS76] Alan Newell and Herbert A. Simon. "Computer Science as Empirical Inquiry: Symbols and Search". In: *Communications of the ACM* 19.3 (1976), pp. 113–126. DOI: 10.1145/360018.360022.

[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.

[Pro] *Protégé.* Project Home page at http://protege.stanford.edu. URL: http://protege.stanford.edu.

[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/.

[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.

[Ran17] Aarne Ranta. *Automatic Translation for Consumers and Producers.* Presentation given at the Chalmers Initiative Seminar. 2017. URL: https://www.grammaticalframework.org/~aarne/mt-digitalization-2017.pdf.

[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.

[RN95] Stuart J. Russell and Peter Norvig. *Artificial Intelligence — A Modern Approach.* Upper Saddle River, NJ: Prentice Hall, 1995.

[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.

[RW91] S. J. Russell and E. Wefald. *Do the Right Thing — Studies in limited Rationality.* MIT Press, 1991.

[She24] Esther Shein. 2024. URL: https://cacm.acm.org/news/the-impact-of-ai-on-computer-science-education/.

[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.

[Smu63] Raymond M. Smullyan. "A Unifying Principle for Quantification Theory". In: *Proc. Nat. Acad Sciences* 49 (1963), pp. 828–832.

[SR14] Guus Schreiber and Yves Raimond. *RDF 1.1 Primer*. W3C Working Group Note. World Wide Web Consortium (W3C), 2014. URL: http://www.w3.org/TR/rdf-primer.

[sTeX] *sTeX: A semantic Extension of TeX/LaTeX*. URL: https://github.com/sLaTeX/sTeX (visited on 05/11/2020).

[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.

[WHI] *Human intelligence — Wikipedia The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Human_intelligence (visited on 04/09/2018).

Part VI

Excursions

As this course is predominantly an overview over the topics of Artificial Intelligence, and not about the theoretical underpinnings, we give the discussion about these as a “suggested readings” part here.

Appendix A

Completeness of Calculi for Propositional Logic

The next step is to analyze the two calculi for completeness. For that we will first give ourselves a very powerful tool: the "model existence theorem" (**??**), which encapsulates the model-theoretic part of completeness theorems. With that, completeness proofs – which are quite tedious otherwise – become a breeze.

A.1 Abstract Consistency and Model Existence

We will now come to an important tool in the theoretical study of reasoning calculi: the "abstract consistency"/"model existence" method. This method for analyzing calculi was developed by Jaako Hintikka, Raymond Smullyan, and Peter Andrews in 1950-1970 as an encapsulation of similar constructions that were used in completeness arguments in the decades before. The basis for this method is Smullyan's Observation [Smu63] that completeness proofs based on Hintikka sets only certain properties of consistency and that with little effort one can obtain a generalization "Smullyan's Unifying Principle".

The basic intuition for this method is the following: typically, a logical system $\mathcal{L} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ has multiple calculi, human-oriented ones like the natural deduction calculi and machine-oriented ones like the automated theorem proving calculi. All of these need to be analyzed for completeness (as a basic quality assurance measure).

A completeness proof for a calculus $\mathcal{C}$ for $\mathcal{S}$ typically comes in two parts: one analyzes $\mathcal{C}$-consistency (sets that cannot be refuted in $\mathcal{C}$), and the other construct $\mathcal{K}$-models for $\mathcal{C}$-consistent sets.

In this situtation the "abstract consistency"/"model existence" method encapsulates the model construction process into a meta-theorem: the "model existence" theorem. This provides a set of syntactic ("abstract consistency") conditions for calculi that are sufficient to construct models.

With the model existence theorem it suffices to show that $\mathcal{C}$-consistency is an abstract consistency property (a purely syntactic task that can be done by a $\mathcal{C}$-proof transformation argument) to obtain a completeness result for $\mathcal{C}$.

Model Existence (Overview)

- ▷ **Definition:** Abstract consistency
- ▷ **Definition:** Hintikka set (maximally abstract consistent)
- ▷ **Theorem:** Hintikka sets are satisfiable

- ▷ **Theorem:** If Φ is abstract consistent, then Φ can be extended to a Hintikka set.
- ▷ **Corollary:** If Φ is abstract consistent, then Φ is satisfiable.
- ▷ **Application:** Let $\mathcal{C}$ be a calculus, if Φ is $\mathcal{C}$-consistent, then Φ is abstract consistent.
- ▷ **Corollary:** $\mathcal{C}$ is complete.

The proof of the model existence theorem goes via the notion of a Hintikka set, a set of formulae with very strong syntactic closure properties, which allow to read off models. Jaako Hintikka's original idea for completeness proofs was that for every complete calculus $\mathcal{C}$ and every $\mathcal{C}$-consistent set one can induce a Hintikka set, from which a model can be constructed. This can be considered as a first model existence theorem. However, the process of obtaining a Hintikka set for a $\mathcal{C}$-consistent set Φ of sentences usually involves complicated calculus dependent constructions.

In this situation, Raymond Smullyan was able to formulate the sufficient conditions for the existence of Hintikka sets in the form of "abstract consistency properties" by isolating the calculus independent parts of the Hintikka set construction. His technique allows to reformulate Hintikka sets as maximal elements of abstract consistency classes and interpret the Hintikka set construction as a maximizing limit process.

To carry out the "model-existence"/"abstract consistency" method, we will first have to look at the notion of consistency.

Consistency and refutability are very important notions when studying the completeness for calculi; they form syntactic counterparts of satisfiability.

Consistency

- ▷ Let $\mathcal{C}$ be a calculus,...
- ▷ **Definition A.1.1.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-refutable, if there is a refutation, i.e. a derivation of a contradiction from Φ. The act of finding a refutation for Φ is called refuting Φ.
- ▷ **Definition A.1.2.** We call a pair of formulae $\mathbf{A}$ and $\neg\mathbf{A}$ a contradiction.
- ▷ So a set Φ is $\mathcal{C}$-refutable, if $\mathcal{C}$ canderive a contradiction from it.
- ▷ **Definition A.1.3.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-consistent, iff there is a formula $\mathbf{B}$, that is not derivable from Φ in $\mathcal{C}$.
- ▷ **Definition A.1.4.** We call a calculus $\mathcal{C}$ reasonable, iff implication elimination and conjunction introduction are admissible in $\mathcal{C}$ and $\mathbf{A} \wedge \neg\mathbf{A} \Rightarrow \mathbf{B}$ is a $\mathcal{C}$-theorem.
- ▷ **Theorem A.1.5.** *$\mathcal{C}$-inconsistency and $\mathcal{C}$-refutability coincide for reasonable calculi.*

It is very important to distinguish the syntactic $\mathcal{C}$-refutability and $\mathcal{C}$-consistency from satisfiability, which is a property of formulae that is at the heart of semantics. Note that the former have the calculus (a syntactic device) as a parameter, while the latter does not. In fact we should actually say $\mathcal{S}$-satisfiability, where $\langle\mathcal{L}, \mathcal{K}, \models\rangle$ is the current logical system.

Even the word "contradiction" has a syntactical flavor to it, it translates to "saying against each other" from its Latin root.

Abstract Consistency

- **Definition A.1.6.** Let ∇ be a collection of sets. We call ∇ closed under subsets, iff for each $\Phi \in \nabla$, all subsets $\Psi \subseteq \Phi$ are elements of ∇.
- **Definition A.1.7 (Notation).** We will use $\Phi{*}\mathbf{A}$ for $\Phi \cup \{\mathbf{A}\}$.
- **Definition A.1.8.** A collection ∇ of sets of propositional formulae is called an abstract consistency class, iff it is closed under subsets, and for each $\Phi \in \nabla$

  ∇_c) $P \notin \Phi$ or $\neg P \notin \Phi$ for $P \in \mathcal{V}_0$

  $\nabla_\neg$) $\neg\neg\mathbf{A} \in \Phi$ implies $\Phi{*}\mathbf{A} \in \nabla$

  $\nabla_\vee$) $\mathbf{A} \vee \mathbf{B} \in \Phi$ implies $\Phi{*}\mathbf{A} \in \nabla$ or $\Phi{*}\mathbf{B} \in \nabla$

  $\nabla_\wedge$) $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$ implies $\Phi \cup \{\neg\mathbf{A}, \neg\mathbf{B}\} \in \nabla$
- **Example A.1.9.** The empty set is an abstract consistency class.
- **Example A.1.10.** The set $\{\emptyset, \{Q\}, \{P \vee Q\}, \{P \vee Q, Q\}\}$ is an abstract consistency class.
- **Example A.1.11.** The family of satisfiable sets is an abstract consistency class.

So a family of sets (we call it a family, so that we do not have to say "set of sets" and we can distinguish the levels) is an abstract consistency class, iff it fulfills five simple conditions, of which the last three are closure conditions.

Think of an abstract consistency class as a family of "consistent" sets (e.g. $\mathcal{C}$-consistent for some calculus $\mathcal{C}$), then the properties make perfect sense: They are naturally closed under subsets — if we cannot derive a contradiction from a large set, we certainly cannot from a subset, furthermore,

∇_c) If both $P \in \Phi$ and $\neg P \in \Phi$, then Φ cannot be "consistent".

$\nabla_\neg$) If we cannot derive a contradiction from Φ with $\neg\neg\mathbf{A} \in \Phi$ then we cannot from $\Phi{*}\mathbf{A}$, since they are logically equivalent.

The other two conditions are motivated similarly. We will carry out the proof here, since it gives us practice in dealing with the abstract consistency properties.

The main result here is that abstract consistency classes can be extended to compact ones. The proof is quite tedious, but relatively straightforward. It allows us to assume that all abstract consistency classes are compact in the first place (otherwise we pass to the compact extension).

Actually we are after abstract consistency classes that have an even stronger property than just being closed under subsets. This will allow us to carry out a limit construction in the Hintikka set extension argument later.

Compact Collections

- **Definition A.1.12.** We call a collection ∇ of sets compact, iff for any set Φ we have

  $\Phi \in \nabla$, iff $\Psi \in \nabla$ for every finite subset Ψ of Φ.
- **Lemma A.1.13.** *If ∇ is compact, then ∇ is closed under subsets.*
- *Proof:*

1. Suppose $S \subseteq T$ and $T \in \nabla$.
2. Every finite subset A of S is a finite subset of T.
3. As ∇ is compact, we know that $A \in \nabla$.
4. Thus $S \in \nabla$.

The property of being closed under subsets is a "downwards-oriented" property: We go from large sets to small sets, compactness (the interesting direction anyways) is also an "upwards-oriented" property. We can go from small (finite) sets to large (infinite) sets. The main application for the compactness condition will be to show that infinite sets of formulae are in a collection ∇ by testing all their finite subsets (which is much simpler).

Compact Abstract Consistency Classes

▷ **Lemma A.1.14.** *Any abstract consistency class can be extended to a compact one.*

▷ *Proof:*

1. We choose $\nabla' := \{\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0) \,|\, \text{every finite subset of } \Phi \text{ is in } \nabla\}$.
2. Now suppose that $\Phi \in \nabla$. ∇ is closed under subsets, so every finite subset of Φ is in ∇ and thus $\Phi \in \nabla'$. Hence $\nabla \subseteq \nabla'$.
3. Next let us show that each ∇ is compact.'
   - 3.1. Suppose $\Phi \in \nabla'$ and Ψ is an arbitrary finite subset of Φ.
   - 3.2. By definition of ∇' all finite subsets of Φ are in ∇ and therefore $\Psi \in \nabla'$.
   - 3.3. Thus all finite subsets of Φ are in ∇' whenever Φ is in ∇'.
   - 3.4. On the other hand, suppose all finite subsets of Φ are in ∇'.
   - 3.5. Then by the definition of ∇' the finite subsets of Φ are also in ∇, so $\Phi \in \nabla'$. Thus ∇' is compact.
4. Note that ∇' is closed under subsets by the Lemma above.
5. Now we show that if ∇ satisfies ∇_*, then ∇ satisfies ∇_*.'
   - 5.1. To show ∇_c, let $\Phi \in \nabla'$ and suppose there is an atom $\mathbf{A}$, such that $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$. Then $\{\mathbf{A}, \neg\mathbf{A}\} \in \nabla$ contradicting ∇_c.
   - 5.2. To show $\nabla_\neg$, let $\Phi \in \nabla'$ and $\neg\neg\mathbf{A} \in \Phi$, then $\Phi{*}\mathbf{A} \in \nabla'$.
     - 5.2.1. Let Ψ be any finite subset of $\Phi{*}\mathbf{A}$, and $\Theta := (\Psi \backslash \{\mathbf{A}\}){*}\neg\neg\mathbf{A}$.
     - 5.2.2. Θ is a finite subset of Φ, so $\Theta \in \nabla$.
     - 5.2.3. Since ∇ is an abstract consistency class and $\neg\neg\mathbf{A} \in \Theta$, we get $\Theta{*}\mathbf{A} \in \nabla$ by $\nabla_\neg$.
     - 5.2.4. We know that $\Psi \subseteq \Theta{*}\mathbf{A}$ and ∇ is closed under subsets, so $\Psi \in \nabla$.
     - 5.2.5. Thus every finite subset Ψ of $\Phi{*}\mathbf{A}$ is in ∇ and therefore by definition $\Phi{*}\mathbf{A} \in \nabla'$.
   - 5.3. the other cases are analogous to $\nabla_\neg$.

Hintikka sets are sets of sentences with very strong analytic closure conditions. These are motivated as maximally consistent sets i.e. sets that already contain everything that can be consistently added to them.

∇-Hintikka Set

- **Definition A.1.15.** Let ∇ be an abstract consistency class, then we call a set $\mathcal{H} \in \nabla$ a ∇ Hintikka Set, iff $\mathcal{H}$ is maximal in ∇, i.e. for all $\mathbf{A}$ with $\mathcal{H}*\mathbf{A} \in \nabla$ we already have $\mathbf{A} \in \mathcal{H}$.
- **Theorem A.1.16 (Hintikka Properties).** *Let ∇ be an abstract consistency class and $\mathcal{H}$ be a ∇-Hintikka set, then*

  $\mathcal{H}_c$) *For all $\mathbf{A} \in wff_0(\mathcal{V}_0)$ we have $\mathbf{A} \notin \mathcal{H}$ or $\neg\mathbf{A} \notin \mathcal{H}$*

  $\mathcal{H}_\neg$) *If $\neg\neg\mathbf{A} \in \mathcal{H}$ then $\mathbf{A} \in \mathcal{H}$*

  $\mathcal{H}_\vee$) *If $\mathbf{A} \vee \mathbf{B} \in \mathcal{H}$ then $\mathbf{A} \in \mathcal{H}$ or $\mathbf{B} \in \mathcal{H}$*

  $\mathcal{H}_\wedge$) *If $\neg(\mathbf{A} \vee \mathbf{B}) \in \mathcal{H}$ then $\neg\mathbf{A}, \neg\mathbf{B} \in \mathcal{H}$*

∇-Hintikka Set

- *Proof:*

  We prove the properties in turn

  1. $\mathcal{H}_c$ *by induction on the structure of* $\mathbf{A}$
     1.1. $\mathbf{A} \in \mathcal{V}_0$ Then $\mathbf{A} \notin \mathcal{H}$ or $\neg\mathbf{A} \notin \mathcal{H}$ by ∇_c.
     1.2. $\mathbf{A} = \neg\mathbf{B}$
        1.2.1. Let us assume that $\neg\mathbf{B} \in \mathcal{H}$ and $\neg\neg\mathbf{B} \in \mathcal{H}$,
        1.2.2. then $\mathcal{H}*\mathbf{B} \in \nabla$ by $\nabla_\neg$, and therefore $\mathbf{B} \in \mathcal{H}$ by maximality.
        1.2.3. So both $\mathbf{B}$ and $\neg\mathbf{B}$ are in $\mathcal{H}$, which contradicts the induction hypothesis.
     1.3. $\mathbf{A} = \mathbf{B} \vee \mathbf{C}$ *similar to the previous case*
  2. We prove $\mathcal{H}_\neg$ by maximality of $\mathcal{H}$ in ∇.
     2.1. If $\neg\neg\mathbf{A} \in \mathcal{H}$, then $\mathcal{H}*\mathbf{A} \in \nabla$ by $\nabla_\neg$.
     2.2. The maximality of $\mathcal{H}$ now gives us that $\mathbf{A} \in \mathcal{H}$.

  Proof sketch: *other $\mathcal{H}_*$ are similar*

The following theorem is one of the main results in the “abstract consistency”/”model existence” method. For any abstract consistent set Φ it allows us to construct a Hintikka set $\mathcal{H}$ with $\Phi \in \mathcal{H}$.

Extension Theorem

- **Theorem A.1.17.** *If ∇ is an abstract consistency class and $\Phi \in \nabla$, then there is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$.*
- *Proof:*

  1. Wlog. we assume that ∇ is compact (otherwise pass to compact extension)
  2. We choose an enumeration $\mathbf{A}_1, \ldots$ of the set $wff_0(\mathcal{V}_0)$
  3. and construct a sequence of sets $\mathbf{H}_i$ with $\mathbf{H}_0 := \Phi$ and

  $$\mathbf{H}_{n+1} := \begin{cases} \mathbf{H}_n & \text{if } \mathbf{H}_n*\mathbf{A}_n \notin \nabla \\ \mathbf{H}_n*\mathbf{A}_n & \text{if } \mathbf{H}_n*\mathbf{A}_n \in \nabla \end{cases}$$

  4. Note that all $\mathbf{H}_i \in \nabla$, choose $\mathcal{H} := \bigcup_{i\in\mathbb{N}} \mathbf{H}_i$

5. $\Psi \subseteq \mathcal{H}$ finite implies there is a $j \in \mathbb{N}$ such that $\Psi \subseteq \mathbf{H}_j$,
6. so $\Psi \in \nabla$ as ∇ is closed under subsets and $\mathcal{H} \in \nabla$ as ∇ is compact.
7. Let $\mathcal{H}*\mathbf{B} \in \nabla$, then there is a $j \in \mathbb{N}$ with $\mathbf{B} = \mathbf{A}_j$, so that $\mathbf{B} \in \mathbf{H}_{j+1}$ and $\mathbf{H}_{j+1} \subseteq \mathcal{H}$
8. Thus $\mathcal{H}$ is ∇-maximal

Note that the construction in the proof above is non-trivial in two respects. First, the limit construction for $\mathcal{H}$ is not executed in our original abstract consistency class ∇, but in a suitably extended one to make it compact — the original would not have contained $\mathcal{H}$ in general. Second, the set $\mathcal{H}$ is not unique for Φ, but depends on the choice of the enumeration of $\mathit{wff}_0(\mathcal{V}_0)$. If we pick a different enumeration, we will end up with a different $\mathcal{H}$. Say if $\mathbf{A}$ and $\neg\mathbf{A}$ are both ∇-consistent[1] with Φ, then depending on which one is first in the enumeration $\mathcal{H}$, will contain that one; with all the consequences for subsequent choices in the construction process.

Valuation

▷ **Definition A.1.18.** A function $\nu\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is called a (propositional) valuation, iff

- ▷ $\nu(\neg\mathbf{A}) = \mathsf{T}$, iff $\nu(\mathbf{A}) = \mathsf{F}$
- ▷ $\nu(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$, iff $\nu(\mathbf{A}) = \mathsf{T}$ and $\nu(\mathbf{B}) = \mathsf{T}$

▷ **Lemma A.1.19.** *If $\nu\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is a valuation and $\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ with $\nu(\Phi) = \{\mathsf{T}\}$, then Φ is satisfiable.*

▷ *Proof sketch:* $\nu|_{\mathcal{V}_0} : \mathcal{V}_0 \to \mathcal{D}_o$ is a satisfying variable assignment.

▷ **Lemma A.1.20.** *If $\varphi\colon \mathcal{V}_0 \to \mathcal{D}_o$ is a variable assignment, then $\mathcal{I}_\varphi\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is a valuation.*

Now, we only have to put the pieces together to obtain the model existence theorem we are after.

Model Existence

▷ **Lemma A.1.21 (Hintikka-Lemma).** *If ∇ is an abstract consistency class and $\mathcal{H}$ a ∇-Hintikka set, then $\mathcal{H}$ is satisfiable.*

▷ *Proof:*

1. We define $\nu(\mathbf{A}) := \mathsf{T}$, iff $\mathbf{A} \in \mathcal{H}$
2. then ν is a valuation by the Hintikka properties
3. and thus $\nu|_{\mathcal{V}_0}$ is a satisfying assignment.

▷ **Theorem A.1.22 (Model Existence).** *If ∇ is an abstract consistency class and $\Phi \in \nabla$, then Φ is satisfiable.*

Proof:

▷ 1. There is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$ (Extension Theorem)
2. We know that $\mathcal{H}$ is satisfiable. (Hintikka-Lemma)

[1] EdNote: introduce this above

3. In particular, $\Phi \subseteq \mathcal{H}$ is satisfiable.

A.2 A Completeness Proof for Propositional Tableaux

With the model existence proof we have introduced in the last section, the completeness proof for first-order natural deduction is rather simple, we only have to check that Tableaux-consistency is an abstract consistency property.

We encapsulate all of the technical difficulties of the problem in a technical Lemma. From that, the completeness proof is just an application of the high-level theorems we have just proven.

Abstract Completeness for $\mathcal{T}_0$

▷ **Lemma A.2.1.** $\{\Phi \mid \Phi^{\mathsf{T}}$ *has no closed tableau*$\}$ *is an abstract consistency class.*

▷ *Proof:* Let's call the set above ∇

We have to convince ourselves of the abstract consistency properties

1. $\nabla_c P, \neg P \in \Phi$ *implies* $P^{\mathsf{F}}, P^{\mathsf{T}} \in \Phi^{\mathsf{T}}$.
2. $\nabla_\neg$ *Let* $\neg\neg\mathbf{A} \in \Phi$.
   - 2.1. For the proof of the contrapositive we assume that $\Phi{*}\mathbf{A}$ has a closed tableau $\mathcal{T}$ and show that already Φ has one:
   - 2.2. applying each of $\mathcal{T}_0\neg^{\mathsf{T}}$ and $\mathcal{T}_0\neg^{\mathsf{F}}$ once allows to extend any tableau with $\neg\neg\mathbf{B}^\alpha$ by $\mathbf{B}^\alpha$.
   - 2.3. any path in $\mathcal{T}$ that is closed with $\neg\neg\mathbf{A}^\alpha$, can be closed by $\mathbf{A}^\alpha$.
3. $\nabla_\vee$ *Suppose* $\mathbf{A} \vee \mathbf{B} \in \Phi$ *and both* $\Phi{*}\mathbf{A}$ *and* $\Phi{*}\mathbf{B}$ *have closed tableaux*
   - 3.1. consider the tableaux:

$$\begin{array}{c}\Phi^{\mathsf{T}}\\ \mathbf{A}^{\mathsf{T}}\\ Rest^1\end{array} \qquad \begin{array}{c}\Phi^{\mathsf{T}}\\ \mathbf{B}^{\mathsf{T}}\\ Rest^2\end{array} \qquad \begin{array}{c}\Psi^{\mathsf{T}}\\ (\mathbf{A}\vee\mathbf{B})^{\mathsf{T}}\\ \begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{B}^{\mathsf{T}}\\ Rest^1 & Rest^2\end{array}\end{array}$$

4. $\nabla_\wedge$ *suppose,* $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$ *and* $\Phi\{\neg\mathbf{A}, \neg\mathbf{B}\}$ *have closed tableau* $\mathcal{T}$.
   - 4.1. We consider

$$\begin{array}{c}\Phi^{\mathsf{T}}\\ \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}}\\ Rest\end{array} \qquad \begin{array}{c}\Psi^{\mathsf{T}}\\ (\mathbf{A}\vee\mathbf{B})^{\mathsf{F}}\\ \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}}\\ Rest\end{array}$$

where $\Phi = \Psi{*}\neg(\mathbf{A} \vee \mathbf{B})$.

Observation: If we look at the completeness proof below, we see that the Lemma above is the only place where we had to deal with specific properties of the $\mathcal{T}_0$.

So if we want to prove completeness of any other calculus with respect to propositional logic, then we only need to prove an analogon to this lemma and can use the rest of the machinery we have already established "off the shelf".

This is one great advantage of the "abstract consistency method"; the other is that the method can be extended transparently to other logics.

Completeness of $\mathcal{T}_0$

▷ **Corollary A.2.2.** *$\mathcal{T}_0$ is complete.*

▷ *Proof:* by contradiction

1. We assume that $\mathbf{A} \in wff_0(\mathcal{V}_0)$ is valid, but there is no closed tableau for $\mathbf{A}^{\mathsf{F}}$.
2. We have $\{\neg\mathbf{A}\} \in \nabla$ as $\neg\mathbf{A}^{\mathsf{T}} = \mathbf{A}^{\mathsf{F}}$.
3. so $\neg\mathbf{A}$ is satisfiable by the model existence theorem (which is applicable as ∇ is an abstract consistency class by our Lemma above)
4. this contradicts our assumption that $\mathbf{A}$ is valid.

Appendix B

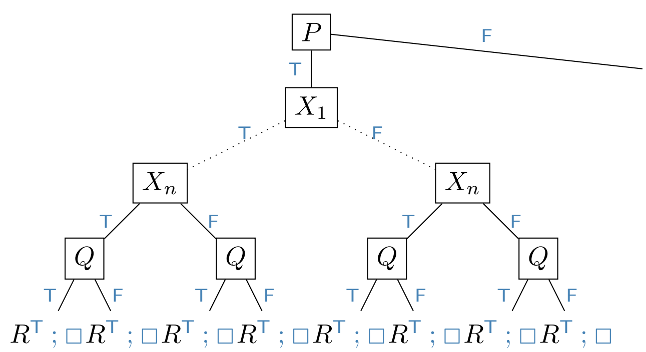
Conflict Driven Clause Learning

B.1 *Why* Did Unit Propagation Yield a Conflict?

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/27026`.

DPLL: Example (Redundance1)

▷ **Example B.1.1.** We introduce some nasty redundance to make DPLL slow.
$\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
DPLL on $\Delta ; \Theta$ with $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}}$



How To *Not* Make the Same Mistakes Over Again?

▷ **It's not that difficult, really:**

(A) Figure out what went wrong.

(B) Learn to not do that again in the future.

▷ **And now for DPLL:**

(A) *Why* did unit propagation yield a Conflict?

- ▷ This Section. We will capture the "what went wrong" in terms of graphs over literals set during the search, and their dependencies.

▷ **What can we learn from that information?:**

- ▷ A new clause! Next section.

Implication Graphs for DPLL

▷ **Definition B.1.2.** Let β be a branch in a DPLL derivation and P a variable on β then we call

- ▷ P^α a choice literal if its value is set to α by the splitting rule.
- ▷ P^α an implied literal, if the value of P is set to α by the UP rule.
- ▷ P^α a conflict literal, if it contributes to a derivation of the empty clause.

▷ **Definition B.1.3 (Implication Graph).**

Let Δ be a clause set, β a DPLL search branch on Δ. The implication graph G_β^{impl} is the directed graph whose vertices are labeled with the choice and implied literals along β, as well as a separate conflict vertex $\Box_C$ for every clause C that became empty on β.

Whereever a clause $l_1, \ldots, l_k \vee l' \in \Delta$ became unit with implied literal l', G_β^{impl} includes the edges $(\overline{l_i}, l')$.

Where $C = l_1 \vee \ldots \vee l_k \in \Delta$ became empty, G_β^{impl} includes the edges $(\overline{l_i}, \Box_C)$.

▷ **Question:** How do we know that $\overline{l_i}$ are vertices in G_β^{impl}?

▷ **Answer:** Because $l_1 \vee \ldots \vee l_k \vee l'$ became unit/empty.

▷ **Observation B.1.4.** G_β^{impl} *is acyclic.*

▷ *Proof sketch:* UP can't derive l' whose value was already set beforehand.

▷ **Intuition:** The initial vertices are the choice literals and unit clauses of Δ.

Implication Graphs: Example (Vanilla1) in Detail

▷ **Example B.1.5.** Let $\Delta := P^{\mathsf{T}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} ; R^{\mathsf{T}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}}$.

We look at the left branch of the derivation from **??**:

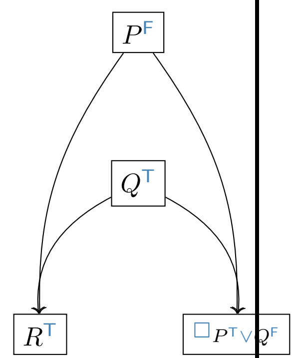
1. UP Rule: $R \mapsto \mathsf{T}$
   Implied literal R^{T}.
   $P^{\mathsf{T}} \vee Q^{\mathsf{T}}$; $P^{\mathsf{F}} \vee Q^{\mathsf{F}}$; $P^{\mathsf{T}} \vee Q^{\mathsf{F}}$

2. Splitting Rule:

   2a. $P \mapsto \mathsf{F}$
   Choice literal P^{F}.
   Q^{T} ; Q^{F}

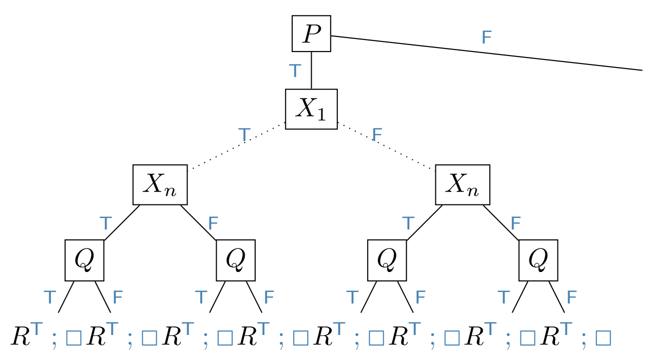
   3a. UP Rule: $Q \mapsto \mathsf{T}$
   Implied literal Q^{T}
   edges $(R^{\mathsf{T}}, Q^{\mathsf{T}})$ and $(P^{\mathsf{F}}, Q^{\mathsf{T}})$.
   $\Box$
   Conflict vertex $\Box_{P^{\mathsf{T}} \vee Q^{\mathsf{F}}}$
   edges $(P^{\mathsf{F}}, \Box_{P^{\mathsf{T}} \vee Q^{\mathsf{F}}})$ and $(Q^{\mathsf{T}}, \Box_{P^{\mathsf{T}} \vee Q^{\mathsf{F}}})$.

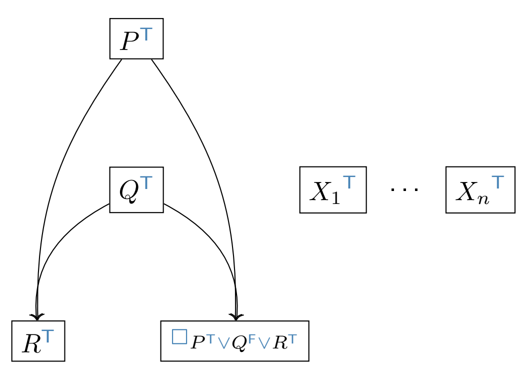
Implication graph:



Implication Graphs: Example (Redundance1)

▷ **Example B.1.6.** Continuing from ??: $\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
DPLL on $\Delta ; \Theta$ with $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}}$
Choice literals: P^{T}, $({X_1}^{\mathsf{T}}), \ldots, ({X_n}^{\mathsf{T}})$, Q^{T}. Implied literal: R^{T}.



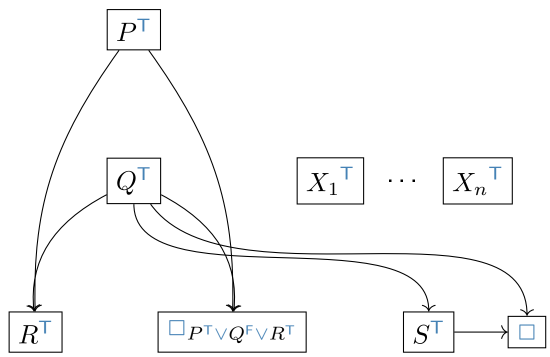


Implication Graphs: Example (Redundance2)

▷ **Example B.1.7.** Continuing from ??:

$$\begin{array}{lcl} \Delta & := & P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}} \\ \Theta & := & {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}} \end{array}$$

DPLL on $\Delta ; \Theta ; \Phi$ with $\Phi := Q^{\mathsf{F}} \vee S^{\mathsf{T}} ; Q^{\mathsf{F}} \vee S^{\mathsf{F}}$
Choice literals: P^{T}, $({X_1}^{\mathsf{T}}), \ldots, ({X_n}^{\mathsf{T}})$, Q^{T}. Implied literals:

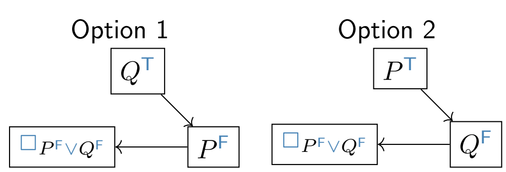


Implication Graphs: A Remark

▷ The implication graph is *not* uniquely determined by the Choice literals.

▷ It depends on "ordering decisions" during UP: Which unit clause is picked first.

▷ **Example B.1.8.** $\Delta = P^{\mathsf{F}} \vee Q^{\mathsf{F}} ; Q^{\mathsf{T}} ; P^{\mathsf{T}}$



Conflict Graphs

▷ A conflict graph captures "what went wrong" in a failed node.

▷ **Definition B.1.9 (Conflict Graph).** Let Δ be a clause set, and let G_β^{impl} be the implication graph for some search branch β of DPLL on Δ. A subgraph C of G_β^{impl} is a conflict graph if:

(i) C contains exactly one conflict vertex $\Box_C$.

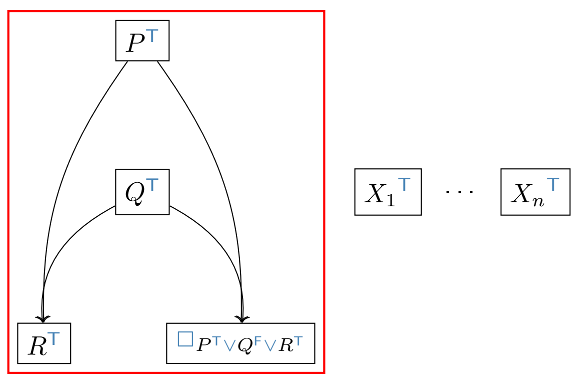
(ii) If l' is a vertex in C, then all parents of l', i.e. vertices $\overline{l_i}$ with a I edge $(\overline{l_i},l')$, are vertices in C as well.

(iii) All vertices in C have a path to $\Box_C$.

▷ Conflict graph $\widehat{=}$ Starting at a conflict vertex, backchain through the implication graph until reaching choice literals.

Conflict-Graphs: Example (Redundance1)

▷ **Example B.1.10.** Continuing from ??: $\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
DPLL on $\Delta ; \Theta$ with $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_{100}}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_{100}}^{\mathsf{F}}$
Choice literals: P^{T}, $({X_1}^{\mathsf{T}}), \ldots, ({X_{100}}^{\mathsf{T}})$, Q^{T}. Implied literals: R^{T}.

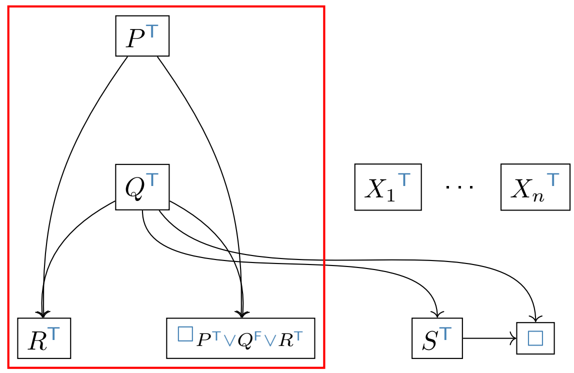


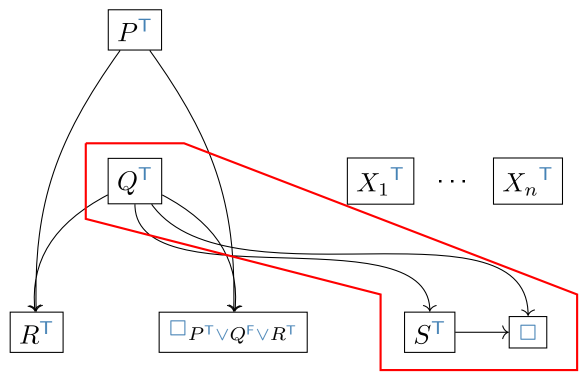
Conflict Graphs: Example (Redundance2)

▷ **Example B.1.11.** Continuing from ?? and ??:

$$\begin{array}{lll} \Delta & := & P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}} \\ \Theta & := & {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}} \end{array}$$

DPLL on $\Delta ; \Theta ; \Phi$ with $\Phi := Q^{\mathsf{F}} \vee S^{\mathsf{T}} ; Q^{\mathsf{F}} \vee S^{\mathsf{F}}$
Choice literals: P^{T}, $({X_1}^{\mathsf{T}}), \ldots, ({X_n}^{\mathsf{T}})$, Q^{T}. Implied literals: R^{T}.





B.2 Clause Learning

Clause Learning

▷ **Observation:** Conflict graphs encode the entailment relation.

▷ **Definition B.2.1.** Let Δ be a clause set, C be a conflict graph at some time point during a run of DPLL on Δ, and L be the choice literals in C, then we call $c := \bigvee_{l \in L} \bar{l}$ the learned clause for C.

▷ **Theorem B.2.2.** *Let Δ, C, and c as in ??, then $\Delta \vDash c$.*

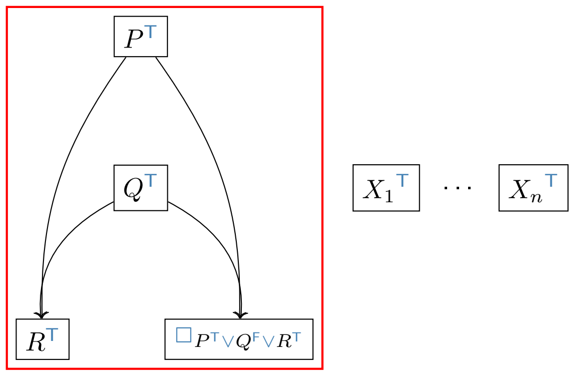
▷ **Idea:** We can add learned clauses to DPLL derivations at any time without losing soundness. (maybe this helps, if we have a good notion of learned clauses)

▷ **Definition B.2.3.** Clause learning is the process of adding learned clauses to DPLL clause sets at specific points. (details coming up)

Clause Learning: Example (Redundance1)

▷ **Example B.2.4.** Continuing from ??:

$\Delta := P^F \vee Q^F \vee R^T ; P^F \vee Q^F \vee R^F ; P^F \vee Q^T \vee R^T ; P^F \vee Q^T \vee R^F$
DPLL on $\Delta ; \Theta$ with $\Theta := X_1{}^T \vee \ldots \vee X_n{}^T ; X_1{}^F \vee \ldots \vee X_n{}^F$
Choice literals: P^T, $(X_1{}^T), \ldots, (X_n{}^T)$, Q^T. Implied literals: R^T.



Learned clause: $P^F \vee Q^F$

The Effect of Learned Clauses (in Redundance1)

▷ What happens after we learned a new clause C?

1. We add C into Δ. e.g. $C = P^F \vee Q^F$.
2. We retract the last choice l'. e.g. the choice $l' = Q$.

▷ **Observation:** Let C be a learned clause, i.e. $C = \bigvee_{l \in L} \bar{l}$, where L is the set of conflict literals in a conflict graph G.

Before we learn C, G must contain the most recent choice l': otherwise, the conflict would have occured earlier on.

So $C = l_1{}^T \vee \ldots \vee l_k{}^T \vee \overline{l'}$ where $l_1, \ldots, l_k$ are earlier choices.

▷ **Example B.2.5.** $l_1 = P$, $C = P^F \vee Q^F$, $l' = Q$.

▷ **Observation:** Given the earlier choices $l_1, \ldots, l_k$, after we learned the new clause $C = \overline{l_1} \vee \ldots \vee \overline{l_k} \vee \overline{l'}$, the value of $\overline{l'}$ is now set by UP!

▷ So we can continue:

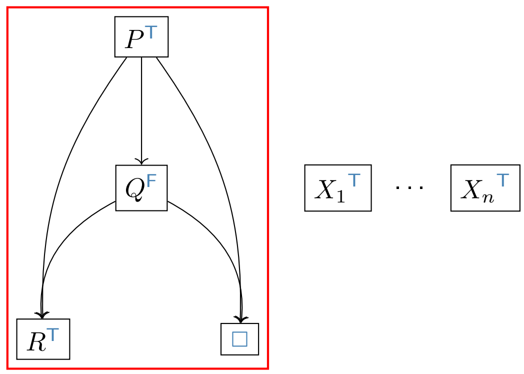
3. We set the opposite choice $\overline{l'}$ as an implied literal.
   e.g. Q^F as an implied literal.
4. We run UP and analyze conflicts.
   Learned clause: earlier choices only! e.g. $C = P^F$, see next slide.

The Effect of Learned Clauses: Example (Redundance1)

▷ **Example B.2.6.** Continuing from ??:

$$\Delta := P^F \lor Q^F \lor R^T ; P^F \lor Q^F \lor R^F ; P^F \lor Q^T \lor R^T ; P^F \lor Q^T \lor R^F$$
$$\Theta := X_1{}^T \lor \ldots \lor X_{100}{}^T ; X_1{}^F \lor \ldots \lor X_{100}{}^F$$

DPLL on $\Delta ; \Theta ; \Phi$ with $\Phi := P^F \lor Q^F$
Choice literals: P^T, $(X_1{}^T), \ldots, (X_{100}{}^T)$, Q^T. Implied literals: Q^F, R^T.



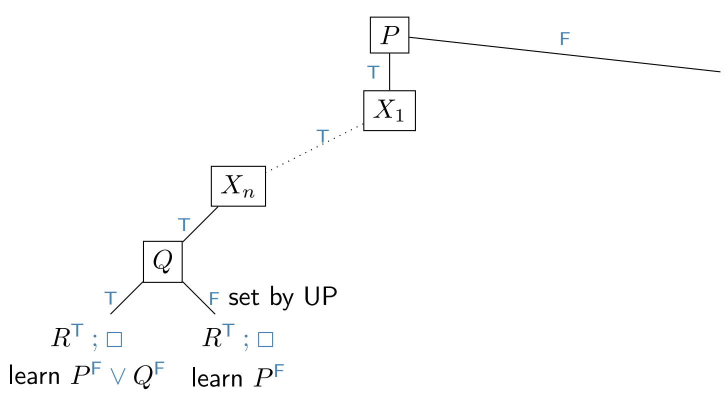
Learned clause: P^F

NOT the same Mistakes over Again: (Redundance1)

▷ **Example B.2.7.** Continuing from ??:

$$\Delta := P^F \lor Q^F \lor R^T ; P^F \lor Q^F \lor R^F ; P^F \lor Q^T \lor R^T ; P^F \lor Q^T \lor R^F$$

DPLL on $\Delta ; \Theta$ with $\Theta := X_1{}^T \lor \ldots \lor X_n{}^T ; X_1{}^F \lor \ldots \lor X_n{}^F$



▷ **Note:** Here, the problem could be avoided by splitting over different variables.

▷ **Problem:** This is not so in general! (see next slide)

Clause Learning vs. Resolution

▷ **Recall:** DPLL $\widehat{=}$ tree resolution (from slide 400)

1. **in particular**: each derived clause C (not in Δ) is derived anew every time it is used.
2. **Problem**: there are Δ whose shortest tree resolution proof is exponentially longer than their shortest (general) resolution proof.

▷ **Good News:** This is no longer the case with clause learning!

1. We add each learned clause C to Δ, can use it as often as we like.
2. Clause learning renders DPLL equivalent to full resolution [BKS04; PD09]. (Inhowfar exactly this is the case was an open question for ca. 10 years, so it's not as easy as I made it look here ...)

▷ **In particular:** Selecting different variables/values to split on can *provably* not bring DPLL up to the power of DPLL+Clause Learning. (cf. slide 713, and previous slide)

"DPLL + Clause Learning"?

▷ **Disclaimer:** We have only seen *how to learn a clause from a conflict*.

▷ We will *not* cover how the overall DPLL algorithm changes, given this learning. Slides 711 – 713 are merely meant to give a *rough intuition* on "backjumping".

▷ **Definition B.2.8 (Just for the record).** (not exam or exercises relevant)

- One *could* run "DPLL + Clause Learning" by always backtracking to the maximal-level choice variable contained in the learned clause.
- The actual algorithm is called Conflict Directed Clause Learning (CDCL), and differs from DPLL more radically:

let $L := 0$; $I := \emptyset$
repeat
  execute UP
  if a conflict was reached **then** /∗ learned clause $C = \overline{l_1} \vee \ldots \vee \overline{l_k} \vee \overline{l'}$ ∗/
    if $L = 0$ **then return** UNSAT
    $L := \max_{i=1}^{k}$ level(l_i); erase I below L
    add C into Δ; add $\overline{l'}$ **to** I at level L
  else
    if I is a total interpretation **then return** I
    choose a new decision literal l; add l **to** I at level L
    $L := L + 1$

Remarks

- **Which clause(s) to learn?:**
  - While we only select choice literals, much more can be done.
  - For any cut through the conflict graph, with Choice literals on the "left hand" side of the cut and the conflict literals on the right-hand side, the literals on the left border of the cut yield a learnable clause.
  - Must take care to *not learn too many clauses* ...
- **Origins of clause learning:**
  - Clause learning originates from "explanation-based (no-good) learning" developed in the CSP community.
  - The distinguishing feature here is that the "no-good" is a clause:
    - The exact same type of constraint as the rest of Δ.

B.3 Phase Transitions: Where the *Really* Hard Problems Are

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25088`.

Where Are the Hard Problems?

- SAT is **NP** hard. Worst case for DPLL is $\mathcal{O}(2^n)$, with n propositions.
- Imagine I gave you as homework to make a formula family $\{\varphi\}$ where DPLL running time necessarily is in the order of $\mathcal{O}(2^n)$.
  - I promise you're not gonna find this easy ... (although it is of course possible: e.g., the "Pigeon Hole Problem").
- People noticed by the early 90s that, in practice, the DPLL worst case does not tend to happen.
- Modern SAT solvers successfully tackle practical instances where $n > 1.000.000$.

Where Are the Hard Problems?

- **So, what's the problem:** Science is about *understanding the world*.
  - Are "hard cases" just pathological outliers?
  - Can we say something about the *typical case*?
- **Difficulty 1:** What is the "typical case" in applications? E.g., what is the "average" hardware verification instance?

- ▷ Consider precisely defined random distributions instead.

▷ **Difficulty 2:** Search trees get very complex, and are difficult to analyze mathematically, even in trivial examples. Never mind examples of practical relevance ...

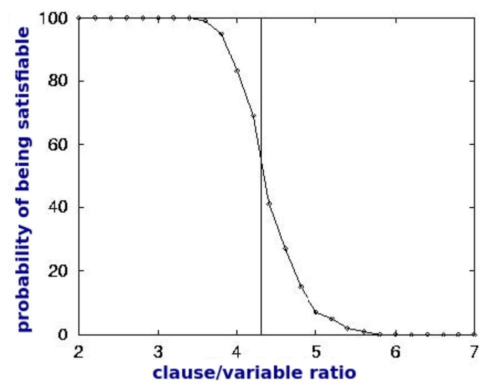
- ▷ The most successful works are empirical. (Interesting theory is mainly concerned with *hand-crafted* formulas, like the Pigeon Hole Problem.)

Phase Transitions in SAT [MSL92]

▷ **Fixed clause length model:** Fix clause length k; n variables.
Generate m clauses, by uniformly choosing k variables P for each clause C, and for each variable P deciding uniformly whether to add P or P^{F} into C.

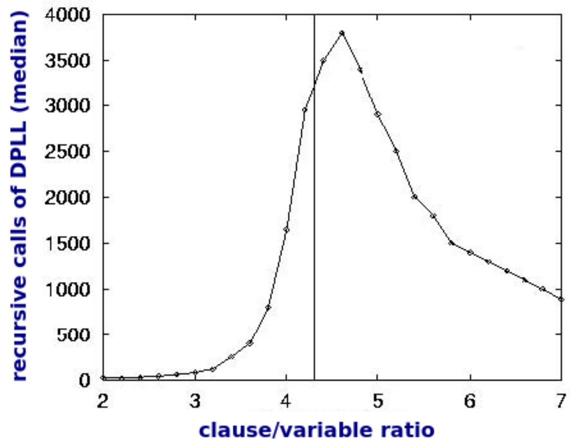
▷ **Order parameter:** Clause/variable ratio $\frac{m}{n}$.

▷ **Phase transition:** (Fixing $k = 3$, $n = 50$)



Does DPLL Care?

▷ **Oh yes, it does:** Extreme running time peak at the phase transition!



Why Does DPLL Care?

▷ **Intuition:**

Under-Constrained: Satisfiability likelihood close to 1. Many solutions, first DPLL search path usually successful. ("Deep but narrow")

Over-Constrained: Satisfiability likelihood close to 0. Most DPLL search paths short, conflict reached after few applications of splitting rule. ("Broad but shallow")

Critically Constrained: At the phase transition, many *almost-successful* DPLL search paths. ("Close, but no cigar")

The Phase Transition Conjecture

▷ **Definition B.3.1.** We say that a class P of problems exhibits a phase transition, if there is an order parameter o, i.e. a structural parameter of P, so that almost all the hard problems of P cluster around a critical value c of o and c separates one region of the problem space from another, e.g. over-constrained and under-constrained regions.

▷ All **NP**-complete problems exhibit at least one phase transition.

▷ [CKT91] confirmed this for Graph Coloring and Hamiltonian Circuits. Later work confirmed it for SAT (see previous slides), and for numerous other **NP**-complete problems.

Why Should *We* Care?

- **Enlightenment:**
  - Phase transitions contribute to the fundamental understanding of the behavior of search, even if it's only in random distributions.
  - There are interesting theoretical connections to phase transition phenomena in physics. (See [GS05] for a short summary.)
- **Ok, but what can we use these results for?:**
  - **Benchmark design**: Choose instances from phase transition region.
    - Commonly used in competitions etc. (In SAT, random phase transition formulas are the most difficult for DPLL style searches.)
  - **Predicting solver performance**: Yes, but very limited because:
- All this works only for the particular considered *distributions of instances*! Not meaningful for any other instances.

Appendix C

Completeness of Calculi for First-Order Logic

We will now analyze the first-order calculi for completeness. Just as in the case of the propositional calculi, we prove a model existence theorem for the first-order model theory and then use that for the completeness proofs[2]. The proof of the first-order model existence theorem is completely analogous to the propositional one; indeed, apart from the model construction itself, it is just an extension by a treatment for the first-order quantifiers.[3] EdN:2 EdN:3

C.1 Abstract Consistency and Model Existence

We will now come to an important tool in the theoretical study of reasoning calculi: the "abstract consistency"/"model existence" method. This method for analyzing calculi was developed by Jaako Hintikka, Raymond Smullyan, and Peter Andrews in 1950-1970 as an encapsulation of similar constructions that were used in completeness arguments in the decades before. The basis for this method is Smullyan's Observation [Smu63] that completeness proofs based on Hintikka sets only certain properties of consistency and that with little effort one can obtain a generalization "Smullyan's Unifying Principle".

The basic intuition for this method is the following: typically, a logical system $\mathcal{L} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ has multiple calculi, human-oriented ones like the natural deduction calculi and machine-oriented ones like the automated theorem proving calculi. All of these need to be analyzed for completeness (as a basic quality assurance measure).

A completeness proof for a calculus $\mathcal{C}$ for $\mathcal{S}$ typically comes in two parts: one analyzes $\mathcal{C}$-consistency (sets that cannot be refuted in $\mathcal{C}$), and the other construct $\mathcal{K}$-models for $\mathcal{C}$-consistent sets.

In this situtation the "abstract consistency"/"model existence" method encapsulates the model construction process into a meta-theorem: the "model existence" theorem. This provides a set of syntactic ("abstract consistency") conditions for calculi that are sufficient to construct models.

With the model existence theorem it suffices to show that $\mathcal{C}$-consistency is an abstract consistency property (a purely syntactic task that can be done by a $\mathcal{C}$-proof transformation argument) to obtain a completeness result for $\mathcal{C}$.

Model Existence (Overview)

▷ **Definition:** Abstract consistency

[2] EdNote: reference the theorems
[3] EdNote: MK: what about equality?

- ▷ **Definition:** Hintikka set (maximally abstract consistent)
- ▷ **Theorem:** Hintikka sets are satisfiable
- ▷ **Theorem:** If Φ is abstract consistent, then Φ can be extended to a Hintikka set.
- ▷ **Corollary:** If Φ is abstract consistent, then Φ is satisfiable.
- ▷ **Application:** Let $\mathcal{C}$ be a calculus, if Φ is $\mathcal{C}$-consistent, then Φ is abstract consistent.
- ▷ **Corollary:** $\mathcal{C}$ is complete.

The proof of the model existence theorem goes via the notion of a Hintikka set, a set of formulae with very strong syntactic closure properties, which allow to read off models. Jaako Hintikka's original idea for completeness proofs was that for every complete calculus $\mathcal{C}$ and every $\mathcal{C}$-consistent set one can induce a Hintikka set, from which a model can be constructed. This can be considered as a first model existence theorem. However, the process of obtaining a Hintikka set for a $\mathcal{C}$-consistent set Φ of sentences usually involves complicated calculus dependent constructions.

In this situation, Raymond Smullyan was able to formulate the sufficient conditions for the existence of Hintikka sets in the form of "abstract consistency properties" by isolating the calculus independent parts of the Hintikka set construction. His technique allows to reformulate Hintikka sets as maximal elements of abstract consistency classes and interpret the Hintikka set construction as a maximizing limit process.

To carry out the "model-existence"/"abstract consistency" method, we will first have to look at the notion of consistency.

Consistency and refutability are very important notions when studying the completeness for calculi; they form syntactic counterparts of satisfiability.

Consistency

- ▷ Let $\mathcal{C}$ be a calculus,. . .
- ▷ **Definition C.1.1.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-refutable, if there is a refutation, i.e. a derivation of a contradiction from Φ. The act of finding a refutation for Φ is called refuting Φ.
- ▷ **Definition C.1.2.** We call a pair of formulae $\mathbf{A}$ and $\neg\mathbf{A}$ a contradiction.
- ▷ So a set Φ is $\mathcal{C}$-refutable, if $\mathcal{C}$ canderive a contradiction from it.
- ▷ **Definition C.1.3.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-consistent, iff there is a formula $\mathbf{B}$, that is not derivable from Φ in $\mathcal{C}$.
- ▷ **Definition C.1.4.** We call a calculus $\mathcal{C}$ reasonable, iff implication elimination and conjunction introduction are admissible in $\mathcal{C}$ and $\mathbf{A}\wedge\neg\mathbf{A}\Rightarrow\mathbf{B}$ is a $\mathcal{C}$-theorem.
- ▷ **Theorem C.1.5.** *$\mathcal{C}$-inconsistency and $\mathcal{C}$-refutability coincide for reasonable calculi.*

It is very important to distinguish the syntactic $\mathcal{C}$-refutability and $\mathcal{C}$-consistency from satisfiability, which is a property of formulae that is at the heart of semantics. Note that the former have the calculus (a syntactic device) as a parameter, while the latter does not. In fact we should actually say $\mathcal{S}$-satisfiability, where $\langle\mathcal{L},\mathcal{K},\models\rangle$ is the current logical system.

Even the word "contradiction" has a syntactical flavor to it, it translates to "saying against each other" from its Latin root.
The notion of an "abstract consistency class" provides the a calculus-independent notion of consistency: A set Φ of sentences is considered "consistent in an abstract sense", iff it is a member of an abstract consistency class ∇.

Abstract Consistency

▷ **Definition C.1.6.** Let ∇ be a collection of sets. We call ∇ closed under subsets, iff for each $\Phi \in \nabla$, all subsets $\Psi \subseteq \Phi$ are elements of ∇.

▷ **Notation:** We will use $\Phi * \mathbf{A}$ for $\Phi \cup \{\mathbf{A}\}$.

▷ **Definition C.1.7.** A family $\nabla \subseteq wff_o(\Sigma_\iota, \mathcal{V}_\iota)$ of sets of formulae is called a (first-order) abstract consistency class, iff it is closed under subsets, and for each $\Phi \in \nabla$

- ∇_c) $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$ for atomic $\mathbf{A} \in wff_o(\Sigma_\iota, \mathcal{V}_\iota)$.
- $\nabla_\neg$) $\neg\neg\mathbf{A} \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$
- $\nabla_\wedge$) $\mathbf{A} \wedge \mathbf{B} \in \Phi$ implies $\Phi \cup \{\mathbf{A}, \mathbf{B}\} \in \nabla$
- $\nabla_\vee$) $\neg(\mathbf{A} \wedge \mathbf{B}) \in \Phi$ implies $\Phi * \neg\mathbf{A} \in \nabla$ or $\Phi * \neg\mathbf{B} \in \nabla$
- $\nabla_\forall$) If $\forall X.\mathbf{A} \in \Phi$, then $\Phi * ([\mathbf{B}/X](\mathbf{A})) \in \nabla$ for each closed term $\mathbf{B}$.
- $\nabla_\exists$) If $\neg(\forall X.\mathbf{A}) \in \Phi$ and c is an individual constant that does not occur in Φ, then $\Phi * \neg([c/X](\mathbf{A})) \in \nabla$

FAU Michael Kohlhase: Artificial Intelligence 1 726 2025-02-06

The conditions are very natural: Take for instance ∇_c, it would be foolish to call a set Φ of sentences "consistent under a complete calculus", if it contains an elementary contradiction. The next condition $\nabla_\neg$ says that if a set Φ that contains a sentence $\neg\neg\mathbf{A}$ is "consistent", then we should be able to extend it by $\mathbf{A}$ without losing this property; in other words, a complete calculus should be able to recognize $\mathbf{A}$ and $\neg\neg\mathbf{A}$ to be equivalent. We will carry out the proof here, since it gives us practice in dealing with the abstract consistency properties.
The main result here is that abstract consistency classes can be extended to compact ones. The proof is quite tedious, but relatively straightforward. It allows us to assume that all abstract consistency classes are compact in the first place (otherwise we pass to the compact extension).
Actually we are after abstract consistency classes that have an even stronger property than just being closed under subsets. This will allow us to carry out a limit construction in the Hintikka set extension argument later.

Compact Collections

▷ **Definition C.1.8.** We call a collection ∇ of sets compact, iff for any set Φ we have $\Phi \in \nabla$, iff $\Psi \in \nabla$ for every finite subset Ψ of Φ.

▷ **Lemma C.1.9.** *If ∇ is compact, then ∇ is closed under subsets.*

▷ *Proof:*

1. Suppose $S \subseteq T$ and $T \in \nabla$.
2. Every finite subset A of S is a finite subset of T.
3. As ∇ is compact, we know that $A \in \nabla$.
4. Thus $S \in \nabla$.

The property of being closed under subsets is a "downwards-oriented" property: We go from large sets to small sets, compactness (the interesting direction anyways) is also an "upwards-oriented" property. We can go from small (finite) sets to large (infinite) sets. The main application for the compactness condition will be to show that infinite sets of formulae are in a collection ∇ by testing all their finite subsets (which is much simpler).

Compact Abstract Consistency Classes

▷ **Lemma C.1.10.** *Any first-order abstract consistency class can be extended to a compact one.*

▷ *Proof:*

1. We choose $\nabla' := \{\Phi \subseteq cwff_o(\Sigma_\iota) \,|\, \text{every finite subset of } \Phi \text{is in } \nabla\}$.
2. Now suppose that $\Phi \in \nabla$. ∇ is closed under subsets, so every finite subset of Φ is in ∇ and thus $\Phi \in \nabla'$. Hence $\nabla \subseteq \nabla'$.
3. Let us now show that each ∇ is compact.'
   3.1. Suppose $\Phi \in \nabla'$ and Ψ is an arbitrary finite subset of Φ.
   3.2. By definition of ∇' all finite subsets of Φ are in ∇ and therefore $\Psi \in \nabla'$.
   3.3. Thus all finite subsets of Φ are in ∇' whenever Φ is in ∇'.
   3.4. On the other hand, suppose all finite subsets of Φ are in ∇'.
   3.5. Then by the definition of ∇' the finite subsets of Φ are also in ∇, so $\Phi \in \nabla'$. Thus ∇' is compact.
4. Note that ∇' is closed under subsets by the Lemma above.
5. Next we show that if ∇ satisfies ∇_*, then ∇ satisfies ∇_*.'
   5.1. To show ∇_c, let $\Phi \in \nabla'$ and suppose there is an atom $\mathbf{A}$, such that $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$. Then $\{\mathbf{A}, \neg\mathbf{A}\} \in \nabla$ contradicting ∇_c.
   5.2. To show $\nabla_\neg$, let $\Phi \in \nabla'$ and $\neg\neg\mathbf{A} \in \Phi$, then $\Phi{*}\mathbf{A} \in \nabla'$.
      5.2.1. Let Ψ be any finite subset of $\Phi{*}\mathbf{A}$, and $\Theta := (\Psi \backslash \{\mathbf{A}\}){*}\neg\neg\mathbf{A}$.
      5.2.2. Θ is a finite subset of Φ, so $\Theta \in \nabla$.
      5.2.3. Since ∇ is an abstract consistency class and $\neg\neg\mathbf{A} \in \Theta$, we get $\Theta{*}\mathbf{A} \in \nabla$ by $\nabla_\neg$.
      5.2.4. We know that $\Psi \subseteq \Theta{*}\mathbf{A}$ and ∇ is closed under subsets, so $\Psi \in \nabla$.
      5.2.5. Thus every finite subset Ψ of $\Phi{*}\mathbf{A}$ is in ∇ and therefore by definition $\Phi{*}\mathbf{A} \in \nabla'$.
   5.3. the other cases are analogous to $\nabla_\neg$.

Hintikka sets are sets of sentences with very strong analytic closure conditions. These are motivated as maximally consistent sets i.e. sets that already contain everything that can be consistently added to them.

∇-Hintikka Set

▷ **Definition C.1.11.** Let ∇ be an abstract consistency class, then we call a set $\mathcal{H} \in \nabla$ a ∇ Hintikka Set, iff $\mathcal{H}$ is maximal in ∇, i.e. for all $\mathbf{A}$ with $\mathcal{H}{*}\mathbf{A} \in \nabla$ we already have $\mathbf{A} \in \mathcal{H}$.

▷ **Theorem C.1.12 (Hintikka Properties).** *Let ∇ be an abstract consistency class and $\mathcal{H}$ be a ∇-Hintikka set, then*

$\mathcal{H}_c$) *For all* $\mathbf{A} \in wff_o(\Sigma_\iota, \mathcal{V}_\iota)$ *we have* $\mathbf{A} \notin \mathcal{H}$ *or* $\neg\mathbf{A} \notin \mathcal{H}$.

$\mathcal{H}_\neg$) *If* $\neg\neg\mathbf{A} \in \mathcal{H}$ *then* $\mathbf{A} \in \mathcal{H}$.

$\mathcal{H}_\wedge$) *If* $\mathbf{A} \wedge \mathbf{B} \in \mathcal{H}$ *then* $\mathbf{A}, \mathbf{B} \in \mathcal{H}$.

$\mathcal{H}_\vee$) *If* $\neg(\mathbf{A} \wedge \mathbf{B}) \in \mathcal{H}$ *then* $\neg\mathbf{A} \in \mathcal{H}$ *or* $\neg\mathbf{B} \in \mathcal{H}$.

$\mathcal{H}_\forall$) *If* $\forall X.\mathbf{A} \in \mathcal{H}$, *then* $[\mathbf{B}/X](\mathbf{A}) \in \mathcal{H}$ *for each closed term* $\mathbf{B}$.

$\mathcal{H}_\exists$) *If* $\neg(\forall X.\mathbf{A}) \in \mathcal{H}$ *then* $\neg([\mathbf{B}/X](\mathbf{A})) \in \mathcal{H}$ *for some term closed term* $\mathbf{B}$.

▷ *Proof:*

We prove the properties in turn $\mathcal{H}_c$ *goes by induction on the structure of* $\mathbf{A}$

1. $\mathbf{A}$ atomic
   1.1. Then $\mathbf{A} \notin \mathcal{H}$ or $\neg\mathbf{A} \notin \mathcal{H}$ by ∇_c.
2. $\mathbf{A} = \neg\mathbf{B}$
   2.1. Let us assume that $\neg\mathbf{B} \in \mathcal{H}$ and $\neg\neg\mathbf{B} \in \mathcal{H}$,
   2.2. then $\mathcal{H}*\mathbf{B} \in \nabla$ by $\nabla_\neg$, and therefore $\mathbf{B} \in \mathcal{H}$ by maximality.
   2.3. So $\{\mathbf{B}, \neg\mathbf{B}\} \subseteq \mathcal{H}$, which contradicts the induction hypothesis.
3. $\mathbf{A} = \mathbf{B} \vee \mathbf{C}$ *similar to the previous case*
4. We prove $\mathcal{H}_\neg$ by maximality of $\mathcal{H}$ in ∇.
   4.1. If $\neg\neg\mathbf{A} \in \mathcal{H}$, then $\mathcal{H}*\mathbf{A} \in \nabla$ by $\nabla_\neg$.
   4.2. The maximality of $\mathcal{H}$ now gives us that $\mathbf{A} \in \mathcal{H}$.
5. The other $\mathcal{H}_*$ are similar

The following theorem is one of the main results in the "abstract consistency"/"model existence" method. For any abstract consistent set Φ it allows us to construct a Hintikka set $\mathcal{H}$ with $\Phi \in \mathcal{H}$.

Extension Theorem

▷ **Theorem C.1.13.** *If* ∇ *is an abstract consistency class and* $\Phi \in \nabla$ *finite, then there is a* ∇*-Hintikka set* $\mathcal{H}$ *with* $\Phi \subseteq \mathcal{H}$.

▷ *Proof:*

1. Wlog. assume that ∇ compact (else use compact extension)
2. Choose an enumeration $\mathbf{A}_1, \ldots$ of $cwff_o(\Sigma_\iota)$ and $c_1, \ldots$ of Σ_0^{sk}.
3. and construct a sequence of sets $\mathbf{H}_i$ with $\mathbf{H}_0 := \Phi$ and

$$\mathbf{H}_{n+1} := \begin{cases} \mathbf{H}_n & \text{if } \mathbf{H}_n*\mathbf{A}_n \notin \nabla \\ \mathbf{H}_n \cup \{\mathbf{A}_n, \neg([c_n/X](\mathbf{B}))\} & \text{if } \mathbf{H}_n*\mathbf{A}_n \in \nabla \text{ and } \mathbf{A}_n = \neg(\forall X.\mathbf{B}) \\ \mathbf{H}_n*\mathbf{A}_n & \text{else} \end{cases}$$

4. Note that all $\mathbf{H}_i \in \nabla$, choose $\mathcal{H} := \bigcup_{i\in\mathbb{N}}\mathbf{H}_i$
5. $\Psi \subseteq \mathcal{H}$ finite implies there is a $j \in \mathbb{N}$ such that $\Psi \subseteq \mathbf{H}_j$,
6. so $\Psi \in \nabla$ as ∇ closed under subsets and $\mathcal{H} \in \nabla$ as ∇ is compact.
7. Let $\mathcal{H}*\mathbf{B} \in \nabla$, then there is a $j \in \mathbb{N}$ with $\mathbf{B} = \mathbf{A}_j$, so that $\mathbf{B} \in \mathbf{H}_{j+1}$ and $\mathbf{H}_{j+1} \subseteq \mathcal{H}$
8. Thus $\mathcal{H}$ is ∇-maximal

Note that the construction in the proof above is non-trivial in two respects. First, the limit construction for $\mathcal{H}$ is not executed in our original abstract consistency class ∇, but in a suitably extended one to make it compact — the original would not have contained $\mathcal{H}$ in general. Second, the set $\mathcal{H}$ is not unique for Φ, but depends on the choice of the enumeration of $cwff_o(\Sigma_\iota)$. If we pick a different enumeration, we will end up with a different $\mathcal{H}$. Say if $\mathbf{A}$ and $\neg\mathbf{A}$ are both ∇-consistent[4] with Φ, then depending on which one is first in the enumeration $\mathcal{H}$, will contain that one; with all the consequences for subsequent choices in the construction process.

Valuations

- **Definition C.1.14.** A function $\nu\colon cwff_o(\Sigma_\iota)\to\mathcal{D}_0$ is called a (first-order) valuation, iff ν is a propositional valuation and
  - $\nu(\forall X.\mathbf{A}) = \mathsf{T}$, iff $\nu([\mathbf{B}/X](\mathbf{A})) = \mathsf{T}$ for all closed terms $\mathbf{B}$.
- **Lemma C.1.15.** *If $\varphi\colon \mathcal{V}_\iota \to U$ is a variable assignment, then $\mathcal{I}_\varphi\colon cwff_o(\Sigma_\iota) \to \mathcal{D}_0$ is a valuation.*
- *Proof sketch:* Immediate from the definitions

Michael Kohlhase: Artificial Intelligence 1 733 2025-02-06

Thus a valuation is a weaker notion of evaluation in first-order logic; the other direction is also true, even though the proof of this result is much more involved: The existence of a first-order valuation that makes a set of sentences true entails the existence of a model that satisfies it.[5]

Valuation and Satisfiability

- **Lemma C.1.16.** *If $\nu\colon cwff_o(\Sigma_\iota) \to \mathcal{D}_0$ is a valuation and $\Phi \subseteq cwff_o(\Sigma_\iota)$ with $\nu(\Phi) = \{\mathsf{T}\}$, then Φ is satisfiable.*
- *Proof:* We construct a model for Φ.
  1. Let $\mathcal{D}_\iota := cwff_\iota(\Sigma_\iota)$, and
     - $\mathcal{I}(f)\,:\,{\mathcal{D}_\iota}^k \to \mathcal{D}_\iota\,;\langle \mathbf{A}_1,\ldots,\mathbf{A}_k\rangle \mapsto f(\mathbf{A}_1,\ldots,\mathbf{A}_k)$ for $f \in \Sigma^f$
     - $\mathcal{I}(p)\,:\,{\mathcal{D}_\iota}^k \to \mathcal{D}_0\,;\langle \mathbf{A}_1,\ldots,\mathbf{A}_k\rangle \mapsto \nu(p(\mathbf{A}_1,\ldots,\mathbf{A}_k))$ for $p \in \Sigma^p$.
  2. Then variable assignments into $\mathcal{D}_\iota$ are ground substitutions.
  3. We show $\mathcal{I}_\varphi(\mathbf{A}) = \varphi(\mathbf{A})$ for $\mathbf{A} \in wff_\iota(\Sigma_\iota,\mathcal{V}_\iota)$ by induction on $\mathbf{A}$:

     3.1. $\mathbf{A} = X$

     3.1.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \varphi(X)$ by definition.

     3.2. $\mathbf{A} = f(\mathbf{A}_1,\ldots,\mathbf{A}_k)$

     3.2.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1),\ldots,\mathcal{I}_\varphi(\mathbf{A}_n)) = \mathcal{I}(f)(\varphi(\mathbf{A}_1),\ldots,\varphi(\mathbf{A}_n)) = f(\varphi(\mathbf{A}_1),\ldots,\varphi(\mathbf{A}_n)) = \varphi(f(\mathbf{A}_1,\ldots,\mathbf{A}_k)) = \varphi(\mathbf{A})$

     We show $\mathcal{I}_\varphi(\mathbf{A}) = \nu(\varphi(\mathbf{A}))$ for $\mathbf{A} \in wff_o(\Sigma_\iota,\mathcal{V}_\iota)$ by induction on $\mathbf{A}$.

     3.3. $\mathbf{A} = p(\mathbf{A}_1,\ldots,\mathbf{A}_k)$

     3.3.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}(p)(\mathcal{I}_\varphi(\mathbf{A}_1),\ldots,\mathcal{I}_\varphi(\mathbf{A}_n)) = \mathcal{I}(p)(\varphi(\mathbf{A}_1),\ldots,\varphi(\mathbf{A}_n)) = \nu(p(\varphi(\mathbf{A}_1),\ldots,\varphi(\mathbf{A}_n))) = \nu(\varphi(p(\mathbf{A}_1,\ldots,\mathbf{A}_k))) = \nu(\varphi(\mathbf{A}))$

     3.4. $\mathbf{A} = \neg\mathbf{B}$

     3.4.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathbf{B}) = \nu(\varphi(\mathbf{B})) = \mathsf{F}$, iff $\nu(\varphi(\mathbf{A})) = \mathsf{T}$.

     3.5. $\mathbf{A} = \mathbf{B} \wedge \mathbf{C}$

     3.5.1. similar

     3.6. $\mathbf{A} = \forall X.\mathbf{B}$

[4] EdNote: introduce this above
[5] EdNote: I think that we only get a semivaluation, look it up in Andrews.

3.6.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\psi(\mathbf{B}) = \nu(\psi(\mathbf{B})) = \mathsf{T}$, for all $\mathbf{C} \in \mathcal{D}_\iota$, where $\psi = \varphi,[\mathbf{C}/X]$. This is the case, iff $\nu(\varphi(\mathbf{A})) = \mathsf{T}$.
4. Thus $\mathcal{I}_\varphi(\mathbf{A})\nu(\varphi(\mathbf{A})) = \nu(\mathbf{A}) = \mathsf{T}$ for all $\mathbf{A} \in \Phi$.
5. Hence $\mathcal{M} \models \mathbf{A}$ for $\mathcal{M} := \langle \mathcal{D}_\iota, \mathcal{I} \rangle$.

Now, we only have to put the pieces together to obtain the model existence theorem we are after.

Model Existence

▷ **Theorem C.1.17 (Hintikka-Lemma).** *If ∇ is an abstract consistency class and $\mathcal{H}$ a ∇-Hintikka set, then $\mathcal{H}$ is satisfiable.*

▷ *Proof:*

1. we define $\nu(\mathbf{A}) := \mathsf{T}$, iff $\mathbf{A} \in \mathcal{H}$,
2. then ν is a valuation by the Hintikka set properties.
3. We have $\nu(\mathcal{H}) = \{\mathsf{T}\}$, so $\mathcal{H}$ is satisfiable.

▷ **Theorem C.1.18 (Model Existence).** *If ∇ is an abstract consistency class and $\Phi \in \nabla$, then Φ is satisfiable.*

Proof:

▷ 1. There is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$ (Extension Theorem)
2. We know that $\mathcal{H}$ is satisfiable. (Hintikka-Lemma)
3. In particular, $\Phi \subseteq \mathcal{H}$ is satisfiable.

C.2 A Completeness Proof for First-Order ND

With the model existence proof we have introduced in the last section, the completeness proof for first-order natural deduction is rather simple, we only have to check that ND-consistency is an abstract consistency property.

Consistency, Refutability and Abstract Consistency

▷ **Theorem C.2.1 (Non-Refutability is an Abstract Consistency Property).** $\Gamma := \{\Phi \subseteq \mathit{cwff}_o(\Sigma_\iota) \,|\, \Phi \text{ not } \mathcal{ND}^1\text{−refutable}\}$ *is an abstract consistency class.*

▷ *Proof:* We check the properties of an ACC

1. If Φ is non-refutable, then any subset is as well, so Γ is closed under subsets.
We show the abstract consistency conditions ∇_* for $\Phi \in \Gamma$.
2. ∇_c
2.1. We have to show that $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$ for atomic $\mathbf{A} \in \mathit{wff}_o(\Sigma_\iota, \mathcal{V}_\iota)$.
2.2. Equivalently, we show the contrapositive: If $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$, then $\Phi \notin \Gamma$.
2.3. So let $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$, then Φ is $\mathcal{ND}^1$-refutable by construction.
2.4. So $\Phi \notin \Gamma$.
3. $\nabla_\neg$ *We show the contrapositive again*
3.1. Let $\neg\neg\mathbf{A} \in \Phi$ and $\Phi * \mathbf{A} \notin \Gamma$
3.2. Then we have a refutation $\mathcal{D} \colon \Phi * \mathbf{A} \vdash_{\mathcal{ND}^1} F$

3.3. By prepending an application of $\mathcal{ND}_0\neg E$ for $\neg\neg\mathbf{A}$ to $\mathcal{D}$, we obtain a refutation $\mathcal{D}$: $\Phi\vdash_{\mathcal{ND}^1}F'$.
3.4. Thus $\Phi \notin \Gamma$.

Proof sketch: *other* ∇_* *similar*

This directly yields two important results that we will use for the completeness analysis.

Henkin's Theorem

▷ **Corollary C.2.2 (Henkin's Theorem).** *Every $\mathcal{ND}^1$-consistent set of sentences has a model.*

▷ *Proof:*

1. Let Φ be a $\mathcal{ND}^1$-consistent set of sentences.
2. The class of sets of $\mathcal{ND}^1$-consistent propositions constitute an abstract consistency class.
3. Thus the model existence theorem guarantees a model for Φ.

▷ **Corollary C.2.3 (Löwenheim&Skolem Theorem).** *Satisfiable set Φ of first-order sentences has a countable model.*

Proof sketch: The model we constructed is countable, since the set of ground terms is.

Now, the completeness result for first-order natural deduction is just a simple argument away. We also get a compactness theorem (almost) for free: logical systems with a complete calculus are always compact.

▷ Completeness and Compactness

▷ **Theorem C.2.4 (Completeness Theorem for $\mathcal{ND}^1$).** *If $\Phi \models \mathbf{A}$, then $\Phi\vdash_{\mathcal{ND}^1}\mathbf{A}$.*

▷ *Proof:* We prove the result by playing with negations.

1. If $\mathbf{A}$ is valid in all models of Φ, then $\Phi * \neg\mathbf{A}$ has no model
2. Thus $\Phi * \neg\mathbf{A}$ is inconsistent by (the contrapositive of) Henkins Theorem.
3. So $\Phi\vdash_{\mathcal{ND}^1}\neg\neg\mathbf{A}$ by $\mathcal{ND}_0\neg I$ and thus $\Phi\vdash_{\mathcal{ND}^1}\mathbf{A}$ by $\mathcal{ND}_0\neg E$.

▷ **Theorem C.2.5 (Compactness Theorem for first-order logic).** *If $\Phi \models \mathbf{A}$, then there is already a finite set $\Psi \subseteq \Phi$ with $\Psi \models \mathbf{A}$.*

Proof: This is a direct consequence of the completeness theorem

▷ 1. We have $\Phi \models \mathbf{A}$, iff $\Phi\vdash_{\mathcal{ND}^1}\mathbf{A}$.
2. As a proof is a finite object, only a finite subset $\Psi \subseteq \Phi$ can appear as leaves in the proof.

C.3 Soundness and Completeness of First-Order Tableaux

The soundness of the first-order free-variable tableaux calculus can be established a simple induction over the size of the tableau.

Soundness of $\mathcal{T}_1^f$

▷ **Lemma C.3.1.** *Tableau rules transform satisfiable tableaux into satisfiable ones.*

▷ *Proof:*

we examine the tableau rules in turn

1. propositional rules *as in propositional tableaux*
2. $\mathcal{T}_1^f\exists$ *by* **??**
3. $\mathcal{T}_1^f\bot$ *by* **??** *(substitution value lemma)*
4. $\mathcal{T}_1^f\forall$
   4.1. $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\psi(\mathbf{A}) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$
   4.2. so in particular for some $a \in \mathcal{D}_\iota \neq \emptyset$.

▷ **Corollary C.3.2.** $\mathcal{T}_1^f$ *is correct.*

The only interesting steps are the cut rule, which can be directly handled by the substitution value lemma, and the rule for the existential quantifier, which we do in a separate lemma.

Soundness of $\mathcal{T}_1^f\exists$

▷ **Lemma C.3.3.** $\mathcal{T}_1^f\exists$ *transforms satisfiable tableaux into satisfiable ones.*

▷ *Proof:* Let $\mathcal{T}'$ be obtained by applying $\mathcal{T}_1^f\exists$ to $(\forall X.\mathbf{A})^\mathsf{F}$ in $\mathcal{T}$, extending it with $([f(X^1,\ldots,X^k)/X](\mathbf{A}))^\mathsf{F}$, where $W := \mathrm{free}(\forall X.\mathbf{A}) = \{X^1,\ldots,X^k\}$

1. Let $\mathcal{T}$ be satisfiable in $\mathcal{M} := \langle\mathcal{D},\mathcal{I}\rangle$, then $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{F}$.
   We need to find a model $\mathcal{M}'$ that satisfies $\mathcal{T}'$ *(find interpretation for f)*
2. By definition $\mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{F}$ for some $a \in \mathcal{D}$ (depends on $\varphi|_W$)
3. Let $g\colon \mathcal{D}^k \to \mathcal{D}$ be defined by $g(a_1,\ldots,a_k) := a$, if $\varphi(X^i) = a_i$
4. choose $\mathcal{M} = \langle\mathcal{D},\mathcal{I}'\rangle'$ with $\mathcal{I}' := \mathcal{I},[g/f]$, then by subst. value lemma

$$\begin{aligned}\mathcal{I}'_\varphi([f(X^1,\ldots,X^k)/X](\mathbf{A})) &= \mathcal{I}'_{\varphi,[\mathcal{I}'_\varphi(f(X^1,\ldots,X^k))/X]}(\mathbf{A})\\ &= \mathcal{I}'_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{F}\end{aligned}$$

5. So $([f(X^1,\ldots,X^k)/X](\mathbf{A}))^\mathsf{F}$ satisfiable in $\mathcal{M}'$

This proof is paradigmatic for soundness proofs for calculi with Skolemization. We use the axiom of choice at the meta-level to choose a meaning for the Skolem constant. Armed with the Model Existence Theorem for first-order logic (**??**), the completeness of first-order tableaux is similarly straightforward. We just have to show that the collection of tableau-irrefutable sentences is an abstract consistency class, which is a simple proof-transformation exercise in all but the universal quantifier case, which we postpone to its own Lemma (**??**).

Completeness of ($\mathcal{T}_1^f$)

▷ **Theorem C.3.4.** *$\mathcal{T}_1^f$ is refutation complete.*

▷ *Proof:* We show that $\nabla := \{\Phi \mid \Phi^{\mathsf{T}}$ has no closed Tableau$\}$ is an abstract consistency class

1. as for propositional case.
2. by the lifting lemma below
3. Let $\mathcal{T}$ be a closed tableau for $\neg(\forall X.\mathbf{A}) \in \Phi$ and $\Phi^{\mathsf{T}}*([c/X](\mathbf{A}))^{\mathsf{F}} \in \nabla$.

$$\begin{array}{c} \Psi^{\mathsf{T}} \\ (\forall X.\mathbf{A})^{\mathsf{F}} \\ ([c/X](\mathbf{A}))^{\mathsf{F}} \\ Rest \end{array} \qquad \begin{array}{c} \Psi^{\mathsf{T}} \\ (\forall X.\mathbf{A})^{\mathsf{F}} \\ ([f(X_1,\ldots,X_k)/X](\mathbf{A}))^{\mathsf{F}} \\ [f(X_1,\ldots,X_k)/c](Rest) \end{array}$$

So we only have to treat the case for the universal quantifier. This is what we usually call a "lifting argument", since we have to transform ("lift") a proof for a formula $\theta(\mathbf{A})$ to one for $\mathbf{A}$. In the case of tableaux we do that by an induction on the tableau refutation for $\theta(\mathbf{A})$ which creates a tableau-isomorphism to a tableau refutation for $\mathbf{A}$.

Tableau-Lifting

▷ **Theorem C.3.5.** *If $\mathcal{T}_\theta$ is a closed tableau for a set $\theta(\Phi)$ of formulae, then there is a closed tableau $\mathcal{T}$ for Φ.*

▷ *Proof:* by induction over the structure of $\mathcal{T}_\theta$ we build an isomorphic tableau $\mathcal{T}$, and a tableau-isomorphism $\omega\colon \mathcal{T} \to \mathcal{T}_\theta$, such that $\omega(\mathbf{A}) = \theta(\mathbf{A})$.

only the tableau-substitution rule is interesting.

1. Let $(\theta(\mathbf{A}_i))^{\mathsf{T}}$ and $(\theta(\mathbf{B}_i))^{\mathsf{F}}$ cut formulae in the branch Θ_θ^i of $\mathcal{T}_\theta$
2. there is a joint unifier σ of $(\theta(\mathbf{A}_1)) =^? (\theta(\mathbf{B}_1)) \wedge \ldots \wedge (\theta(\mathbf{A}_n)) =^? (\theta(\mathbf{B}_n))$
3. thus $\sigma \circ \theta$ is a unifier of $\mathbf{A}$ and $\mathbf{B}$
4. hence there is a most general unifier ρ of $\mathbf{A}_1 =^? \mathbf{B}_1 \wedge \ldots \wedge \mathbf{A}_n =^? \mathbf{B}_n$
5. so Θ is closed.

Again, the "lifting lemma for tableaux" is paradigmatic for lifting lemmata for other refutation calculi.

C.4 Soundness and Completeness of First-Order Resolution

Correctness (CNF)

▷ **Lemma C.4.1.** *A set Φ of sentences is satisfiable, iff $CNF_1(\Phi)$ is.*

▷ *Proof:* propositional rules and $\forall$-rule are trivial; do the $\exists$-rule

1. Let $(\forall X.\mathbf{A})^{\mathsf{F}}$ satisfiable in $\mathcal{M} := \langle \mathcal{D}, \mathcal{I} \rangle$ and $\mathrm{free}(\mathbf{A}) = \{X^1, \ldots, X^n\}$
2. $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{F}$, so there is an $a \in \mathcal{D}$ with $\mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{F}$ (only depends on $\varphi|_{\mathrm{free}(\mathbf{A})}$)
3. let $g\colon \mathcal{D}^n \to \mathcal{D}$ be defined by $g(a_1, \ldots, a_n) := a$, iff $\varphi(X^i) = a_i$.
4. choose $\mathcal{M}' := \langle \mathcal{D}, \mathcal{I}' \rangle$ with $\mathcal{I}(f)' := g$, then $\mathcal{I}'_\varphi([f(X^1, \ldots, X^k)/X](\mathbf{A})) = \mathsf{F}$
5. Thus $([f(X^1, \ldots, X^k)/X](\mathbf{A}))^{\mathsf{F}}$ is satisfiable in $\mathcal{M}'$

Resolution (Correctness)

▷ **Definition C.4.2.** A clause is called satisfiable, iff $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$ for one of its literals $\mathbf{A}^\alpha$.

▷ **Lemma C.4.3.** *$\Box$ is unsatisfiable*

▷ **Lemma C.4.4.** *CNF transformations preserve satisfiability* (see above)

▷ **Lemma C.4.5.** *Resolution and factorization too!*

Completeness ($\mathcal{R}_1$)

▷ **Theorem C.4.6.** *$\mathcal{R}_1$ is refutation complete.*

▷ *Proof:* $\nabla := \{\Phi \mid \Phi^{\mathsf{T}}$ has no closed tableau$\}$ is an abstract consistency class

1. as for propositional case.
2. by the lifting lemma below
3. Let $\mathcal{T}$ be a closed tableau for $\neg(\forall X.\mathbf{A}) \in \Phi$ and $\Phi^{\mathsf{T}} * ([c/X](\mathbf{A}))^{\mathsf{F}} \in \nabla$.
4. $CNF_1(\Phi^{\mathsf{T}}) = CNF_1(\Psi^{\mathsf{T}}) \cup CNF_1(([f(X_1, \ldots, X_k)/X](\mathbf{A}))^{\mathsf{F}})$
5. $([f(X_1, \ldots, X_k)/c](CNF_1(\Phi^{\mathsf{T}}))) * ([c/X](\mathbf{A}))^{\mathsf{F}} = CNF_1(\Phi^{\mathsf{T}})$
6. so $\mathcal{R}_1 : CNF_1(\Phi^{\mathsf{T}}) \vdash_{\mathcal{D}'} \Box$, where $\mathcal{D} = [f(X_1', \ldots, X_k')/c](\mathcal{D})$.

Clause Set Isomorphism

▷ **Definition C.4.7.** Let $\mathbf{B}$ and $\mathbf{C}$ be clauses, then a clause isomorphism $\omega\colon \mathbf{C} \to \mathbf{D}$ is a bijection of the literals of $\mathbf{C}$ and $\mathbf{D}$, such that $\omega(\mathbf{L})^\alpha = \mathbf{M}^\alpha$ (conserves labels) We call ω θ compatible, iff $\omega(\mathbf{L}^\alpha) = (\theta(\mathbf{L}))^\alpha$

▷ **Definition C.4.8.** Let Φ and Ψ be clause sets, then we call a bijection $\Omega\colon \Phi \to \Psi$ a clause set isomorphism, iff there is a clause isomorphism $\omega\colon \mathbf{C} \to \Omega(\mathbf{C})$ for each $\mathbf{C} \in \Phi$.

▷ **Lemma C.4.9.** *If $\theta(\Phi)$ is set of formulae, then there is a θ-compatible clause set isomorphism $\Omega\colon CNF_1(\Phi) \to CNF_1(\theta(\Phi))$.*

▷ *Proof sketch:* by induction on the CNF derivation of $CNF_1(\Phi)$.

Lifting for $\mathcal{R}_1$

▷ **Theorem C.4.10.** *If $\mathcal{R}_1\colon(\theta(\Phi))\vdash_{\mathcal{D}_\theta}\Box$ for a set $\theta(\Phi)$ of formulae, then there is a $\mathcal{R}_1$-refutation for Φ.*

▷ *Proof:* by induction over $\mathcal{D}_\theta$ we construct a $\mathcal{R}_1$-derivation $\mathcal{R}_1\colon\Phi\vdash_{\mathcal{D}}\mathbf{C}$ and a θ-compatible clause set isomorphism $\Omega\colon\mathcal{D}\to\mathcal{D}_\theta$

1. If $\mathcal{D}_\theta$ ends in $$\frac{\dfrac{\mathcal{D}'_\theta}{((\theta(\mathbf{A}))\vee(\theta(\mathbf{C})))^{\mathsf{T}}}\quad\dfrac{\mathcal{D}''_\theta}{(\theta(\mathbf{B}))^{\mathsf{F}}\vee(\theta(\mathbf{D}))}}{(\sigma(\theta(\mathbf{C})))\vee(\sigma(\theta(\mathbf{B})))}\,res$$

   then we have (IH) clause isormorphisms $\omega'\colon\mathbf{A}^{\mathsf{T}}\vee\mathbf{C}\to(\theta(\mathbf{A}))^{\mathsf{T}}\vee(\theta(\mathbf{C}))$ and $\omega'\colon\mathbf{B}^{\mathsf{T}}\vee\mathbf{D}\to(\theta(\mathbf{B}))^{\mathsf{T}},\theta(\mathbf{D})$
2. thus $\dfrac{\mathbf{A}^{\mathsf{T}}\vee\mathbf{C}\quad\mathbf{B}^{\mathsf{F}}\vee\mathbf{D}}{(\rho(\mathbf{C}))\vee(\rho(\mathbf{B}))}\,Res$ where $\rho=\mathbf{mgu}(\mathbf{A},\mathbf{B})$(exists, as $\sigma\circ\theta$ unifier)