

Artificial Intelligence 1

Prof. Dr. Michael Kohlhase

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

2022-05-04

Chapter 1 Administrativa

- ▶ **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)

- ▶ **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)
- ▶ **Intuition:** (take them with a kilo of salt)
 - ▶ This is what I assume you know! (I have to assume something)
 - ▶ In many cases, the dependency of AI-1 on these is partial and "in spirit".
 - ▶ If you have not taken these (or do not remember), read up on them as needed!

- ▶ **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](#))
- ▶ **Intuition:** ([take them with a kilo of salt](#))
 - ▶ This is what I assume you know! ([I have to assume something](#))
 - ▶ In many cases, the dependency of AI-1 on these is partial and "in spirit".
 - ▶ If you have not taken these (or do not remember), read up on them as needed!
- ▶ **The real Prerequisite:** Motivation, Interest, Curiosity, hard work. ([AI-1 is non-trivial](#))
- ▶ You can do this course if you want! ([and I hope you are successful](#))

- ▶ Academic Assessment: 90 minutes exam directly after courses end (~ Feb. 10. 2022)
- ▶ Retake Exam: 90 min exam directly after courses end the following semester (~ July. 15. 2022)
- ▶ Mid-semester mini-exam: online, optional, corrected but ungraded, (so you can predict the exam style)
- ▶ Module Grade:
 - ▶ Grade via the exam (Klausur) ~ 100% of the grade
 - ▶ Results from “Übungen zu Künstliche Intelligenz” give up to 10% bonus to a passing exam (not passed ~ no bonus)
- ▶ I do not think that this is the best possible scheme, but I have very little choice.

AI-1 Homework Assignments

- ▶ **Homeworks:** will be small individual problem/programming/proof assignments
(but take time to solve) group submission if and only if explicitly permitted.
- ▶ **⚠ Double Jeopardy ⚠:** Homeworks only give 10% bonus points for the exam, but without trying you are unlikely to pass the exam.
- ▶ **Admin:** To keep things running smoothly
 - ▶ Homeworks will be posted on StudOn.
 - ▶ Sign up for AI-1 under <https://www.studon.fau.de/crs4074217.html>.
 - ▶ Homeworks are handed in electronically there. (plain text, program files, PDF)
 - ▶ **Go to the tutorials, discuss with your TA!** (they are there for you!)
- ▶ **Homework Discipline:**
 - ▶ **Start early!** (many assignments need more than one evening's work)
 - ▶ Don't start by sitting at a blank screen
 - ▶ Humans will be trying to understand the text/code/math when grading it.

Tutorials for Artificial Intelligence 1

- ▶ Weekly tutorials and homework assignments (first one in week two)
- ▶ Instructor/Lead TA: Florian Rabe (florian.rabe@fau.de) Room: 11.137 @ Händler building
- ▶ Tutorials: one each taught by Florian Rabe (lead); Matthias Lobenhofer, Ankitha Reddy, Mark Tintemann, Zilu Zhao
- ▶ Goal 1: Reinforce what was taught in class (you need practice)
- ▶ Goal 2: Allow you to ask any question you have in a small and protected environment
- ▶ Life-saving Advice: go to your tutorial, and prepare for it by having looked at the slides and the homework assignments

Textbook, Handouts and Information, Forums, Video

- ▶ **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 **third edition**, which is vastly improved over earlier ones.

Textbook, Handouts and Information, Forums, Video

- ▶ **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 **third edition**, which is vastly improved over earlier ones.
- ▶ **Course notes:** will be posted at
<http://kwarc.info/teaching/AI/notes.pdf>
 - ▶ more detailed than [RN09] in some areas
 - ▶ I mostly prepare them as we go along (**semantically preloaded** \rightsquigarrow **research resource**)
 - ▶ please e-mail me any errors/shortcomings you notice. (improve for the group)

Textbook, Handouts and Information, Forums, Video

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

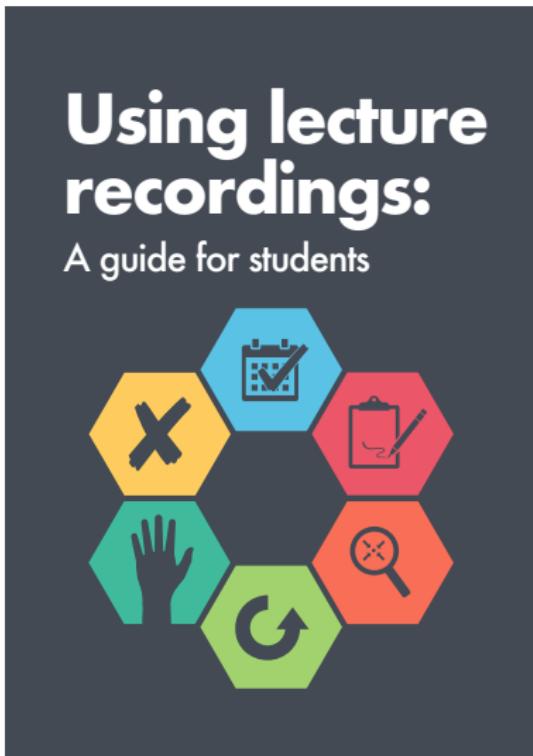
Make sure that you read the **third edition**, which is vastly improved over earlier ones.
- ▶ **Course notes:** will be posted at
<http://kwarc.info/teaching/AI/notes.pdf>
 - ▶ more detailed than [RN09] in some areas
 - ▶ I mostly prepare them as we go along (**semantically preloaded** \rightsquigarrow **research resource**)
 - ▶ please e-mail me any errors/shortcomings you notice. **(improve for the group)**
- ▶ **Course Forum:** on StudOn: <https://www.studon.fau.de/crs4074217.html> for Announcements, homeworks, discussions

Textbook, Handouts and Information, Forums, Video

- ▶ **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 **third edition**, which is vastly improved over earlier ones.
- ▶ **Course notes:** will be posted at
<http://kwarc.info/teaching/AI/notes.pdf>
 - ▶ more detailed than [RN09] in some areas
 - ▶ I mostly prepare them as we go along (**semantically preloaded** \rightsquigarrow **research resource**)
 - ▶ please e-mail me any errors/shortcomings you notice. **(improve for the group)**
- ▶ **Course Forum:** on StudOn: <https://www.studon.fau.de/crs4074217.html> for Announcements, homeworks, discussions
- ▶ **Course Videos:**
 - ▶ **New and shiny:** Video course nuggets are available at
<https://fau.tv/course/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>

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



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

Special Admin Conditions

- ▶ Some degree programs do not “import” the course Artificial Intelligence, and thus you may not be able to register for the exam via <https://campus.fau.de>.
 - ▶ Just send me an e-mail and come to the exam, we will issue a “Schein”.
 - ▶ Tell your program coordinator about AI-1/2 so that they remedy this situation
- ▶ In “Wirtschafts-Informatik” you can only take AI-1 and AI-2 together in the “Wahlpflichtbereich”.
- ▶ ECTS credits need to be divisible by five $\leftrightarrow 7.5 + 7.5 = 15$.

Chapter 2 Format of the AI Course/Lecturing

Flipped/Inverted Classroom: Activating Students in Online Courses

- ▶ **Observation:** Traditional lecture styles work less well online/hybrid.
 - ▶ big classroom lectures – just via Zoom: interaction difficult (I cannot see you)
 - ▶ watch-my-video-of-past-lectures: 90 min video put students to sleep.
- ▶ New format: Flipped classroom. (better activation/interaction)
- ▶ **Definition 0.1.** A **flipped classroom** is an instructional strategy and a type of **blended learning** focused on student engagement and active learning. Usually, students prepare **plenary sessions** with online materials and use classroom time for discussions, applications and worked exercises.
- ▶ **Definition 0.2.** **Blended learning** is an approach to education that combines online educational materials and opportunities for interaction online with traditional place-based classroom methods.

Flipped Classroom: An Experiment for AI-1

- ▶ What does this mean concretely for AI-1?
- ▶ we will have plenary sessions during “class times” (Tue 16:15-17:??, Wed 12:15-13:??)
- ▶ Students prepare for plenary sessions with (please really do that)
 - ▶ course notes: <https://kwarc.info/teaching/AI/notes.pdf>
 - ▶ videos nuggets: <https://fau.tv/course/1690>
 - ▶ the Russell/Norvig Book [RN09]
- ▶ The “reading/watching list” is on StudOn, but also subscribe to the StudOn Calendar: https://www.studon.fau.de/studon/calendar.php?client_id=StudOn&token=223af71cd80ac98552e0714dc1fde1cb
- ▶ In the plenary sessions we discuss (streamed, not recorded)
 - ▶ student questions (if there is something you did not understand)
 - ▶ my discussion challenges/questionnaires (up next)
- ▶ Application and practice in the AI-1 tutorials (go there)

► **Question:** How many scientific articles (6-page double-column “papers”) were submitted to the 2020 International Joint Conference on Artificial Intelligence (IJCAI’20; online in Montreal)?

- a) 7? (6 accepted for publication)
- b) 811? (205 accepted accepted for publication)
- c) 1996? (575 accepted accepted for publication)
- d) 4717? (592 accepted accepted for publication)

► **Question:** How many scientific articles (6-page double-column “papers”) were submitted to the 2020 International Joint Conference on Artificial Intelligence (IJCAI’20; online in Montreal)?

- a) 7? (6 accepted for publication)
- b) 811? (205 accepted accepted for publication)
- c) 1996? (575 accepted accepted for publication)
- d) 4717? (592 accepted accepted for publication)

► **Answer:** (d) is correct. ((c) if for IJCAI’15 ...)

Questionnaire

- ▶ **Question:** How many scientific articles (6-page double-column “papers”) were submitted to the 2020 International Joint Conference on Artificial Intelligence (IJCAI’20; online in Montreal)?
 - a) 7? (6 accepted for publication)
 - b) 811? (205 accepted accepted for publication)
 - c) 1996? (575 accepted accepted for publication)
 - d) 4717? (592 accepted accepted for publication)
- ▶ **Answer:** (d) is correct. ((c) if for IJCAI’15 ...)
- ▶ **Questionnaires** are my attempt to get you to interact in **plenary sessions**.
 - ▶ I will send random groups of 4 into breakout rooms
 - ▶ You get 2 -5 minutes to discuss in the room
 - ▶ I will ask representatives from some of the rooms present their findings.

Call for Help/Ideas with/for Questionnaires

- ▶ I have some questionnaires . . . ,

Call for Help/Ideas with/for Questionnaires

- ▶ I have some questionnaires . . . , but more would be good!
- ▶ I made some good ones . . . ,

Call for Help/Ideas with/for Questionnaires

- ▶ I have some questionnaires . . . , but more would be good!
- ▶ I made some good ones . . . , but better ones would be better
- ▶ Please help me with your ideas (I am not Stefan Raab)

Call for Help/Ideas with/for Questionnaires

- ▶ I have some questionnaires . . . , but more would be good!
- ▶ I made some good ones . . . , but better ones would be better
- ▶ Please help me with your ideas (I am not Stefan Raab)
- ▶ You know something about AI-1 by then.

Call for Help/Ideas with/for Questionnaires

- ▶ I have some questionnaires . . . , but more would be good!
- ▶ I made some good ones . . . , but better ones would be better
- ▶ Please help me with your ideas (I am not Stefan Raab)
- ▶ You know something about AI-1 by then.
- ▶ You know when you would like to break the lecture by a questionnaire.

Call for Help/Ideas with/for Questionnaires

- ▶ I have some questionnaires . . . , but more would be good!
- ▶ I made some good ones . . . , but better ones would be better
- ▶ Please help me with your ideas (I am not Stefan Raab)
- ▶ You know something about AI-1 by then.
- ▶ You know when you would like to break the lecture by a questionnaire.
- ▶ There must be a lot of hidden talent! (you are many, I am only one)

Call for Help/Ideas with/for Questionnaires

- ▶ I have some questionnaires . . . , but more would be good!
- ▶ I made some good ones . . . , but better ones would be better
- ▶ Please help me with your ideas (I am not Stefan Raab)
 - ▶ You know something about AI-1 by then.
 - ▶ You know when you would like to break the lecture by a questionnaire.
 - ▶ There must be a lot of hidden talent! (you are many, I am only one)
 - ▶ I would be grateful just for the idea. (I can work out the details)

Chapter 3 Artificial Intelligence – Who?, What?, When?, Where?, and Why?

- ▶ 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?

1 What is Artificial Intelligence?

What is Artificial Intelligence? Definition

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



What is Artificial Intelligence? Components

- ▶ Elaine Rich: AI studies how we can make the computer do things that humans can still do better at the moment.
- ▶ This needs a combination of
 - ▶ the ability to learn
 - ▶ inference
 - ▶ perception
 - ▶ language understanding
 - ▶ emotion



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



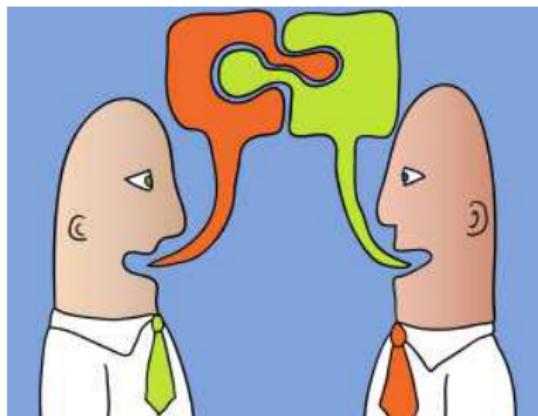
What is Artificial Intelligence? Components

- ▶ 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



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



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

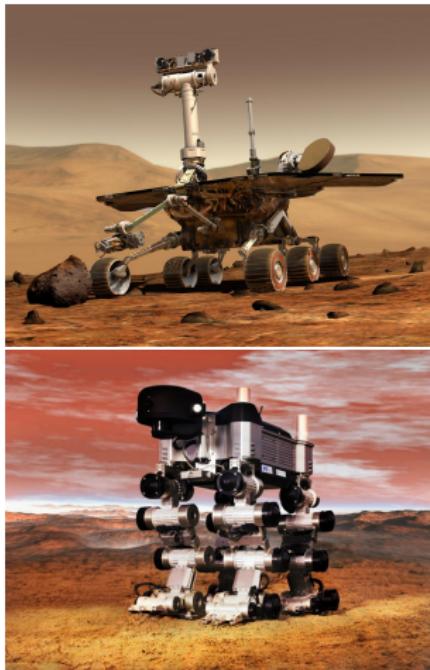


Luisenburg-Festspiele 2004 – "Anatevka" mit Günter Marz und Gisela Ehrenspäger

2 Artificial Intelligence is here today!

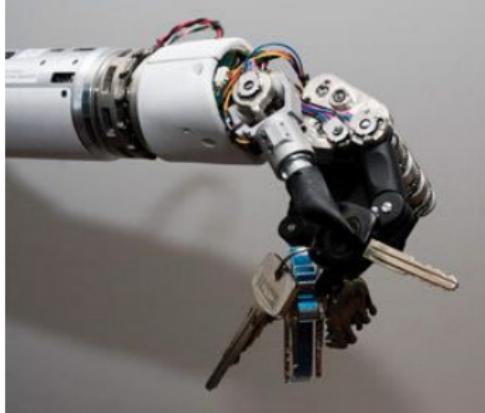
Artificial Intelligence is here today!

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



Artificial Intelligence is here today!

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



Artificial Intelligence is here today!

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



Artificial Intelligence is here today!

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



Artificial Intelligence is here today!

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



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

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



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

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



- ▶ [AlphaGo](#) is a program by Google DeepMind to play the board game [go](#).
In December 2017 [AlphaZero](#), a successor of [AlphaGo](#) “learned” the games [go](#), [chess](#), and [shogi](#) in 24 hours, achieving a superhuman level of play in these three games by defeating world-champion programs.

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



- ▶ [AlphaGo](#) is a program by Google DeepMind to play the board game [go](#).
By September 2019, [AlphaStar](#), a variant of [AlphaGo](#), attained “grandmaster level” in [Starcraft II](#), a real-time strategy game with partially observable state.
AlphaStar now among the top 0.2% of human players.

The AI Conundrum

- ▶ **Observation:** Reserving the term “Artificial Intelligence” has been quite a land-grab!
- ▶ **But:** researchers at the Dartmouth Conference (1950) really thought they would solve AI in two/three decades.
- ▶ **Consequence:** AI still asks the big questions.
- ▶ **Another Consequence:** AI as a field is an incubator for many innovative technologies.
- ▶ **AI Conundrum:** Once AI solves a subfield it is called “Computer Science”.
(becomes a separate subfield of CS)
- ▶ **Example 2.1.** Functional/Logic Programming, Automated Theorem Proving, Planning, Machine Learning, Knowledge Representation, ...
- ▶ **Still Consequence:** AI research was alternately flooded with money and cut off brutally.

3 Ways to Attack the AI Problem

Three Main Approaches to Artificial Intelligence

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

Three Main Approaches to Artificial Intelligence

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

Three Main Approaches to Artificial Intelligence

- ▶ **Definition 3.1.** **Symbolic AI** is based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into structures (expressions) and manipulating them (using processes) to produce new expressions.
- ▶ **Definition 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 3.3.** **Subsymbolic AI** attacks the assumption of **symbolic** and **statistical AI** that intelligence can be achieved by reasoning about the state of the world. Instead it posits that intelligence must be **embodied** – i.e. situated in the world and interact with it via sensors and actuators. The main method for realizing intelligent behavior is by learning from the world, i.e. **machine learning**.

Two ways of reaching Artificial Intelligence?

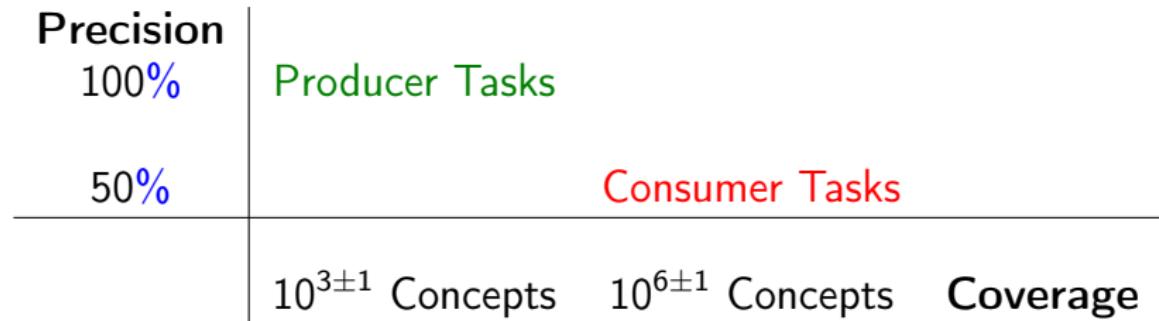
- We can classify the AI approaches by their coverage and the analysis depth (**they are complementary**)

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

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

Environmental Niches for both Approaches to AI

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



- ▶ **General Rule:** **Subsymbolic AI** is well-suited for **consumer tasks**, while **symbolic AI** is better-suited for **producer tasks**.
- ▶ A domain of **producer tasks** I am interested in: Mathematical/Technical Documents.

To get this out of the way . . .



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

4 Strong vs. Weak AI

Strong AI vs. Narrow AI

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

A few words on AGI...

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

- ▶ “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)

5 AI Topics Covered

Topics of AI-1 (Winter Semester)

- ▶ Getting Started
 - ▶ What is Artificial Intelligence
 - ▶ Logic Programming in Prolog
 - ▶ Intelligent Agents
- ▶ Problem Solving
 - ▶ Problem Solving and Search
 - ▶ Adversarial Search (Game playing)
 - ▶ Constraint Satisfaction Problems
- ▶ Knowledge and Reasoning
 - ▶ Formal Logic as the Mathematics of Meaning
 - ▶ Propositional Logic and Satisfiability
 - ▶ First-Order Logic and Theorem Proving
 - ▶ Logic Programming
 - ▶ Description Logics and Semantic Web
- ▶ Planning
 - ▶ Planning
 - ▶ Planning and Acting in the real world

(situating ourselves)

(An influential paradigm)

(a unifying framework)

(Black Box World States and Actions)

(A nice application of Search)

(Factored World States)

(Atomic Propositions)

(Quantification)

(Logic + Search \leadsto Programming)

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

AI1SysProj: A Systems/Project Supplement to AI-1

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

AI1SysProj: A Systems/Project Supplement to AI-1

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

AI1SysProj: A Systems/Project Supplement to AI-1

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

AI1SysProj: A Systems/Project Supplement to AI-1

- ▶ The AI-1 course concentrates on concepts, theory, and algorithms of symbolic AI
- ▶ **Problem:** Engineering/Systems Aspects of AI are very important as well.
- ▶ **Partial Solution:** Getting your hands dirty in the homeworks and the Kalah Challenge
- ▶ **Full Solution:** AI1SysProj: AI-1 Systems Project (10 ECTS, 30-50 places)
 - ▶ For each Topic of AI-1, there 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 \cong 250-300 hours involvement by definition (1/3 of your time/ECTS)

AI1SysProj: A Systems/Project Supplement to AI-1

- ▶ The AI-1 course concentrates on concepts, theory, and algorithms of symbolic AI
- ▶ **Problem:** Engineering/Systems Aspects of AI are very important as well.
- ▶ **Partial Solution:** Getting your hands dirty in the homeworks and the Kalah Challenge
- ▶ **Full Solution:** AI1SysProj: AI-1 Systems Project (10 ECTS, 30-50 places)
 - ▶ For each Topic of AI-1, there 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 \cong 250-300 hours involvement by definition (1/3 of your time/ECTS)
- ▶ **BTW:** There will also be an AI2SysProj next semester! (another chance)

6 AI in the KWARC Group

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

Overview: KWARC Research and Projects

Applications: eMath 3.0, Active Documents, Active Learning, Semantic Spreadsheets/CAD/CAM, Change Management, Global Digital Math Library, Math Search Systems, [SMGloM](#): Semantic Multilingual Math Glossary, Serious Games, ...

Foundations of Math:

- ▶ [MathML](#), [OpenMath](#)
- ▶ advanced Type Theories
- ▶ [MMT](#): Meta Meta Theory
- ▶ Logic Morphisms/Atlas
- ▶ Theorem Prover/CAS Interoperability
- ▶ Mathematical Models/Simulation

KM & Interaction:

- ▶ Semantic Interpretation (aka. Framing)
- ▶ math-literate interaction
- ▶ MathHub: math archives & active docs
- ▶ Semantic Alliance: embedded semantic services

Semantization:

- ▶ [LATEXML](#): [LATEX](#) → XML
- ▶ [STE](#): Semantic [LATEX](#)
- ▶ invasive editors
- ▶ Context-Aware IDEs
- ▶ Mathematical Corpora
- ▶ Linguistics of Math
- ▶ ML for Math Semantics Extraction

Foundations: Computational Logic, Web Technologies, [OMDoc/MMT](#)

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

Part I Getting Started with AI: A Conceptual Framework

Enough philosophy about “Intelligence” (Artificial or Natural)

- ▶ So far we had a nice philosophical chat, about “intelligence” et al.
- ▶ As of today, we look at technical work
- ▶ Naturally we will not start with the most complex action-decision framework...
- ▶ ...but the simplest possible one: “General Problem Solving via Search”
- ▶ Despite its simplicity it is highly relevant in practice.

Chapter 4 Logic Programming

1 Introduction to Logic Programming and ProLog

Logic Programming

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

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

- ▶ How to achieve this?: Restrict the logic calculus sufficiently that it can be used as computational procedure.
- ▶ Slogan: Computation = Logic + Control ([Kowalski '73])
- ▶ We will use the programming language ProLog as an example.

- ▶ **Definition 1.2.** ProLog programs express knowledge about the world via
 - ▶ constants denoted by lower-case strings,
 - ▶ variables denoted by upper-case strings or starting with underscore, and
 - ▶ functions and predicates (lowercase strings) applied to terms.
- ▶ **Definition 1.3.** A ProLog term is
 - ▶ a ProLog variable, or constant, or
 - ▶ a ProLog function applied to terms.A ProLog literals is a constant or a predicate applied to terms.
- ▶ **Example 1.4.** The following are
 - ▶ ProLog terms: john, X, _, father(john), ...
 - ▶ ProLog literals: loves(john,mary), loves(john,_), loves(john,wife_of(john)),...

- ▶ **Definition 1.5.** A ProLog **program** is a sequence of **clauses**, i.e.
 - ▶ **facts** of the form $l..$, where l is a **literal**, (a literal and a dot)
 - ▶ **rules** of the form $h:-b_1,\dots,b_n$, where h is called the **head literal** (or simply **head**) and the b_i are called the **body literals** of the rule.

A rule $h:-b_1,\dots,b_n$, should be read as h (is true) if b_1 and ... and b_n are.

- ▶ **Example 1.6.** 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.

- ▶ **Definition 1.7.** 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.

Querying the Knowledge Base: Size Matters

- ▶ **Idea:** We want to see whether a term is in the knowledge base.
- ▶ **Definition 1.8.** A **query** is a list of ProLog terms called **goal literals** (also **subgoals** or simply **goals**). We write a query as $?-A_1, \dots, A_n$. where A_i are goals.
- ▶ **Problem:** Knowledge bases can be big and even **infinite**. (cannot pre-compute)
- ▶ **Example 1.9.** The **knowledge base** induced by the ProLog program

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

contains the **facts** $\text{nat}(\text{zero})$, $\text{nat}(\text{s}(\text{zero}))$, $\text{nat}(\text{s}(\text{s}(\text{zero})))$, ...

Querying the Knowledge Base: Backchaining

- ▶ **Definition 1.10.** Given a **query** $Q: ?-A_1, \dots, A_n$. and **rule** $R: h:-b_1, \dots, b_n$, **backchaining** computes a new **query** by
 1. finding **terms** for all variables in h to make h and A_1 equal and
 2. replacing A_1 in Q with the **body literals** of R , where all **variables** are suitably replaced.
- ▶ Backchaining motivates the names **goal/subgoal**:
 - ▶ the **literals** in the **query** are “goals” that have to be satisfied,
 - ▶ **backchaining** does that by replacing them by new “goals”.
- ▶ **Definition 1.11.** 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. (anser: false)
- ▶ **Example 1.12 (Backchaining).** We continue 1.9

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

Querying the Knowledge Base: Failure

- ▶ If no instance of a **query** can be derived from the **knowledge base**, then the **ProLog** interpreter reports **failure**.
- ▶ **Example 1.13.** We vary 1.12 using 0 instead of zero.

```
?- nat(s(s(0))).  
?- nat(s(0)).  
?- nat(0).
```

FAIL

false

Querying the Knowledge base: Answer Substitutions

- ▶ **Definition 1.14.** If a **query** contains **variables**, then **ProLog** will return an **answer substitution**, i.e the values for all the **query variables** accumulated during repeated **backchaining**.
- ▶ **Example 1.15.** We talk about (Bavarian) cars for a change, and use a **query** with a **variables**

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

PROLOG: Are there Fallible Greeks?

- ▶ Program:

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

- ▶ Example 1.16 (Query). $?-\text{fallible}(X), \text{greek}(X)$.
- ▶ Answer substitution: [sokrates/ X]

2 Programming as Search

2.1 Knowledge Bases and Backtracking

- ▶ So far, all the examples led to direct **success** or to **failure**. (simpl. KB)
- ▶ **Definition 2.1 (ProLog Search Procedure).** The ProLog interpreter employs top-down, left-right **depth first search**, concretely:
 - ▶ work on the subgoals in left-right order.
 - ▶ match first query with the head literals of the clauses in the program in top-down order.
 - ▶ if there are no matches, **fail** and backtrack to the (chronologically) last **backtrack point**.
 - ▶ otherwise backchain on the first match, keep the other matches in mind for backtracking via **backtrack points**.
- ▶ We can force backtracking to get more solutions by typing ;.

Backtracking by Example

- ▶ **Example 2.2.** We extend 1.15:

```
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
```

2.2 Programming Features

Can We Use This For Programming?

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

```
add(X,zero,X).
```

```
add(X,s(Y),s(Z)) :- add(X,Y,Z).
```

- ▶ similarly with multiplication and exponentiation.

```
mult(X,zero,zero).
```

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

```
expt(X,zero,s(zero)).
```

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

More Examples from elementary Arithmetics

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

```
?-add(s(zero),X,s(s(s(zero)))).
```

```
X = s(s(zero))
```

```
true
```

- **Example 2.5.** Computing the n^{th} Fibonacci Number (0, 1, 1, 2, 3, 5, 8, 13, . . . ; add the last two to get the next), using the addition predicate above.

```
fib(zero,zero).
```

```
fib(s(zero),s(zero)).
```

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

- **Example 2.6.** using ProLog’s internal arithmetic: a goal of the form `?- D is e.` where e is a ground arithmetic expression binds D to the result of evaluating e .

```
fib(0,0).
```

```
fib(1,1).
```

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

Adding Lists to ProLog

- ▶ Lists are represented by terms of the form [a,b,c,...]
- ▶ first/rest representation [F|R], where R is a rest list.
- ▶ predicates for member, append and reverse of lists in default ProLog representation.

```
member(X,[X|_]).
```

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

```
append([],L,L).
```

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

```
reverse([],[]).
```

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

- ▶ **Example 2.7.** Parameters have no unique direction “in” or “out”

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

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

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

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

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

2.3 Advanced Relational Programming

Specifying Control in ProLog

- ▶ *Remark 2.10.* The runtime of the program from 2.9 is not $\mathcal{O}(n(\log_2(n)))$ which is optimal for sorting algorithms.

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

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

Functions and Predicates in ProLog

- ▶ **Remark 2.11.** Functions and predicates have radically different roles in ProLog.
 - ▶ Functions are used to represent data. (e.g. `father(john)` or `s(s(zero))`)
 - ▶ Predicates are used for stating properties about and computing with data.
- ▶ **Remark 2.12.** In functional programming, functions are used for both.
(even more confusing than in ProLog if you think about it)

Functions and Predicates in ProLog

- ▶ **Remark 2.11.** Functions and predicates have radically different roles in ProLog.
 - ▶ Functions are used to represent data. (e.g. `father(john)` or `s(s(zero))`)
 - ▶ Predicates are used for stating properties about and computing with data.
- ▶ **Remark 2.12.** In functional programming, functions are used for both.
(even more confusing than in ProLog if you think about it)
- ▶ **Example 2.13.** Consider again the reverse program for lists below:
An input datum is e.g. `[1,2,3]`, then the output datum is `[3,2,1]`.

```
reverse([],[]).
```

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

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

Functions and Predicates in ProLog

- ▶ **Remark 2.11.** Functions and predicates have radically different roles in ProLog.
 - ▶ Functions are used to represent data. (e.g. `father(john)` or `s(s(zero))`)
 - ▶ Predicates are used for stating properties about and computing with data.
- ▶ **Remark 2.12.** In functional programming, functions are used for both.
(even more confusing than in ProLog if you think about it)

- ▶ **Example 2.13.** Consider again the reverse program for lists below:
An input datum is e.g. `[1,2,3]`, then the output datum is `[3,2,1]`.

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

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

- ▶ **Example 2.14 (Trees and Leaf Counting).** We represent (unlabelled) trees via the function `t` from tree lists to trees. For instance, a balanced binary tree of depth 2 is `t([t([t([]),t([])]),t([t([]),t([])])])`. We count leaves by

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

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

- ▶ **RTFM Resources:** There are also lots of good tutorials on the web,
 - ▶ I like [Fis; LPN],
 - ▶ [Fla94] has a very thorough logic-based introduction,
 - ▶ consult also the SWI Prolog Manual [SWI],

Chapter 5 Recap: Complexity Analysis in AI?

Performance and Scaling

- ▶ Suppose we have three algorithms to choose from. (which one to select)
- ▶ Systematic analysis reveals performance characteristics.
- ▶ **Example 0.1.** For a problem of size n (i.e., detecting cycles out of n nodes) we have

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

What?! One year?

- ▶ $2^{10} = 1\,024$ ($1024\mu\text{s} \simeq 1\text{ms}$)
- ▶ $2^{45} = 35\,184\,372\,088\,832$ ($(3.5 \times 10^{13})\mu\text{s} \simeq (3.5 \times 10^7)\text{s} \simeq 1.1\text{Y}$)
- ▶ **Example 0.2.** we denote all times that are longer than the age of the universe with —

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

Recap: Time/Space Complexity of Algorithms

- We are mostly interested in worst-case complexity in AI-1
- Definition: Let $S \subseteq \mathbb{N} \rightarrow \mathbb{N}$ be a set of natural number functions, then we say that an algorithm α that terminates in time $t(n)$ for all inputs of size n has running time $T(\alpha) := t$.
We say that α has time complexity in S (written $T(\alpha) \in S$ or colloquially $T(\alpha) = S$), iff $t \in S$. We say α has space complexity in S , iff α uses only memory of size $s(n)$ on inputs of size n and $s \in S$.
- time/space complexity depends on size measures. (no canonical one)
- Definition: The following sets are often used for S in

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

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

For $k' > 2$ and $k > 1$ we have

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

- We expect that given an algorithm, you can determine the complexity class.

Determining the Time/Space Complexity of Algorithms

- ▶ We compute the **time complexity** of an algorithm by induction on the composition of an algorithm α given a measure μ such that $\mu(d) \in \mathbb{N}$ and a complexity context γ that maps variable names v to sets $\Gamma(v)$.
 - ▶ **constant:** If $\alpha = \delta$ for a data constant δ with $\mu(\delta) = n$, $T(\alpha) \in \mathcal{O}(1)$.
 - ▶ **variable:** If $\alpha = v$ with $v \in \text{dom}(\Gamma)$, then $T(\alpha) \in \Gamma(v)$.
 - ▶ **application:** If $\alpha = \varphi(\psi)$ with $T(\varphi) \in \mathcal{O}(f)$ and $T(\psi) \in \mathcal{O}(g)$, then $T(\alpha) \in \mathcal{O}(f \circ g)$.
 - ▶ **assignment:** If α is $v := \varphi$ with $T(\varphi) \in S$, then $\Gamma(v) := S$ and $T(\alpha) \in S$.
 - ▶ **composition:** If α is $\varphi; \psi$, with $T(\varphi) \in P$ and $T(\psi) \in Q$, then $T(\alpha) \in \max\{P, Q\}$.
 - ▶ **branching:** If α is **if** γ **then** φ **else** ψ **fi**, with $T(\gamma) \in C$, $T(\varphi) \in P$, and $T(\psi) \in Q$, then $T(\alpha) \in \max\{C, P, Q\}$
 - ▶ **looping:** If α is **while** γ **do** φ **end**, with $T(\gamma) \in \mathcal{O}(f)$ and $T(\varphi) \in \mathcal{O}(g)$, then $T(\alpha) \in \mathcal{O}(f(n)g(n))$

Why Complexity Analysis? (General)

- ▶ **Example 0.3.** Once upon a time I was trying to invent an efficient algorithm.
 - ▶ My first algorithm attempt didn't work, so I had to try harder.



Why Complexity Analysis? (General)

- ▶ **Example 0.3.** Once upon a time I was trying to invent an efficient algorithm.
 - ▶ My first algorithm attempt didn't work, so I had to try harder.
 - ▶ But my 2nd attempt didn't work either, which got me a bit agitated.



Why Complexity Analysis? (General)

- ▶ **Example 0.3.** Once upon a time I was trying to invent an efficient algorithm.
 - ▶ My first algorithm attempt didn't work, so I had to try harder.
 - ▶ But my 2nd attempt didn't work either, which got me a bit agitated.
 - ▶ The 3rd attempt didn't work either...



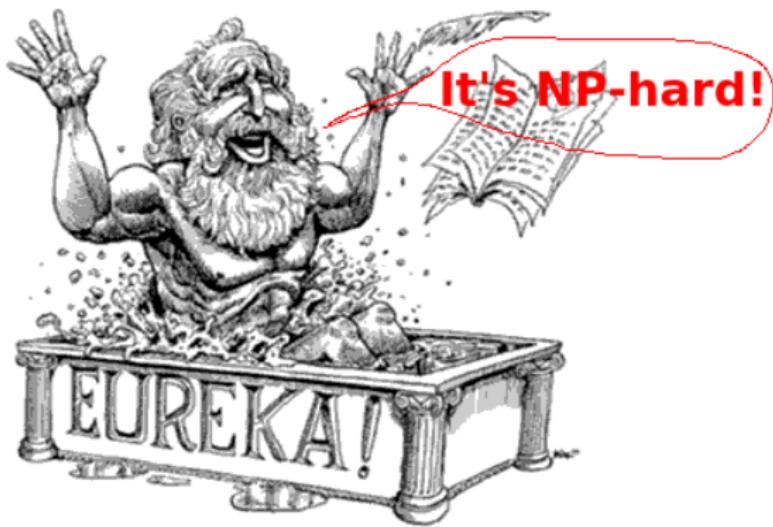
Why Complexity Analysis? (General)

- ▶ **Example 0.3.** Once upon a time I was trying to invent an efficient algorithm.
 - ▶ My first algorithm attempt didn't work, so I had to try harder.
 - ▶ But my 2nd attempt didn't work either, which got me a bit agitated.
 - ▶ The 3rd attempt didn't work either...
 - ▶ And neither the 4th. But then:



Why Complexity Analysis? (General)

- ▶ **Example 0.3.** Once upon a time I was trying to invent an efficient algorithm.
 - ▶ My first algorithm attempt didn't work, so I had to try harder.
 - ▶ But my 2nd attempt didn't work either, which got me a bit agitated.
 - ▶ The 3rd attempt didn't work either...
 - ▶ And neither the 4th. But then:
 - ▶ Ta-da ... when, for once, I turned around and looked in the other direction – CAN one actually solve this efficiently? – **NP-hardness** was there to rescue me.



Why Complexity Analysis? (General)

- ▶ **Example 0.4.** 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.

- ▶ Turing Machine: Works on a **tape** consisting of **cells**, across which its Read/Write **head** moves. The machine has internal **states**. There is a **transition function** that specifies – given the current cell content and internal state – what the subsequent internal state will be, how what the R/W head does (write a symbol and/or move). Some internal states are **accepting**.
- ▶ Decision problems are in **NP** if there is a **non-deterministic Turing machine** that halts with an answer after **time** polynomial in the size of its input. Accepts if *at least one* of the possible runs accepts.
- ▶ Decision problems are in **NPSPACE**, if there is a **non-deterministic Turing machine** that runs in **space** polynomial in the size of its input.
- ▶ NP vs. PSPACE: Non-deterministic polynomial space can be simulated in deterministic polynomial space. Thus **PSPACE = NPSPACE**, and hence (trivially) **NP ⊆ PSPACE**.
It is commonly believed that **NP ⊈ PSPACE**. (similar to **P ⊊ NP**)

Chapter 6 Rational Agents: a Unifying Framework for Artificial Intelligence

1 Introduction: Rationality in Artificial Intelligence

What is AI? Going into Details

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

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

- ▶ expressed by four different definitions/quotes:

	Humanly	Rational
Thinking	<i>"The exciting new effort to make computers think ... machines with human-like minds"</i>	<i>"The formalization of mental faculties in terms of computational models"</i>
Acting	<i>"The art of creating machines that perform actions requiring intelligence when performed by people"</i>	<i>"The branch of CS concerned with the automation of appropriate behavior in complex situations"</i>

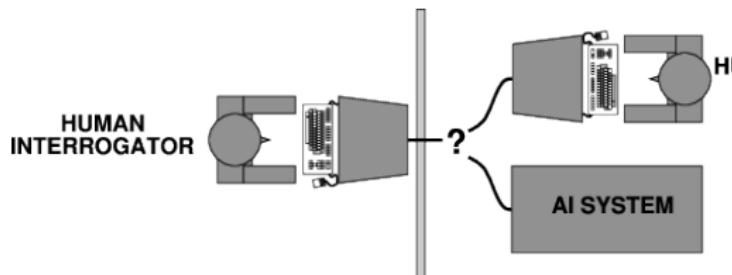
- ▶ Idea: Rationality is performance-oriented rather than based on imitation.

So, what does modern AI do?

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

Acting humanly: The Turing test

- ▶ Alan Turing (1950) "Computing machinery and intelligence" [Tur50]:
- ▶ "Can machines think?" → "Can machines behave intelligently?"
- ▶ **Definition 1.1.** The **Turing test** is an operational test for intelligent behavior based on an **Imitation Game**:



- ▶ Predicted that by 2000, a machine might have a 30% chance of fooling a lay person for 5 minutes
- ▶ Turing anticipated all major arguments against AI in following 50 years
- ▶ 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 (top-down), or
 2. Direct identification from neurological data (bottom-up)
- ▶ **Definition 1.2.** **Cognitive Science** is the interdisciplinary, scientific study of the mind and its processes. It examines the nature, the tasks, and the functions of cognition.
- ▶ **Definition 1.3.** **Cognitive Neuroscience** studies the biological processes and aspects that underlie cognition, with a specific focus on the neural connections in the brain which are involved in mental processes.
- ▶ Both approaches 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

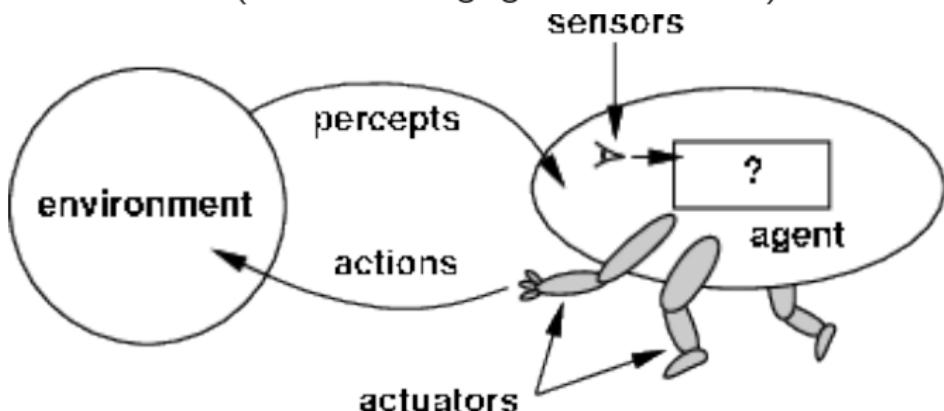
- ▶ Normative (or prescriptive) rather than descriptive
- ▶ Aristotle: what are correct arguments/thought processes?
- ▶ Several Greek schools developed various forms of logic: *notation* and *rules of derivation* for thoughts; may or may not have proceeded to the idea of mechanization.
- ▶ Direct line 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?

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

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

2 Agents and Environments as a Framework for AI

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



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

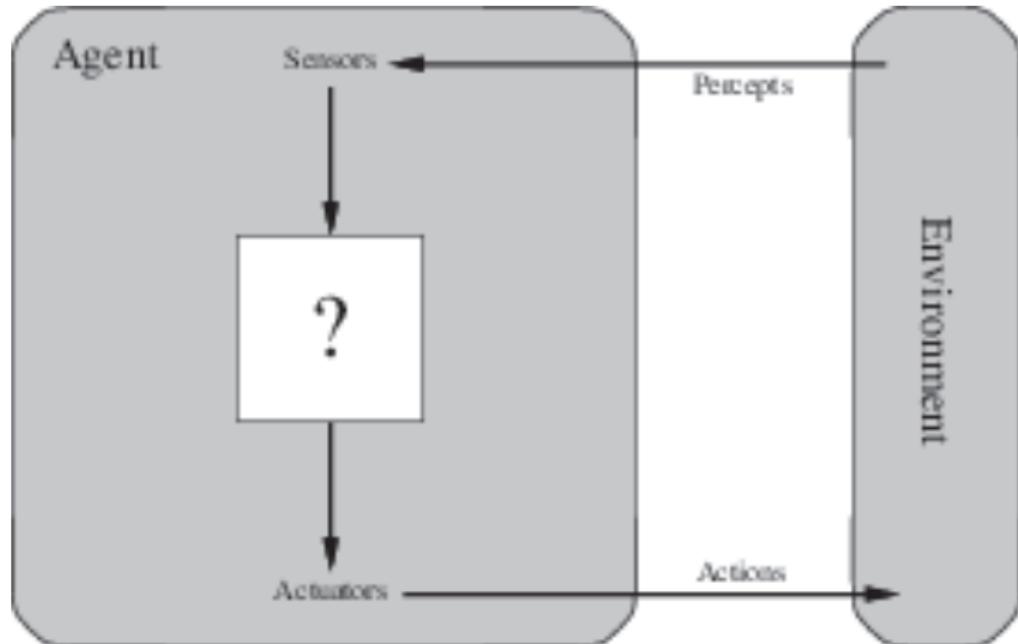
- ▶ **Definition 2.3.** A **percept** is the perceptual input of an **agent** at a specific instant.
- ▶ **Definition 2.4.** Any recognizable, coherent employment of the **actuators** of an **agent** is called an **action**.
- ▶ **Definition 2.5.** The **agent function** f_a of an agent a maps from **percept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

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

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

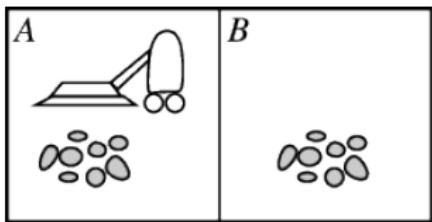
Agent Schema: Visualizing the Internal Agent Structure

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



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

Example: Vacuum-Cleaner World and Agent



- ▶ **percepts:** location and contents, e.g., $[A, Dirty]$
- ▶ **actions:** *Left*, *Right*, *Suck*, *NoOp*
- ▶ Science Question: What is the *right* agent function?
- ▶ AI Question: Is there an agent architecture and an agent program that implements it.

Percept sequence	Action
$[A, Clean]$	<i>Right</i>
$[A, Dirty]$	<i>Suck</i>
$[B, Clean]$	<i>Left</i>
$[B, Dirty]$	<i>Suck</i>
$[A, Clean], [A, Clean]$	<i>Right</i>
$[A, Clean], [A, Dirty]$	<i>Suck</i>
$[A, Clean], [B, Clean]$	<i>Left</i>
$[A, Clean], [B, Dirty]$	<i>Suck</i>
$[A, Dirty], [A, Clean]$	<i>Right</i>
$[A, Dirty], [A, Dirty]$	<i>Suck</i>
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	<i>Right</i>
$[A, Clean], [A, Clean], [A, Dirty]$	<i>Suck</i>
⋮	⋮

Example: Vacuum-Cleaner World and Agent

► Example 2.7 (Agent Program).

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

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

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

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

3 Good Behavior \leadsto Rationality

- ▶ Idea: Try to design agents that are successful (aka. “do the right thing”)
- ▶ Definition 3.1. A **performance measure** is a function that evaluates a sequence of environments.
- ▶ Example 3.2. A **performance measure** for the vacuum cleaner world could
 - ▶ award one point per square cleaned up in time T ?
 - ▶ award one point per clean square per time step, minus one per move?
 - ▶ penalize for $> k$ dirty squares?
- ▶ Definition 3.3. An **agent** is called **rational**, if it chooses whichever **action** maximizes the expected value of the performance measure given the **percept** sequence to date.
- ▶ Question: Why is **rationality** a good quality to aim for?

Consequences of Rationality: Exploration, Learning, Autonomy

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

PEAS: Describing the Task Environment

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

Examples of Agents: PEAS descriptions

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

4 Classifying Environments

Environment types

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

Environment Types (Examples)

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

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

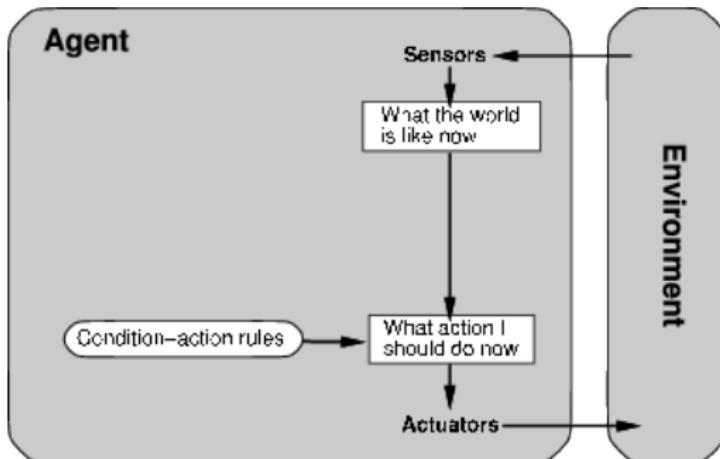
- **Observation 4.4.** The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, and multi agent (worst case for AI)

5 Types of Agents

- ▶ **Observation:** So far we have described (and analyzed) **agents** only by their behavior (cf. **agent function** $f : \mathcal{P}^* \rightarrow \mathcal{A}$).
- ▶ **Problem:** This does not help us to build agents (the goal of AI)
- ▶ **Definition 5.1.** 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 agentsAll these can be turned into **learning agents**.

Simple reflex agents

- ▶ **Definition 5.2.** A **simple reflex agent** is an **agent a** that only bases its actions on the last **percept**: $f_a: \mathcal{P} \rightarrow \mathcal{A}$.
- ▶ **Agent Schema:**



- ▶ **Example 5.3 (Agent Program).**

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

Simple reflex agents (continued)

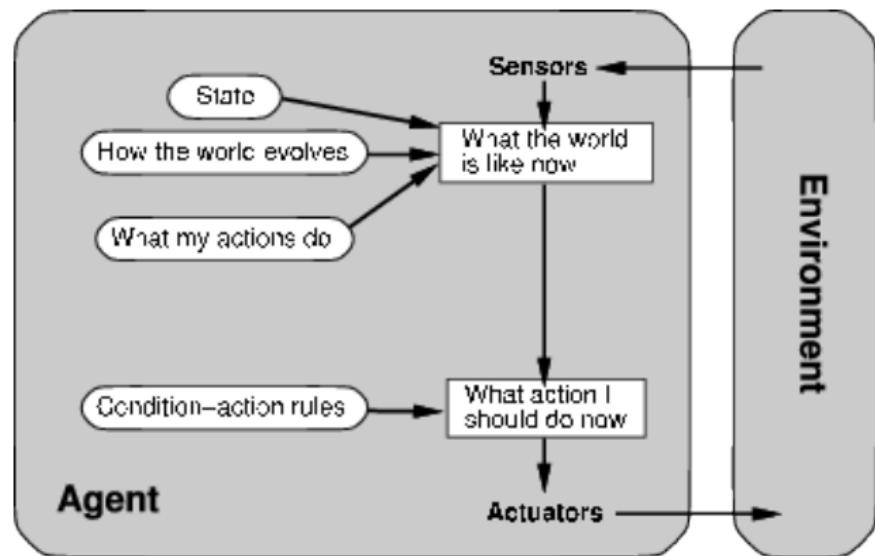
- ▶ General Agent Program:

```
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 5.4. Tail lights signal braking by brightening. A simple reflex agent would have to compare subsequent percepts to realize.
- ▶ Another Problem: Partially observable environments get simple reflex agents into trouble.
- ▶ Example 5.5. Vacuum cleaner robot with defective location sensor \leadsto infinite loops.

Model-based Reflex Agents

- ▶ Idea: Keep track of the **state** of the world we cannot see now in an internal model.
- ▶ **Definition 5.6.** A **model based agent** (also called **reflex agent with state**) whose **agent function** depends on a model of the world (called the **state** or **world model**).
- ▶ Agent Schema:



Model-Based Agents (continued)

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

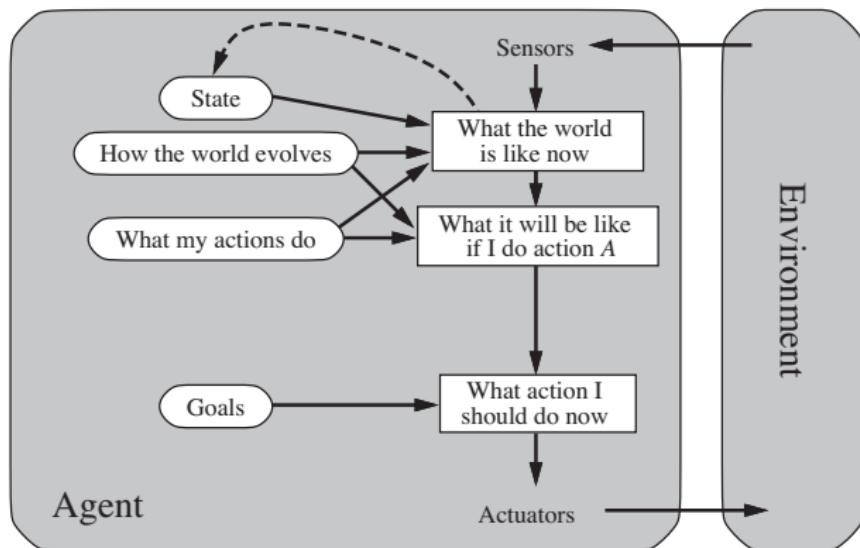
```
function Model-Based-Agent (percept) returns an action
    var state /* a description of the current state of the world */
    persistent rules /* a set of condition-action rules */
    var action /* the most recent action, initially none */

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

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

Goal-based agents

- ▶ **Problem:** Having a model of the world does not always determine what to do (rationally).
- ▶ **Observation:** Having a goal in mind does! (determines future actions)
- ▶ **Definition 5.9.** A **goal based agent** is a **model based agent** that deliberates actions based on **goals** and a **world model**.

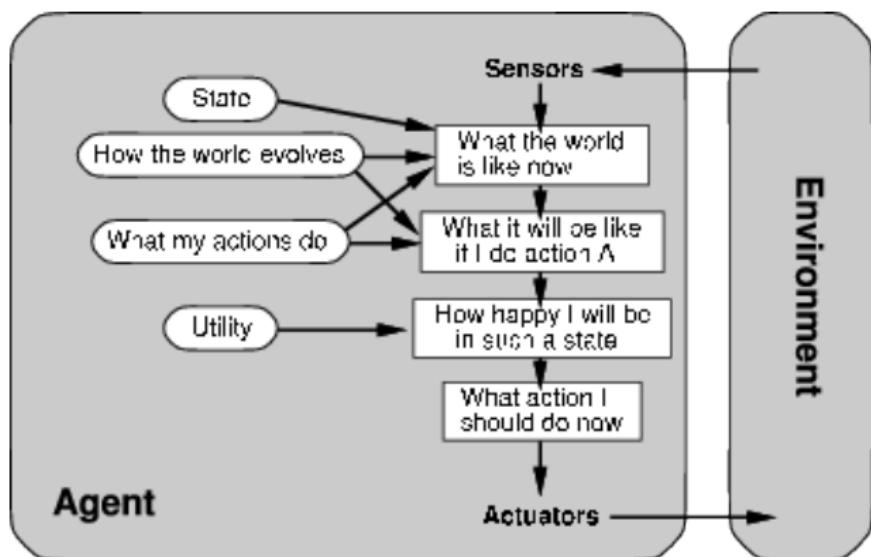


Goal-based agents (continued)

- ▶ **Observation:** A goal based agent is more flexible in the knowledge it can utilize.
- ▶ **Example 5.10.** A goal based agent can easily be changed to go to a new destination, a model based agent's rules make it go to exactly one destination.

Utility-based agents

- ▶ **Definition 5.11.** A **utility based agent** uses a **world model** along with a **utility function** that influences its **preferences** among the **states** of that world. It chooses the **action** that leads to the best **expected utility**, which is computed by averaging over all possible outcome **states**, weighted by the probability of the outcome.
- ▶ **Agent Schema:**



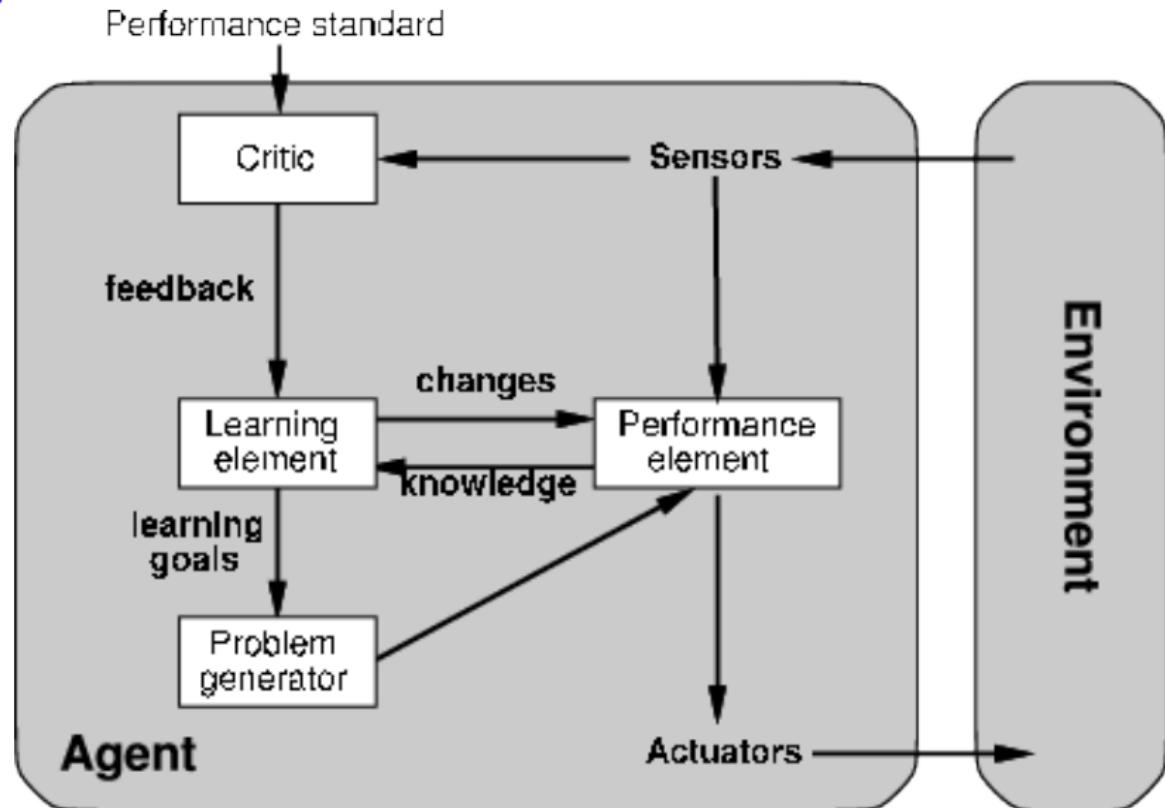
Utility-based vs. Goal-based Agents

- ▶ 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 * likelihood helps)

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

Learning Agents

► Agent Schema:



- ▶ **Example 5.13 (A learning Taxi Agent).** 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 <i>description</i> in a general problem-description language (e.g., the rules of any board game).
More efficient.	VS.	Much less design/maintenance work.

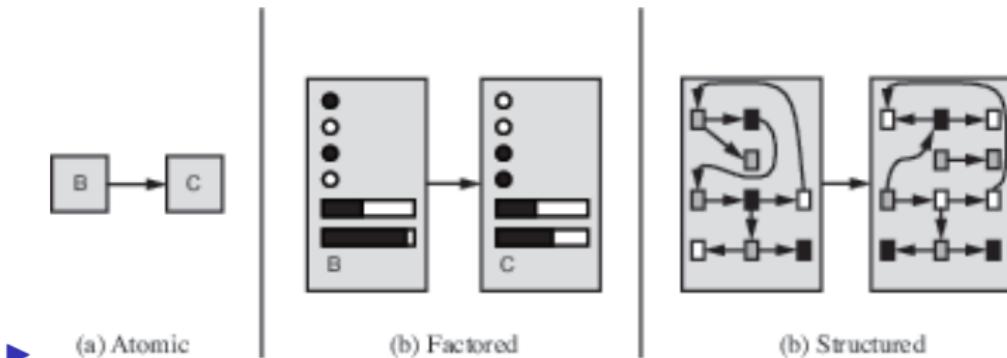
► What kind of agent are you?

6 Representing the Environment in Agents

Representing the Environment in Agents

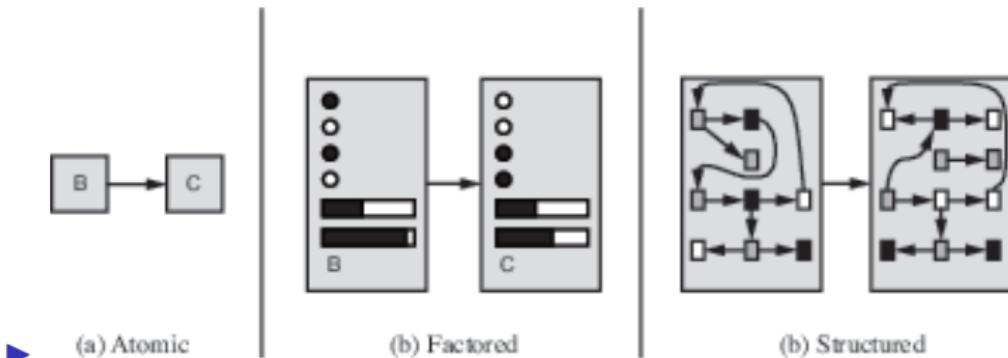
- ▶ We have seen various components of agents that answer questions like
 - ▶ *What is the world like now?*
 - ▶ *What action should I do now?*
 - ▶ *What do my actions do?*
- ▶ Next natural question: How do these work? (see the rest of the course)
- ▶ Important Distinction: What **state** representation
How the agent implement the **world model**.
- ▶ **Definition 6.1.** We call a **state** representation
 - ▶ **atomic**, iff it has no internal structure (black box)
 - ▶ **factored**, iff each **state** is characterized by **attributes** and their **values**.
 - ▶ **structured**, iff the **state** includes objects and their relations.

Atomic/Factored/Structured State Representations



- ▶ **Example 6.2 (Atomic States).** Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities.
- ▶ In an **atomic** representation the **state** is represented by the name of a city

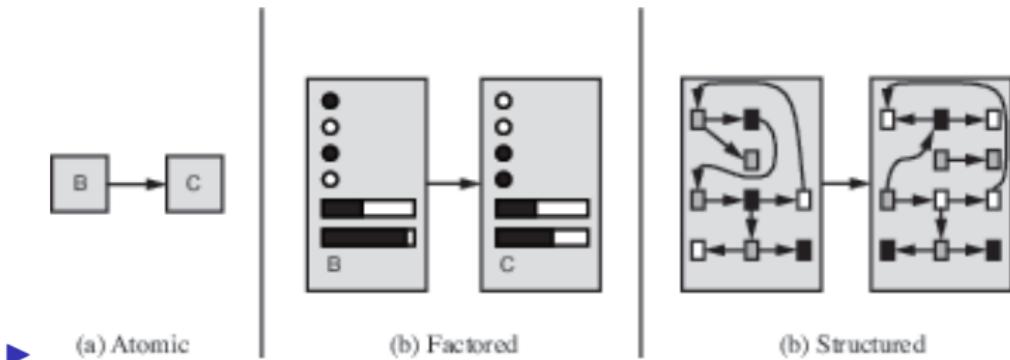
Atomic/Factored/Structured State Representations



► **Example 6.2 (Atomic States).** Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities.

- In an **atomic** representation the **state** is represented by the name of a city
- In a **factored** representation we may have attributes "gps-location", "gas",... (allows information sharing between **states and uncertainty**)
- But how to represent a situation, where a large truck blocking the road, since it is trying to back into a driveway, but a loose cow is blocking its path. (attribute "TruckAheadBackingIntoDairyFarmDrivewayBlockedByLooseCow" is unlikely)

Atomic/Factored/Structured State Representations



- ▶ **Example 6.2 (Atomic States).** 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

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

Part II General Problem Solving

Chapter 7 Problem Solving and Search

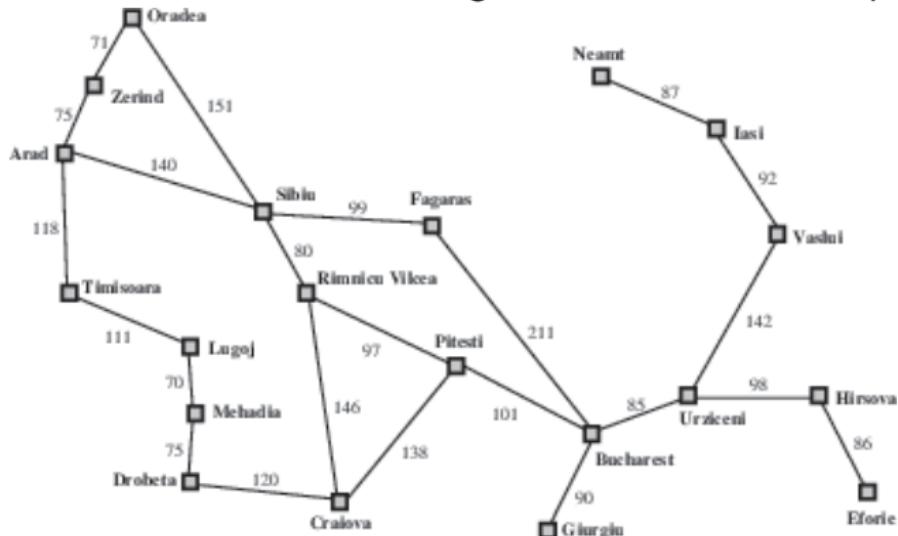
1 Problem Solving

Problem Solving: Introduction

- ▶ Recap: Agents perceive the environment and compute an action.
- ▶ In other words: Agents continually solve “the problem of what to do next”.
- ▶ AI Goal: Find algorithms that help solving problems in general.
- ▶ Idea: If we can describe/represent problems in a standardized way, we may have a chance to find general algorithms.
- ▶ Concretely: We will use the following two concepts to describe problems
 - ▶ States: A set of possible situations in our problem domain ($\hat{=}$ environments)
 - ▶ Actions: that get us from one state to another. ($\hat{=}$ agents)
- A sequence of actions is a solution, if it brings us from an initial state to a goal state. Problem solving computes solutions from problem formulations.
- ▶ Definition 1.1. In offline problem solving an agent computing an action sequence based complete knowledge of the environment.
- ▶ Remark 1.2. Offline problem solving only works in fully observable, deterministic, static, and episodic environments.
- ▶ Definition 1.3. In online problem solving an agent computes one action at a time based on incoming perceptions.
- ▶ This Semester: we largely restrict ourselves to offline problem solving. (easier)

Example: Traveling in Romania

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



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

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

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

Problem solving computes **solutions** from **problem formulations**.

- ▶ Finding the right level of abstraction and the required (not more!) information is often the key to success.

The Math of Problem Formulation: Search Problems

- ▶ **Definition 1.5.** A **search problem** $\Pi := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ consists of a set \mathcal{S} of **states**, a set \mathcal{A} of **actions**, and a **transition model** $\mathcal{T}: \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ that assigns to any action $a \in \mathcal{A}$ and state $s \in \mathcal{S}$ a set of **successor states**. Certain states in \mathcal{S} are designated as **goal states** ($\mathcal{G} \subseteq \mathcal{S}$) and **initial states** $\mathcal{I} \subseteq \mathcal{S}$.
- ▶ **Definition 1.6.** We say that an **action** $a \in \mathcal{A}$ is **applicable** in a **state** $s \in \mathcal{S}$, iff $\mathcal{T}(a, s) \neq \emptyset$. We call $\mathcal{T}_a: \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ with $\mathcal{T}_a(s) := \mathcal{T}(a, s)$ the **result relation** for a and $\mathcal{T}_{\mathcal{A}} := \bigcup_{a \in \mathcal{A}} \mathcal{T}_a$ the **result relation** of Π .
- ▶ **Definition 1.7.** A **solution** for a **search problem** $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ consists of a sequence of **actions** a_1, \dots, a_n such that for all $1 \leq i < n$
 - ▶ a_i is **applicable** to **state** s_{i-1} , where $s_0 \in \mathcal{I}$,
 - ▶ $s_i \in \mathcal{T}_{a_i}(s_{i-1})$, and $s_n \in \mathcal{G}$.
- ▶ Idea: A **solution** bring us from \mathcal{I} to a **goal state**.
- ▶ **Definition 1.8.** Often we add a **cost function** $c: \mathcal{A} \rightarrow \mathbb{R}_0^+$ 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**.

Remark 1.9. Note that search problems formalize problem formulations by making many of the implicit constraints explicit.

Search Problems in deterministic, fully observable Envs

- ▶ This semester, we will restrict ourselves to **search problems**, where (extend in AI-II)
 - ▶ $|\mathcal{T}(a,s)| \leq 1$ for the **transition models** and (↔ deterministic environment)
 - ▶ $\mathcal{I} = \{s_0\}$ (↔ fully observable environment)
- ▶ **Definition 1.10.** Then \mathcal{T}_a induces **partial function** $S_a: \mathcal{S} \rightarrow \mathcal{S}$ whose domain is the set of states where a is **applicable**: $S(s) := s'$ if $\mathcal{T}_a = \{s'\}$ and undefined otherwise.
We call S_a the **successor** function for a and $S_a(s)$ the **successor state** of s .
 $S_{\mathcal{A}} := \bigcup_{a \in \mathcal{A}} S_a$ the **successor relation** of \mathcal{P} .
- ▶ **Definition 1.11.** The predicate that tests for **goal states** is called a **goal test**.

- ▶ **Observation:** $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ from 1.5 is essentially a **blackbox description** (think programming API)
 - ▶ provides the functionality needed to construct a state space.
 - ▶ gives the algorithm no information about the problem
- ▶ **Definition 1.12.** A **declarative description** (also called **whitebox description**) describes the problem itself \leadsto **problem description language**
- ▶ **Example 1.13.** The STRIPS language describes planning problems in terms of
 - ▶ a set P of Boolean variables (propositions)
 - ▶ a set $I \subseteq P$ of propositions true in the initial state
 - ▶ a set $G \subseteq P$, where state $s \subseteq P$ is a goal if $G \subseteq s$
 - ▶ a set a of actions, each $a \in A$ with precondition pre_a , add list add_a , and delete list del_a : a is applicable, if $pre_a \subseteq s$, result state is $s \cup add_a \setminus del_a$
 - ▶ a function c that maps all actions a to their cost $c(a)$.
- ▶ **Observation 1.14.** Declarative descriptions are strictly more powerful than blackbox descriptions: they induce blackbox descriptions, but also allow to analyze/simplify the problem.
- ▶ We will come back to this later \leadsto planning.

2 Problem Types

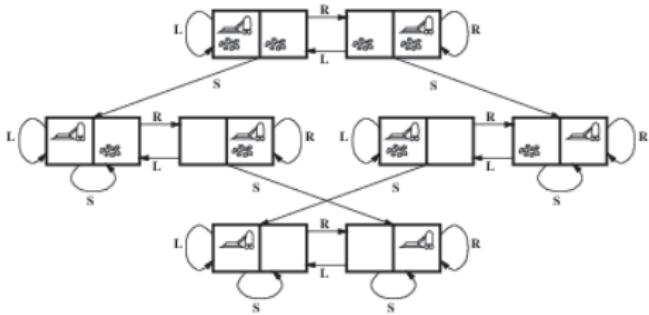
Problem types

- ▶ Single-state problem
 - ▶ observable (at least the initial state)
 - ▶ deterministic (i.e. the successor of each state is determined)
 - ▶ static (states do not change other than by our own actions)
 - ▶ discrete (a countable number of states)
- ▶ Multiple-state problem:
 - ▶ initial state not/partially observable (multiple initial states?)
 - ▶ deterministic, static, discrete
- ▶ Contingency problem:
 - ▶ non-deterministic (solution can branch, depending on contingencies)
 - ▶ unknown state space (like a baby, agent has to learn about states and actions)

Example: vacuum-cleaner world

► Single-state Problem:

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

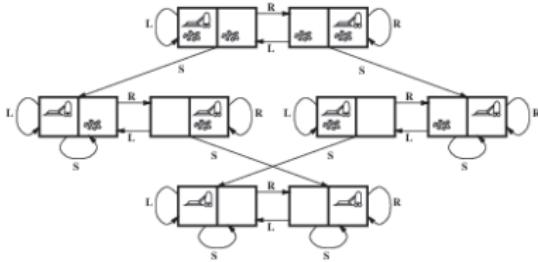


► Multiple-state Problem:

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

Example: Vacuum-Cleaner World (continued)

- ▶ Contingency Problem:
- ▶ Murphy's Law: *suck* can dirty a clean carpet
- ▶ Local sensing: *dirty* / *notdirty* at location only
- ▶ Start in: {1,3}
- ▶ Solution: [*suck*,*right*,*suck*]
 - suck* → {5,7}
 - right* → {6,8}
 - suck* → {6,8}
- ▶ better: [*suck*,*right*,if *dirt* then *suck*] (decide whether in 6 or 8 using local sensing)



Single-state problem formulation

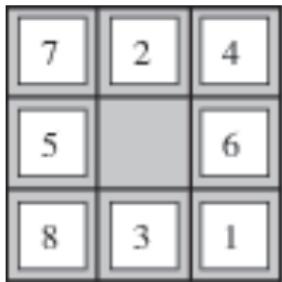
- ▶ Defined by the following four items

1. Initial state: (e.g. Arad)
2. Successor function S : (e.g. $S(Arad) = \{(goZer, Zerind), (goSib, Sibiu), \dots\}$)
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.)

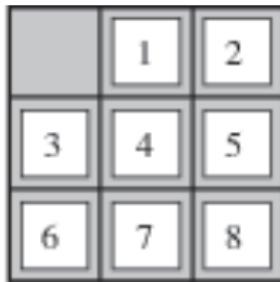
- ▶ Solution: A sequence of actions leading from the initial state to a goal state

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

Example: The 8-puzzle



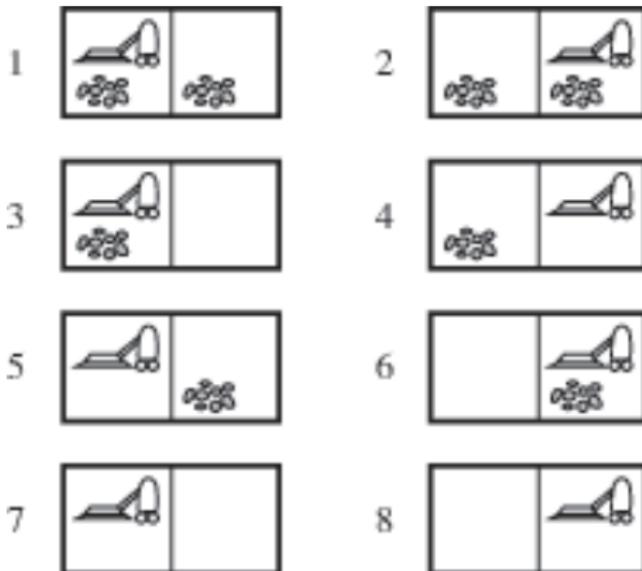
Start State



Goal State

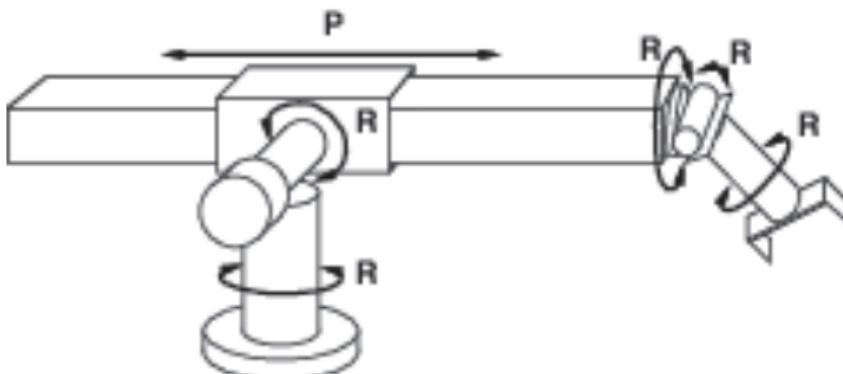
States	integer locations of tiles
Actions	<i>left, right, up, down</i>
Goal test	= goal state?
Path cost	1 per move

Example: Vacuum-cleaner



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

Example: Robotic assembly



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

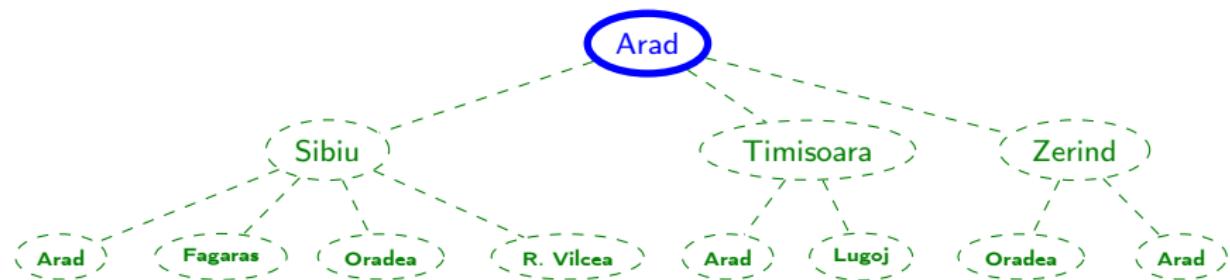
3 Search

Tree Search Algorithms

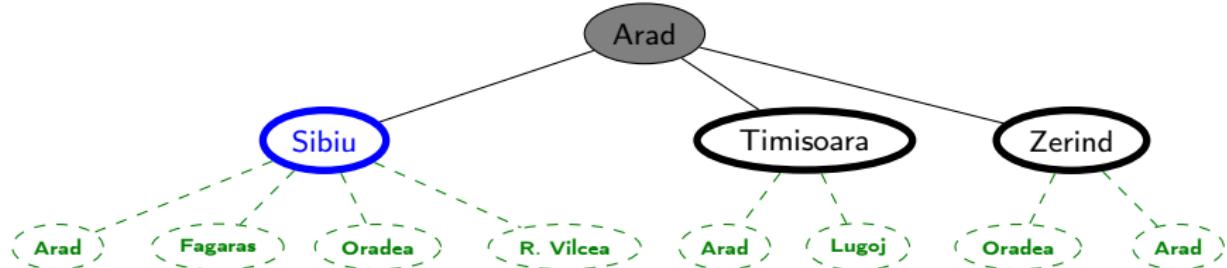
- ▶ **Observation:** The state space of a search problem $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ forms a graph $\langle \mathcal{S}, \mathcal{T}_{\mathcal{A}} \rangle$.
- ▶ As graphs are difficult to compute with, we often compute a corresponding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

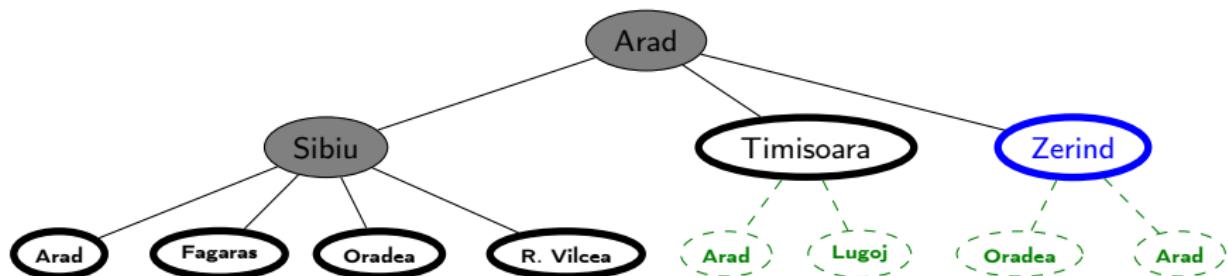
Tree Search: Example



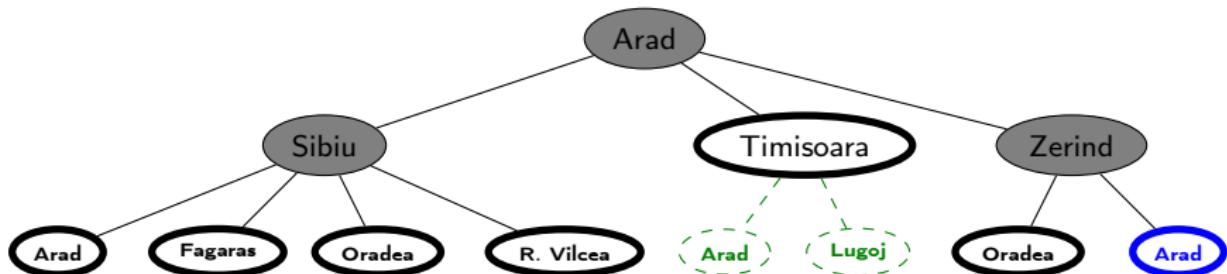
Tree Search: Example



Tree Search: Example

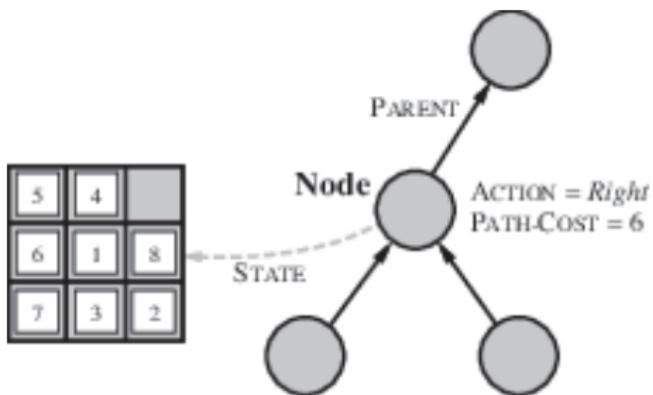


Tree Search: Example



Implementation: States vs. nodes

- ▶ **Recap:** A **state** is a (representation of) a physical configuration.
- ▶ **Definition 3.2.** A **node** is a data structure constituting part of a search tree that includes accessors for **parent**, **children**, **depth**, **path cost**, etc.



- ▶ **Observation:** Paths in the **search tree** correspond to paths in the **state space**.
- ▶ **Definition 3.3.** We define the **path cost** of a **node n** in a **search tree T** to be the sum of the **step costs** on the path from n to the **root** of T . The **cost** of a **solution** is defined analogously.

Implementation of search algorithms

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

- ▶ **Definition 3.4.** The **fringe** is a list **nodes** not yet considered.
- ▶ It is ordered by the **strategy**. (see below)

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

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

- ▶ Time and space complexity measured in terms of:

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

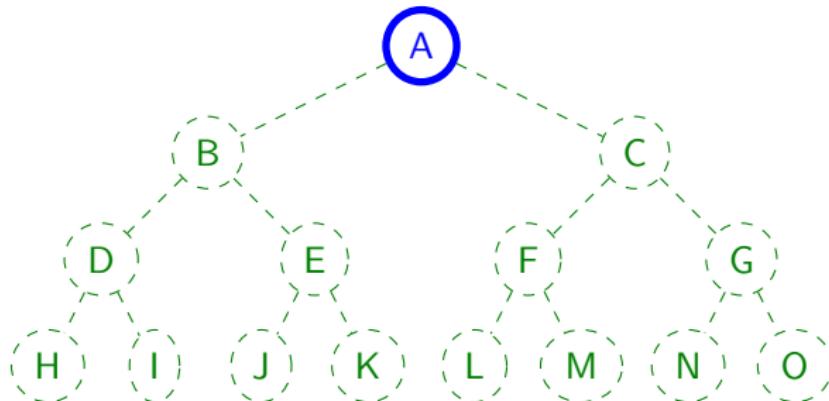
4 Uninformed Search Strategies

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

4.1 Breadth-First Search Strategies

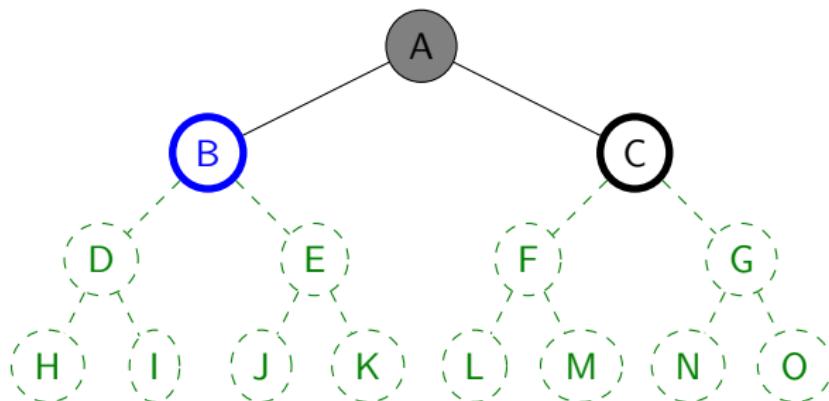
Breadth-First Search

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



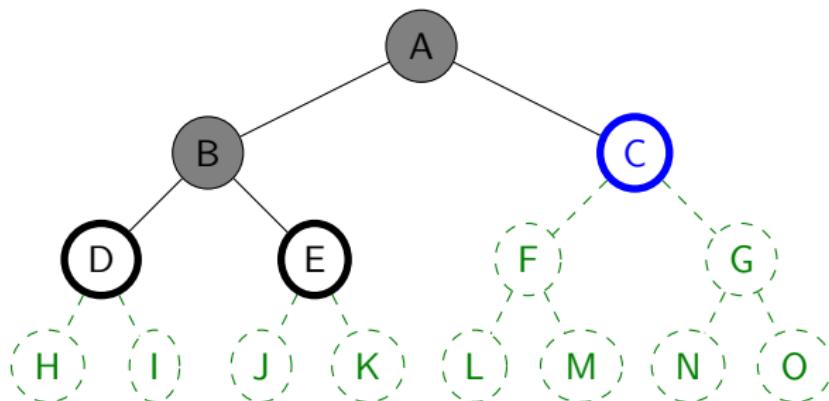
Breadth-First Search

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



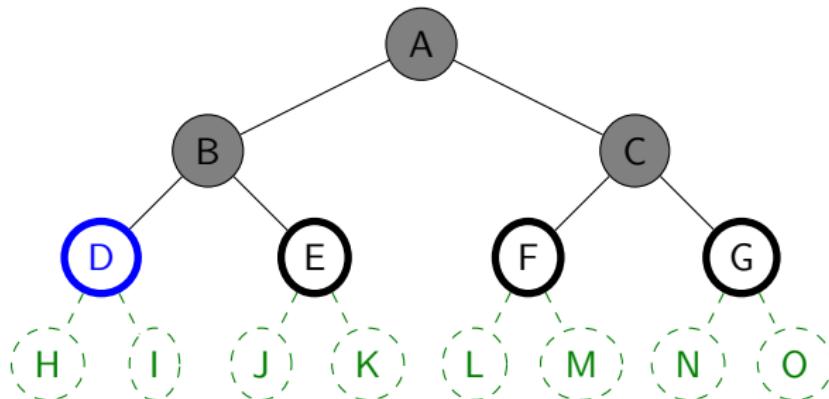
Breadth-First Search

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



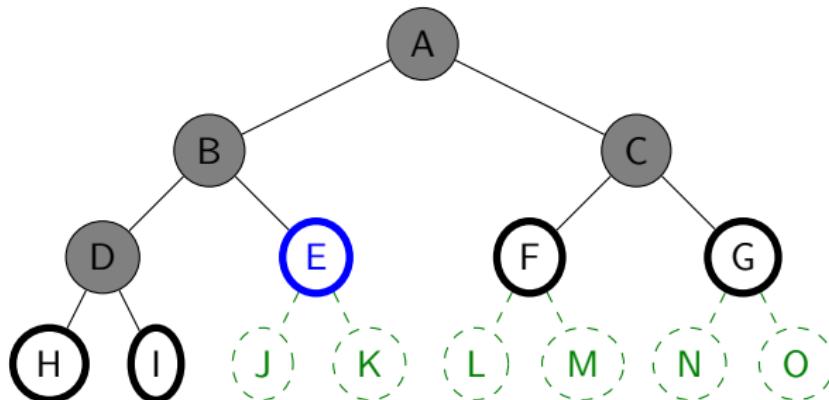
Breadth-First Search

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



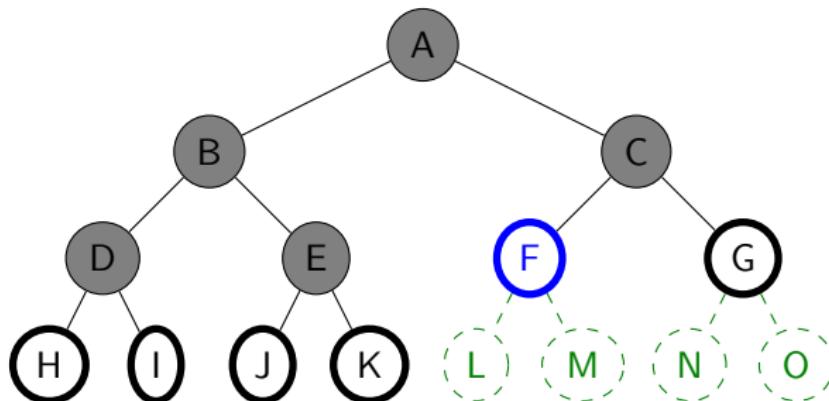
Breadth-First Search

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



Breadth-First Search

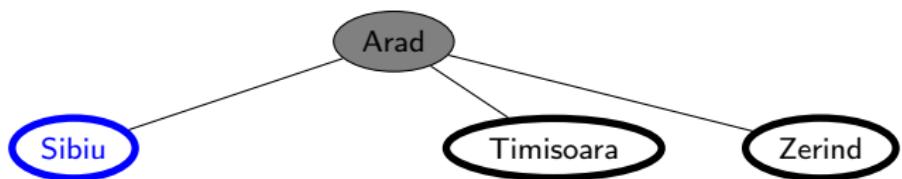
- ▶ **Idea:** Expand the shallowest unexpanded node.
- ▶ **Definition 4.2.** The **breadth-first search (BFS)** strategy treats the **fringe** as a **FIFO queue**, i.e. **successors** go in at the end of the **fringe**.
- ▶ **Example 4.3 (Synthetic).**



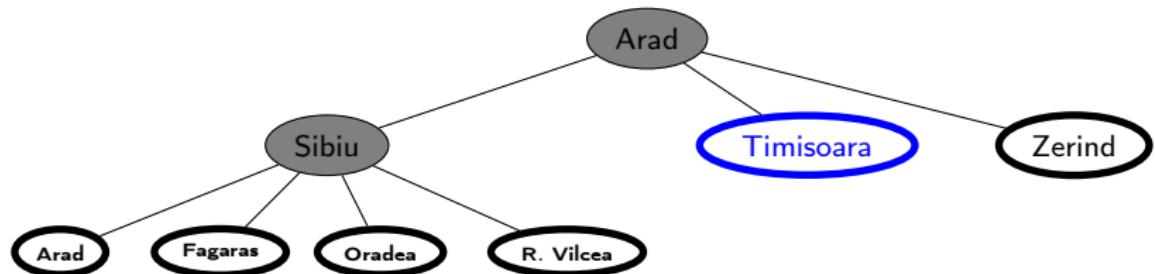
Breadth-First Search: Romania

Arad

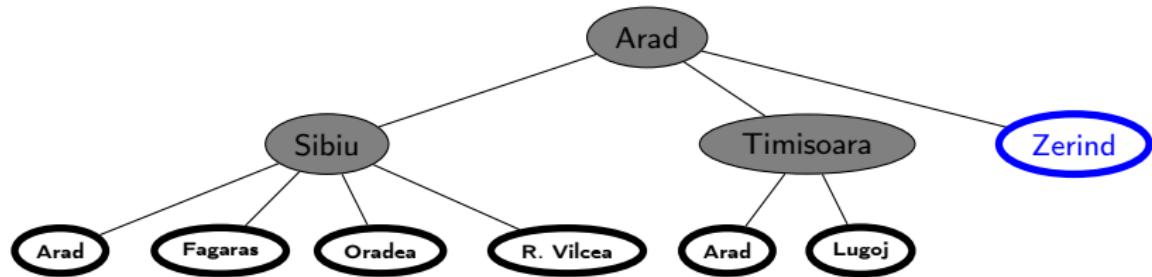
Breadth-First Search: Romania



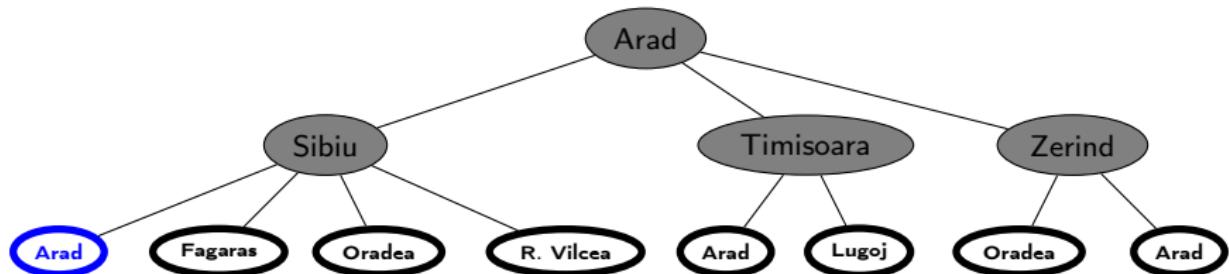
Breadth-First Search: Romania



Breadth-First Search: Romania



Breadth-First Search: Romania

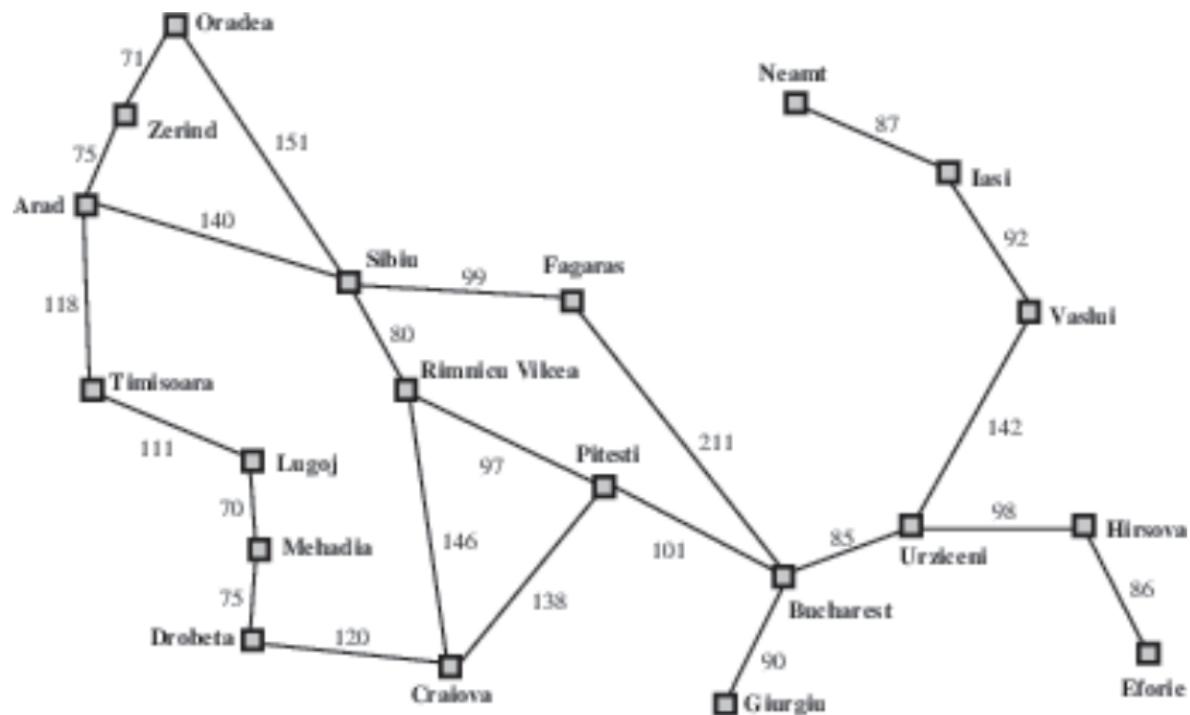


Breadth-first search: Properties

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

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

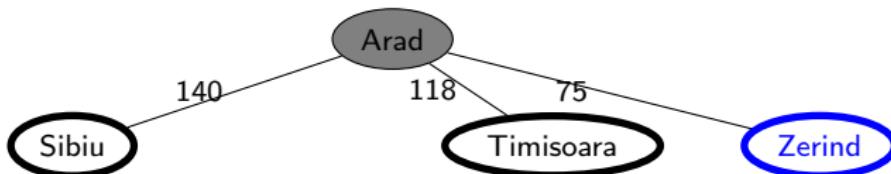
Romania with Step Costs as Distances



- ▶ **Idea:** Expand least-cost unexpanded node
- ▶ **Definition 4.4.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- ▶ **Remark 4.5 (Note).** Equivalent to breadth-first search if all step costs are equal.
- ▶ **Synthetic Example:**

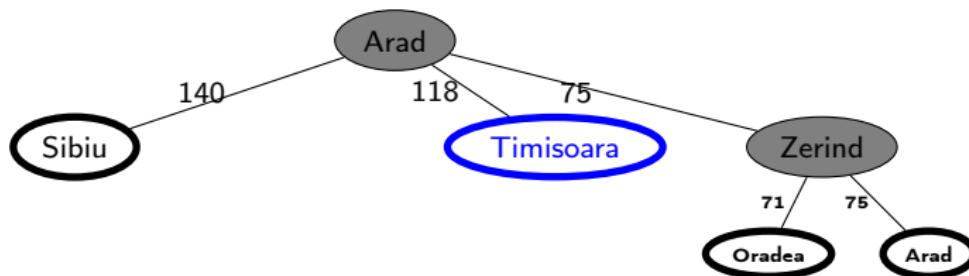


- ▶ Idea: Expand least-cost unexpanded node
- ▶ **Definition 4.4.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- ▶ *Remark 4.5 (Note).* Equivalent to breadth-first search if all step costs are equal.
- ▶ Synthetic Example:



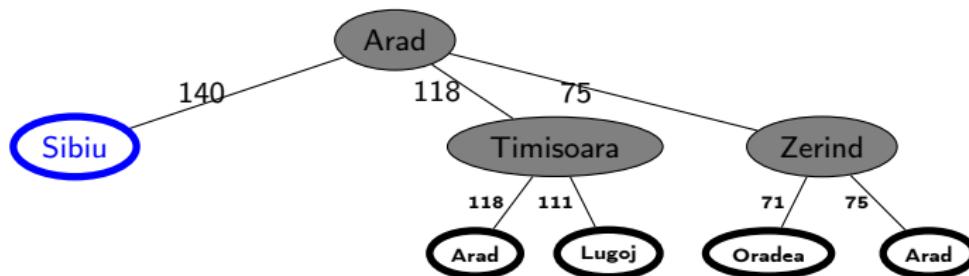
Uniform-cost search

- ▶ **Idea:** Expand least-cost unexpanded node
- ▶ **Definition 4.4.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- ▶ **Remark 4.5 (Note).** Equivalent to breadth-first search if all step costs are equal.
- ▶ **Synthetic Example:**



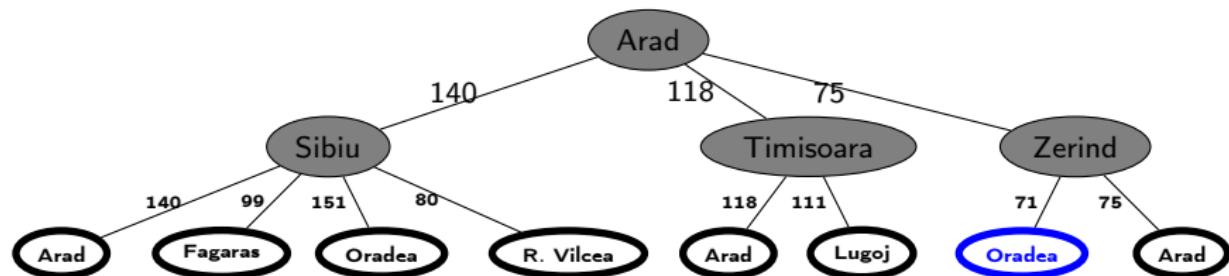
Uniform-cost search

- ▶ **Idea:** Expand least-cost unexpanded node
- ▶ **Definition 4.4.** Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- ▶ **Remark 4.5 (Note).** Equivalent to breadth-first search if all step costs are equal.
- ▶ **Synthetic Example:**



Uniform-cost search

- ▶ Idea: Expand least-cost unexpanded node
- ▶ Definition 4.4. Uniform-cost search (UCS) is the strategy where the fringe is ordered by increasing path cost.
- ▶ Remark 4.5 (Note). Equivalent to breadth-first search if all step costs are equal.
- ▶ Synthetic Example:



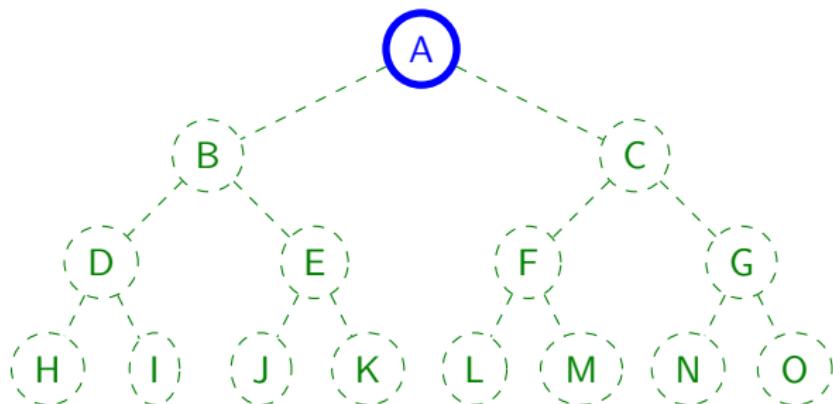
Uniform-cost search: Properties

Complete	Yes (if step costs $\geq \epsilon > 0$)
Time	number of nodes with path-cost less than that of optimal solution
Space	number of nodes with path-cost less than that of optimal solution
Optimal	Yes

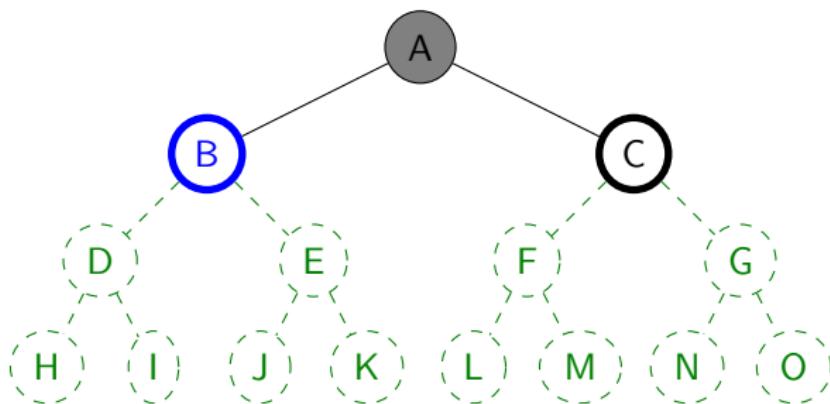
4.2 Depth-First Search Strategies

- ▶ Idea: Expand deepest unexpanded node.
- ▶ **Definition 4.6.** Depth-first search (DFS) is the strategy where the fringe is organized as a (LIFO) stack i.e. successor go in at front of the fringe.
- ▶ Note: Depth-first search can perform infinite cyclic excursions
Need a finite, non-cyclic search space (or repeated-state checking)

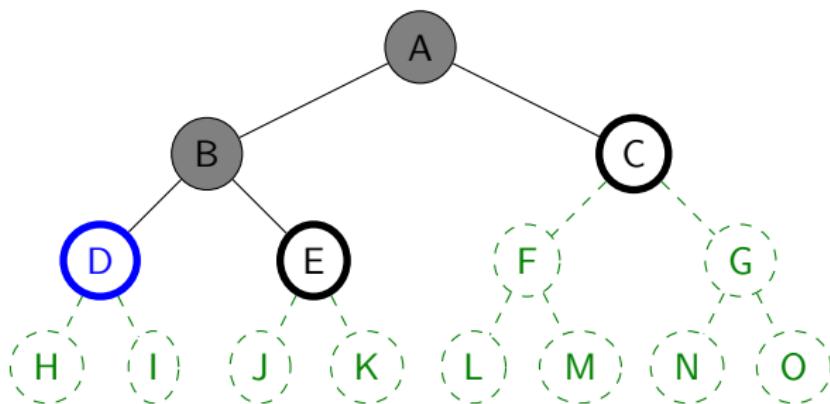
► Example 4.7 (Synthetic).



► Example 4.7 (Synthetic).

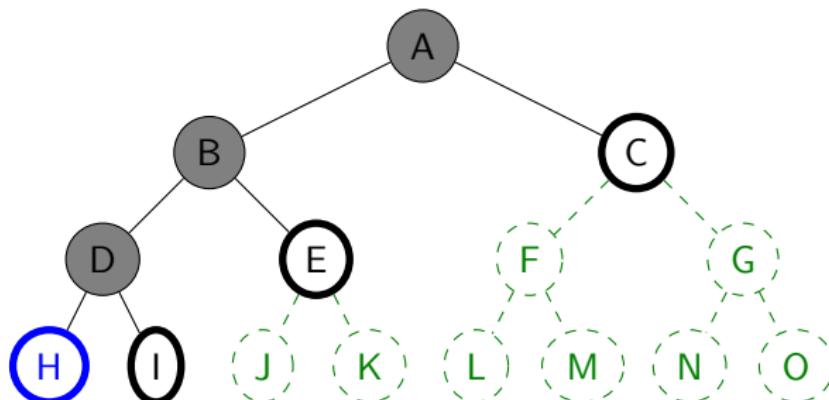


► Example 4.7 (Synthetic).

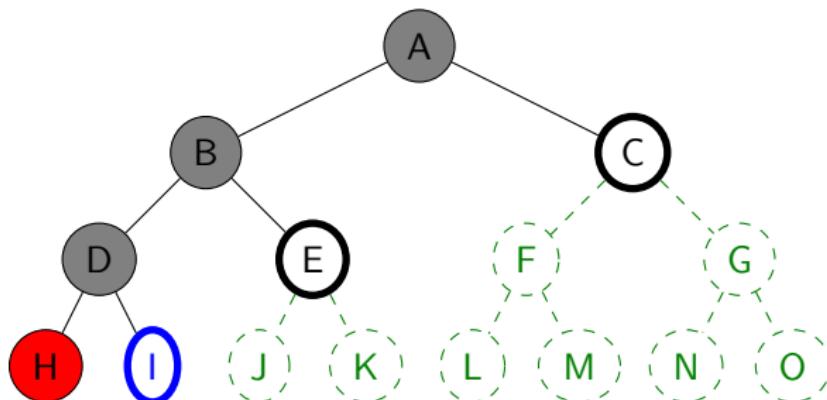


Depth-First Search

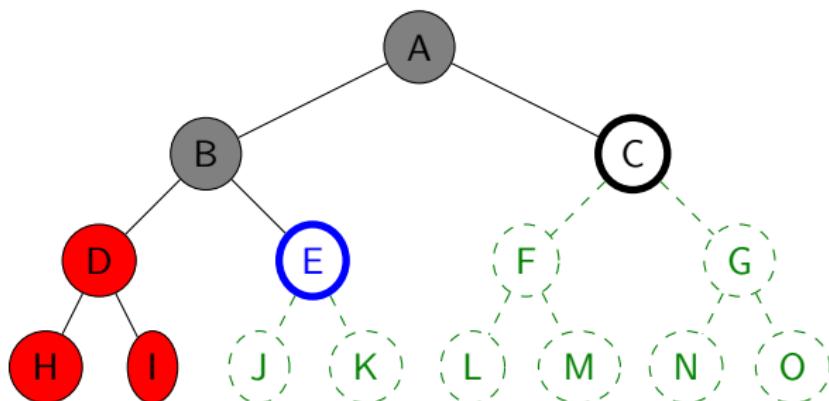
► Example 4.7 (Synthetic).



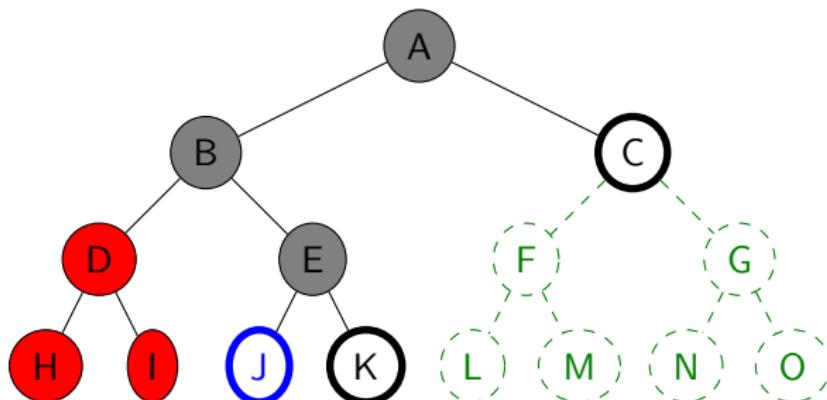
► Example 4.7 (Synthetic).



► Example 4.7 (Synthetic).

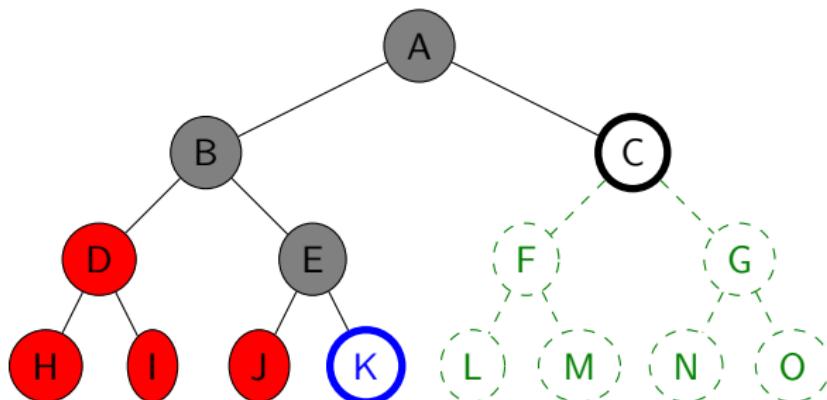


► Example 4.7 (Synthetic).

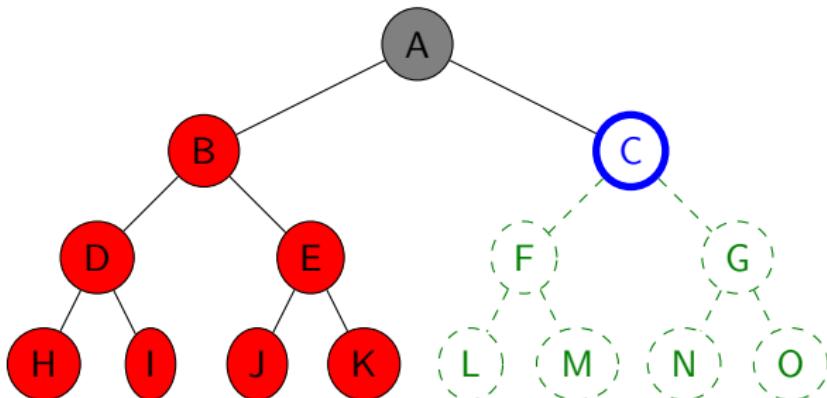


Depth-First Search

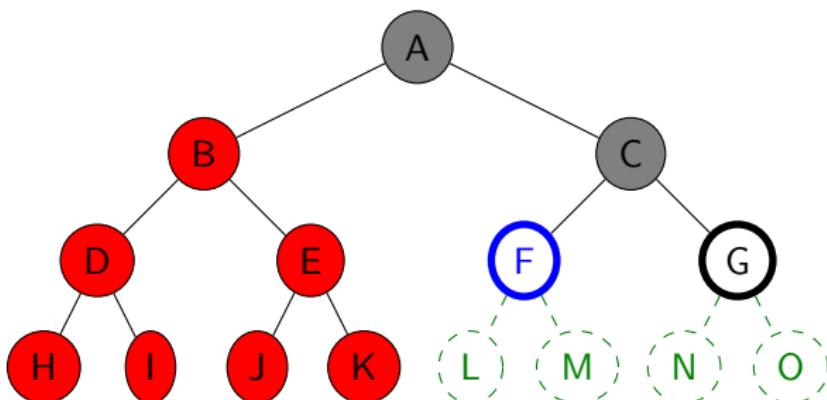
► Example 4.7 (Synthetic).



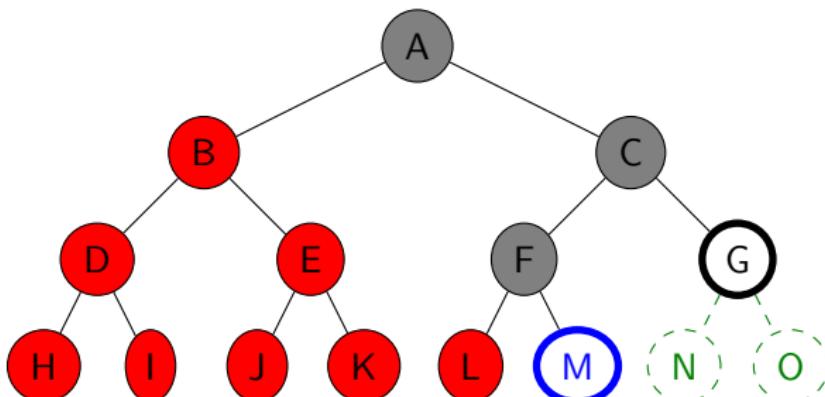
► Example 4.7 (Synthetic).



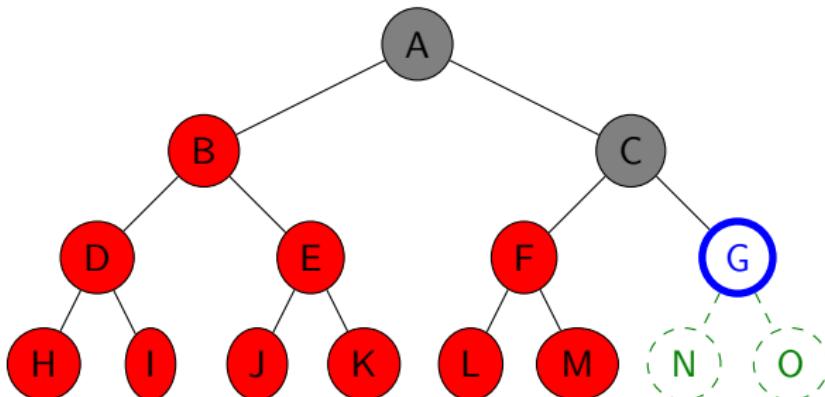
► Example 4.7 (Synthetic).



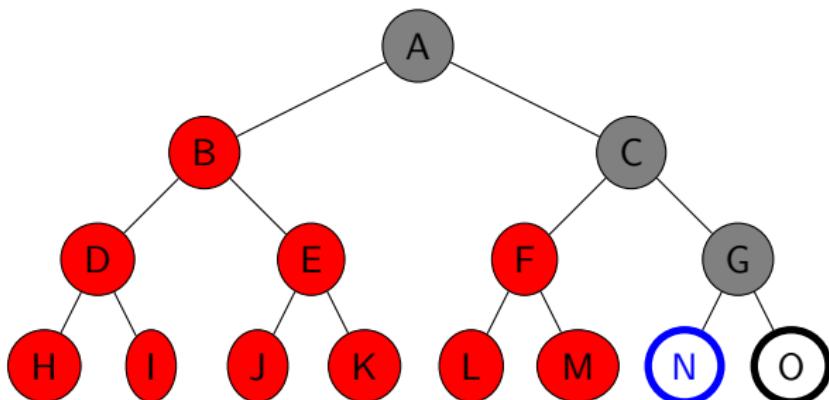
► Example 4.7 (Synthetic).



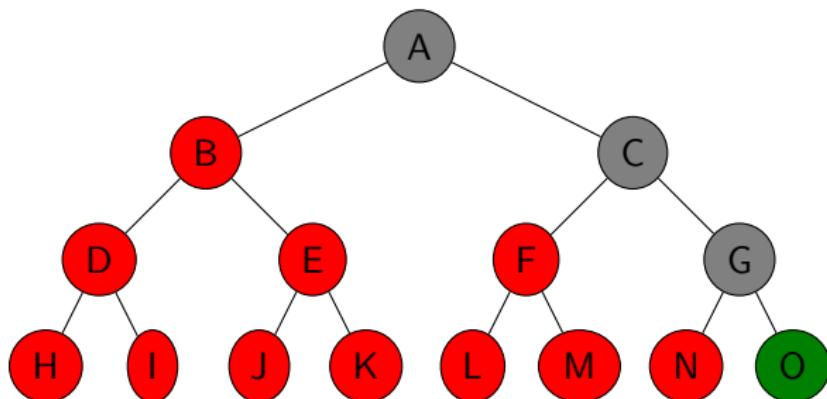
► Example 4.7 (Synthetic).



► Example 4.7 (Synthetic).



► Example 4.7 (Synthetic).

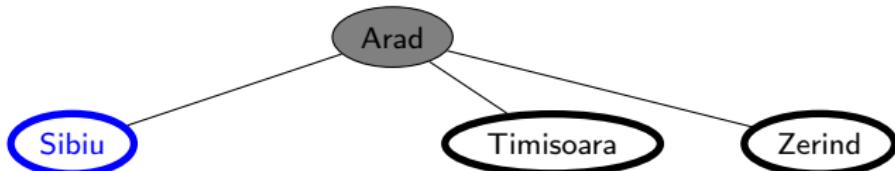


- ▶ Example 4.8 (Romania).



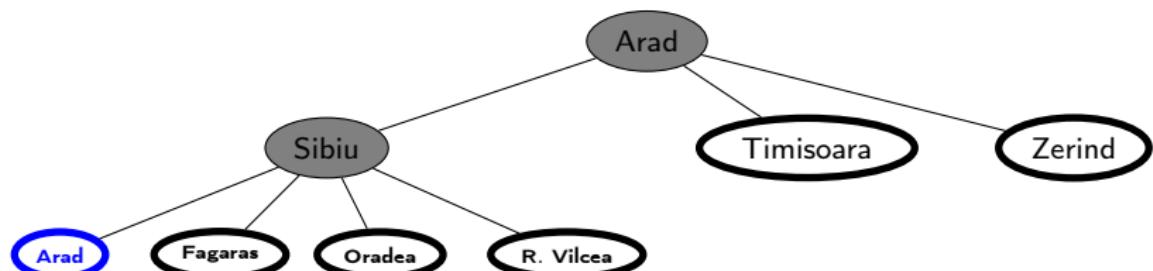
Depth-First Search: Romania

- ▶ Example 4.8 (Romania).



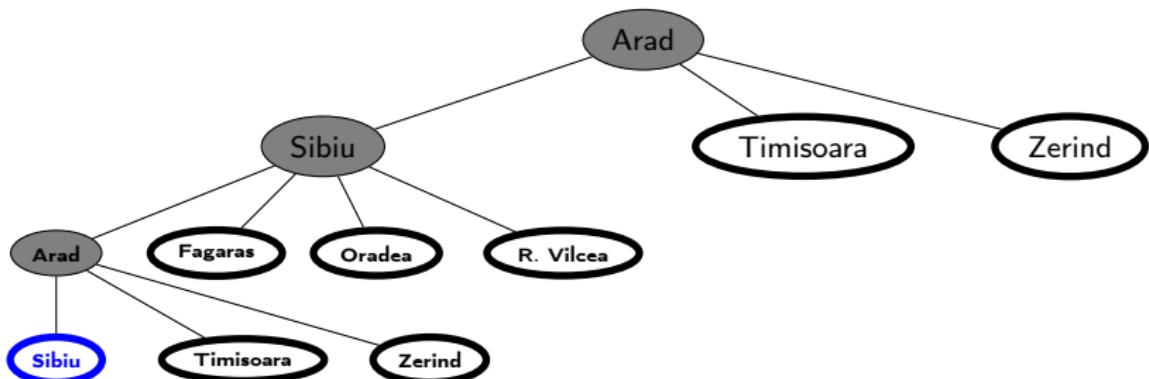
Depth-First Search: Romania

- ▶ Example 4.8 (Romania).



Depth-First Search: Romania

► Example 4.8 (Romania).



Depth-first search: Properties

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

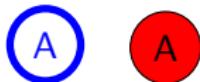
- ▶ **Disadvantage:** Time terrible if m much larger than d .
- ▶ **Advantage:** Time may be much less than **breadth-first search** if solutions are dense.

- ▶ **Definition 4.9.** Depth limited search is depth-first search with a depth limit.
- ▶ **Definition 4.10.** Iterative deepening search (IDS) is depth limited search with ever increasing limits.
- ▶ **procedure** Tree_Search (problem)

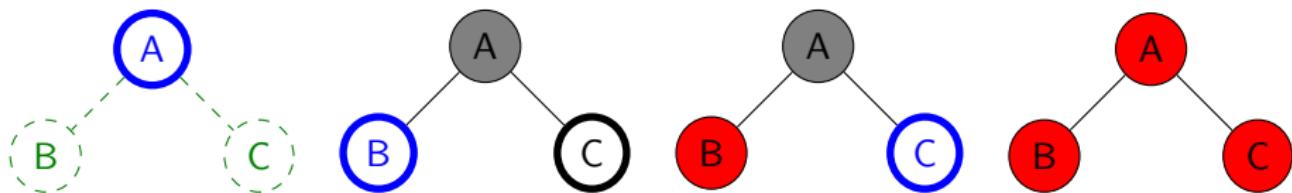
<initialize the search tree using the initial state of problem>

```
for depth = 0 to  $\infty$ 
    result := Depth_Limited_search(problem,depth)
    if depth  $\neq$  cutoff return result end if
end for
end procedure
```

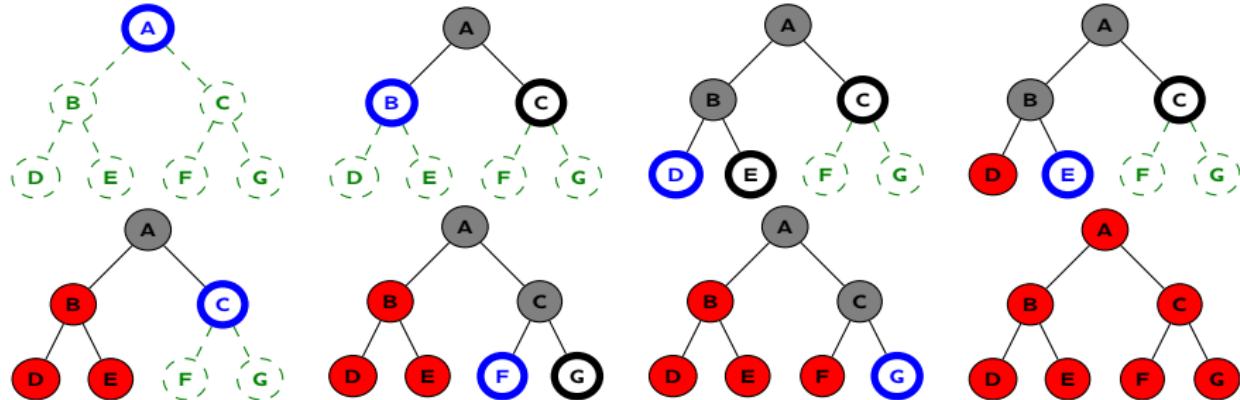
Iterative Deepening Search at various Limit Depths



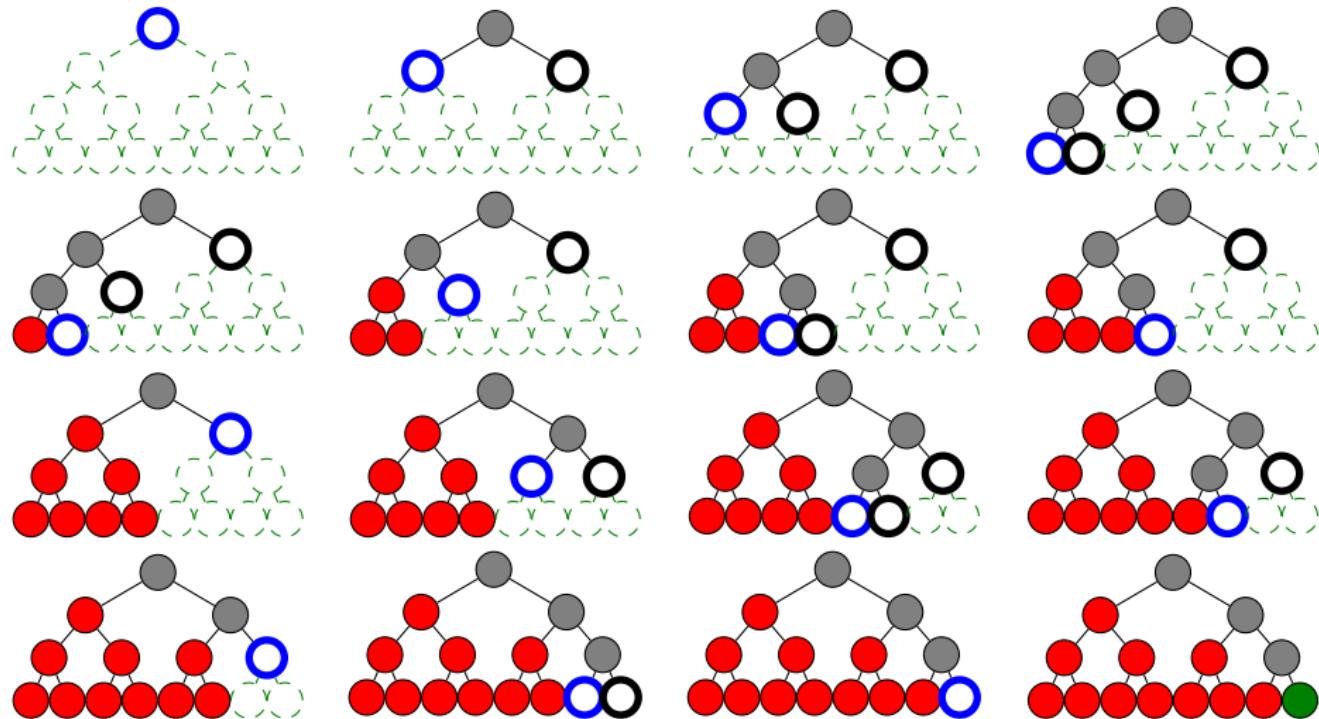
Iterative Deepening Search at various Limit Depths



Iterative Deepening Search at various Limit Depths



Iterative Deepening Search at various Limit Depths



Iterative deepening search: Properties

Complete	Yes
Time	$((d+1) \cdot b^0 + d \cdot b^1 + (d-1) \cdot b^2 + \dots + b^d) \in \mathcal{O}(b^{(d+1)})$
Space	$\mathcal{O}(b \cdot d)$
Optimal	Yes (if step cost = 1)

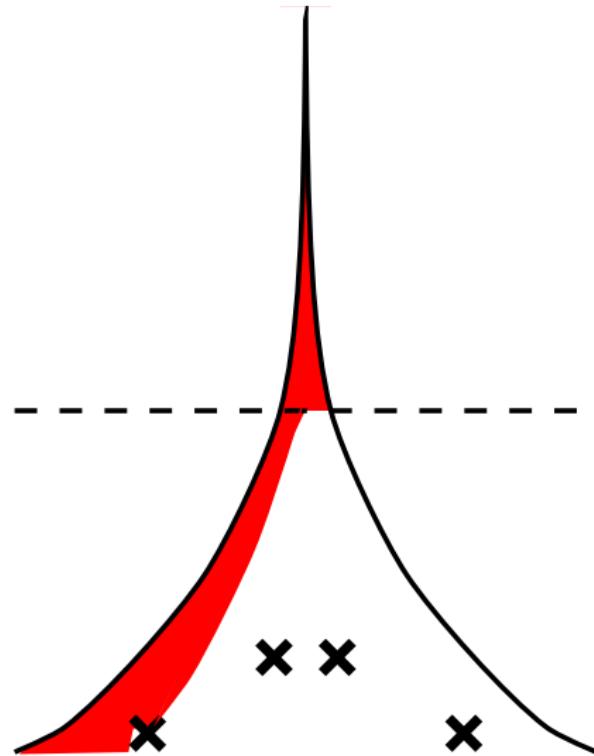
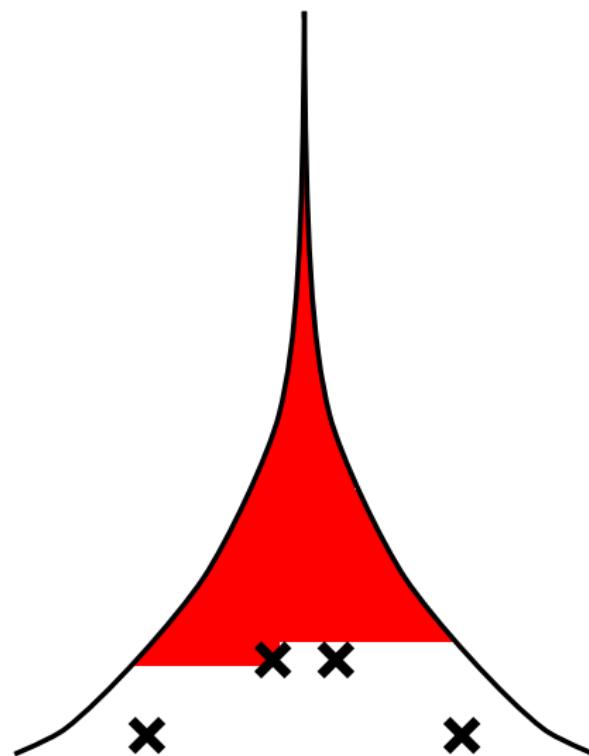
- ▶ **Consequence:** IDS used in practice for search spaces of large, infinite, or unknown depth.

Comparison BFS (optimal) and IDS (not)

- ▶ **Example 4.11.** IDS may fail to be optimal at step sizes > 1.

Breadth-first search

Iterative deepening search



4.3 Further Topics

- ▶ We have only covered tree search algorithms.
- ▶ Nobody uses these in practice ↵ states duplicated in nodes are a huge problem for efficiency.
- ▶ **Definition 4.12.** A graph search algorithm is a variant of a tree search algorithm that prunes nodes whose state has already been considered (duplicate pruning), essentially using a DAG data structure.
- ▶ **Observation 4.13.** 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.
- ▶ **Definition 4.14.** We speak of a search algorithm, when we do not want to distinguish whether it is a tree or graph search algorithm. (difference considered in implementation detail)

Uninformed Search Summary

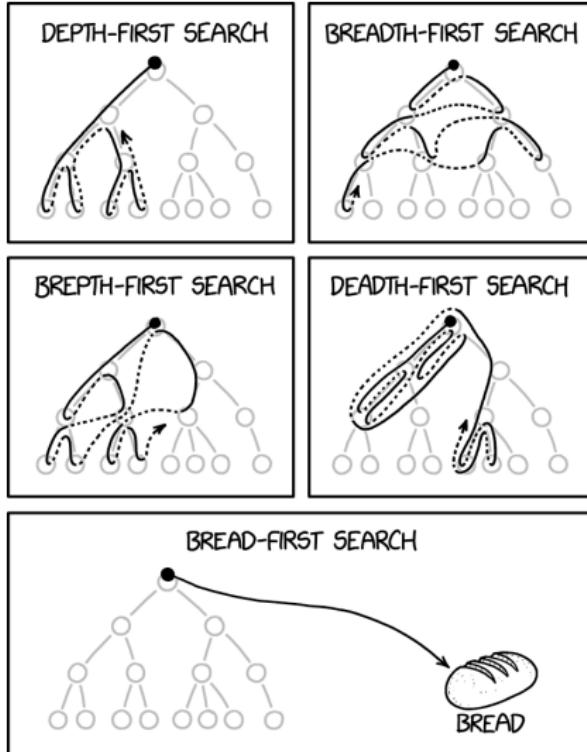
- ▶ **Tree/Graph Search Algorithms:** Systematically explore the state tree/graph induced by a **search problem** in search of a goal state. Search strategies only differ by the treatment of the fringe.
- ▶ **Search Strategies and their Properties:** We have discussed

Criterion	Breadth-first	Uniform-cost	Depth-first	Iterative deepening
Complete?	Yes ¹	Yes ²	No	Yes
Time	b^d	$\approx b^d$	b^m	$b^{(d+1)}$
Space	b^d	$\approx b^d$	bm	bd
Optimal?	Yes*	Yes	No	Yes*
Conditions	¹ b finite	² $0 < \epsilon \leq \text{cost}$		

Search Strategies; the XKCD Take

► More Search Strategies?:

(from <https://xkcd.com/2407/>)



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.

5.1 Greedy Search

- ▶ Idea: Order the **fringe** by estimated “desirability” (Expand most desirable unexpanded node)
- ▶ **Definition 5.1.** An **evaluation function** assigns a **desirability** value to each node of the **search tree**.
- ▶ **Note:** A **evaluation function** is not part of the **search problem**, but must be added externally.
- ▶ **Definition 5.2.** In **best first search**, the **fringe** is a **queue** sorted in decreasing order of **desirability**.
- ▶ Special cases: Greedy search, A^* -search

- ▶ **Idea:** Expand the node that *appears* to be closest to goal.
- ▶ **Definition 5.3.** A **heuristic** is an **evaluation function** h on **states** that estimates of cost from n to the nearest goal state.
- ▶ **Note:** All **nodes** for the same state **states**, must have the same h -value!
- ▶ **Definition 5.4.** Given a **heuristic** h , **greedy search** is the **strategy** where the **fringe** is organized as a **queue** sorted by decreasing h -value.
- ▶ **Example 5.5.** Straight-line distance from/to Bucharest.
- ▶ **Note:** Unlike **uniform-cost search** the node evaluation function has nothing to do with the nodes explored so far

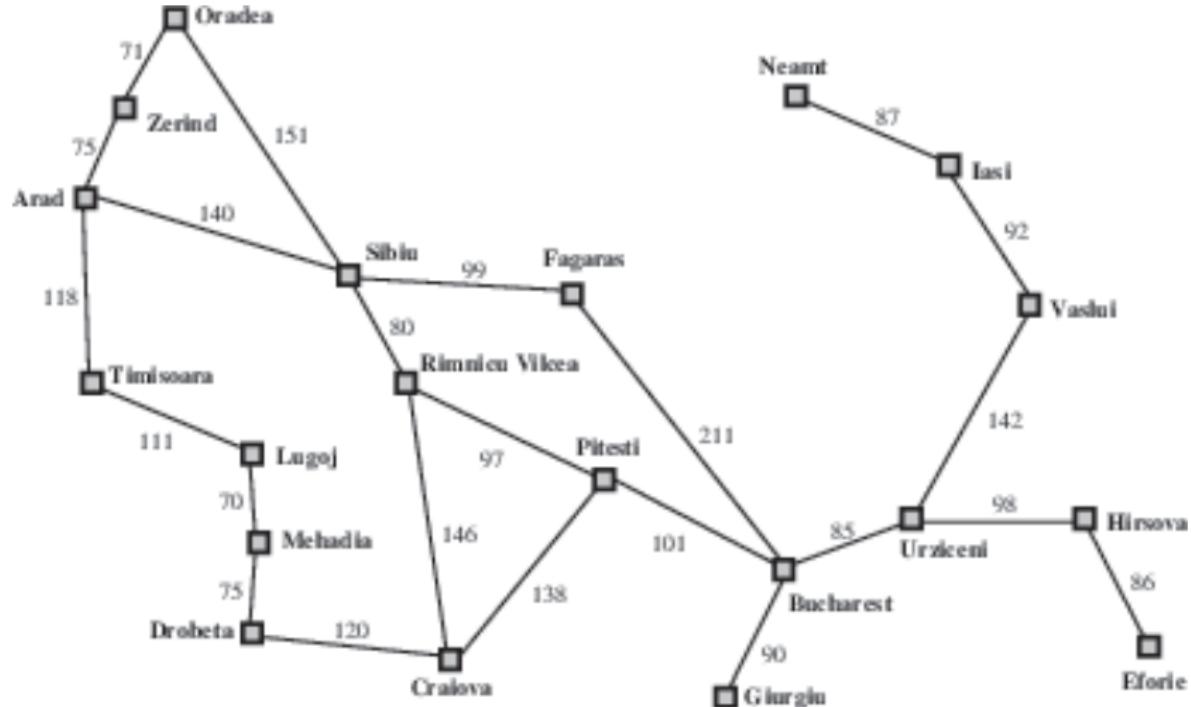
internal search control \leadsto external search control
partial solution cost \leadsto goal cost estimation

- ▶ **Example 5.6 (Informed Travel In Romania).**
 $h_{SLD}(n)$ = straight-line distance from n to Bucharest

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

Romania with Straight-Line Distances

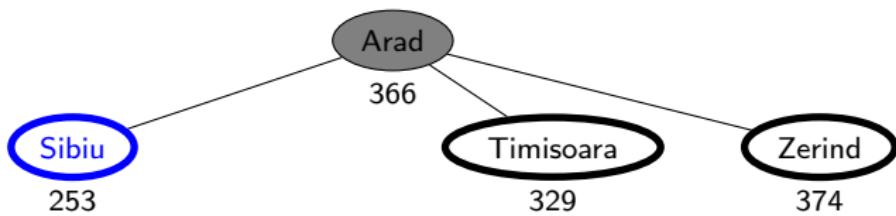
- ▶ Example 5.6 (Informed Travel In Romania).
 $h_{SLD}(n)$ = straight-line distance from n to Bucharest



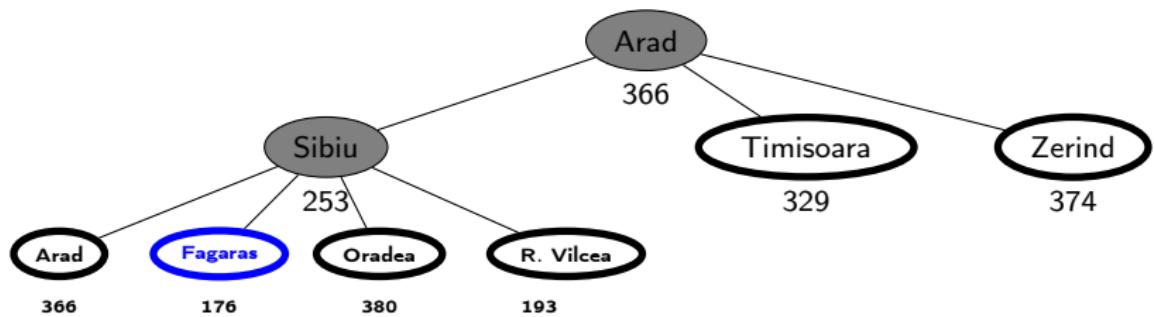
Greedy Search: Romania

Arad
366

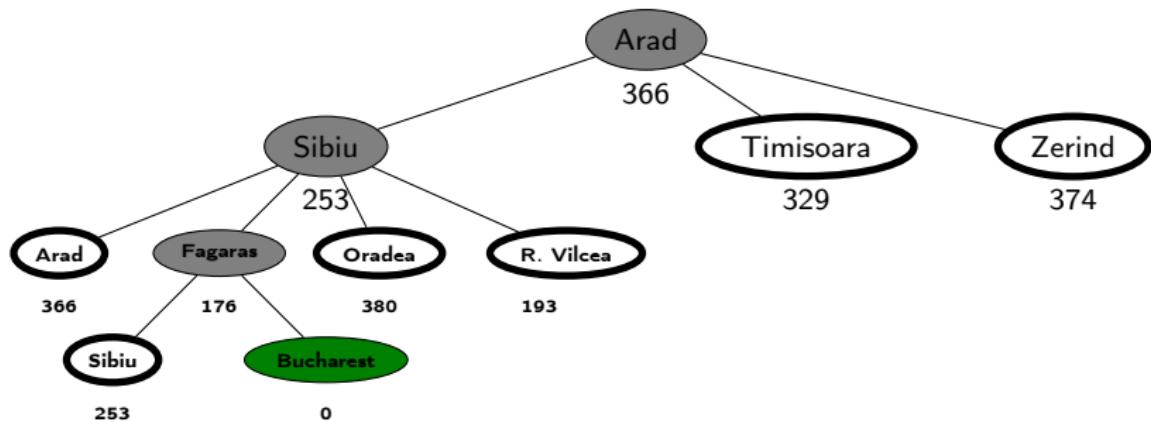
Greedy Search: Romania



Greedy Search: Romania



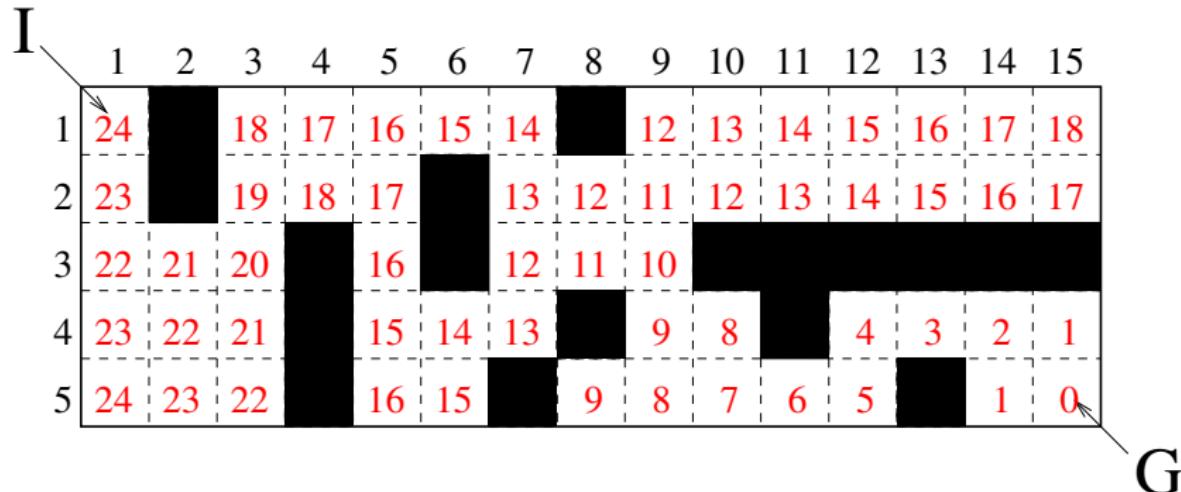
Greedy Search: Romania



Heuristic Functions in Path Planning

► Example 5.7 (The maze solved).

We indicate h^* by giving the goal distance



► Example 5.8 (Maze Heuristic: the good case).

We use the [Manhattan distance](#) to the goal as a heuristic

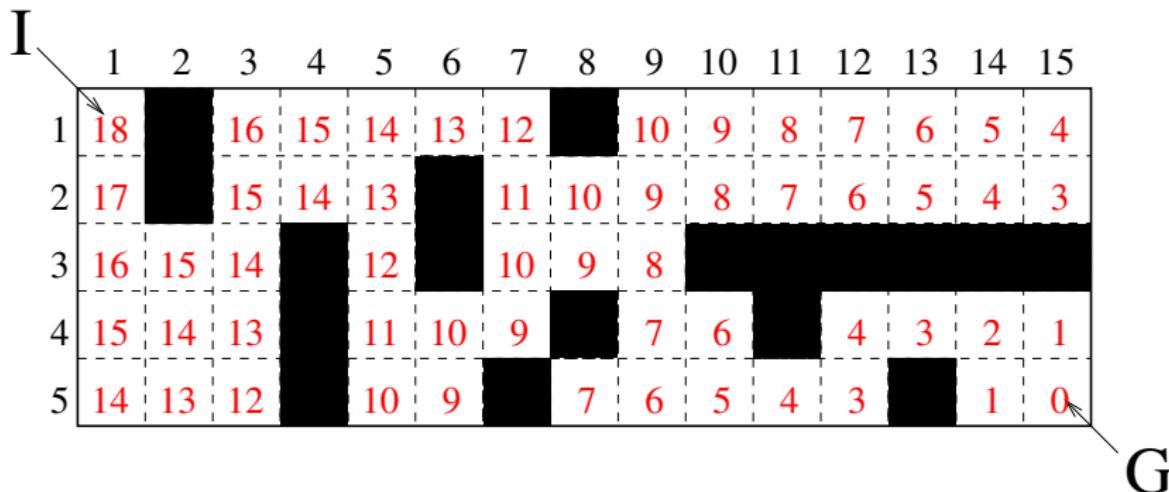
Heuristic Functions in Path Planning

- ▶ **Example 5.7 (The maze solved).**

We indicate h^* by giving the goal distance

- ▶ **Example 5.8 (Maze Heuristic: the good case).**

We use the [Manhattan distance](#) to the goal as a heuristic



Heuristic Functions in Path Planning

- ▶ **Example 5.7 (The maze solved).**

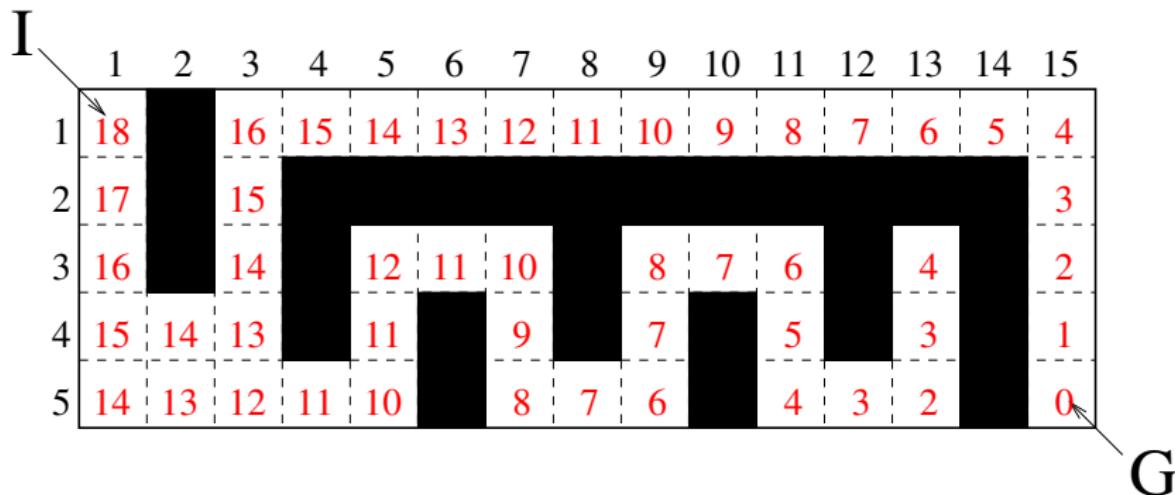
We indicate h^* by giving the goal distance

- ▶ **Example 5.8 (Maze Heuristic: the good case).**

We use the [Manhattan distance](#) to the goal as a heuristic

- ▶ **Example 5.9 (Maze Heuristic: the bad case).**

We use the [Manhattan distance](#) to the goal as a heuristic again



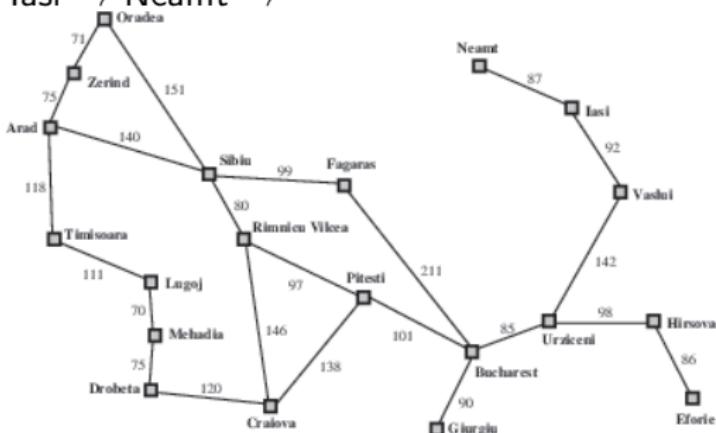
Greedy search: Properties

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

Greedy search: Properties

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

► **Example 5.10.** Greedy search can get stuck going from Iasi to Oradea:
Iasi → Neamt → Iasi → Neamt → ...



Greedy search: Properties

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

- **Example 5.10.** 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](#).
- **But:** A good heuristic can give dramatic improvements.

5.2 Heuristics and their Properties

- ▶ **Definition 5.11.** Let Π be a problem with states S . A **heuristic function** (or short **heuristic**) for Π is a function $h: S \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ so that $h(s)=0$ whenever s is a **goal state**.
- ▶ $h(s)$ is intended as an estimate between state s and the nearest **goal state**.
- ▶ **Definition 5.12.** Let Π be a problem with states S , then the function $h^*: S \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$, 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 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 $\epsilon\nu\rhoισκειν$ ($\hat{=}$ “I find”) (aka. $\epsilon\nu\rho\epsilonκα!$)
- ▶ Popularized in modern science by George Polya: “How to solve it” [Pól73]
- ▶ same word often used for “rule of thumb” or “imprecise solution method”.

- ▶ “Distance Estimate”? (h is an arbitrary function in principle)
 - ▶ In practice, we want it to be **accurate** (aka: **informative**), i.e., close to the actual goal distance.
 - ▶ We also want it to be fast, i.e., a small **overhead** for computing h .
 - ▶ These two wishes are in contradiction!
- ▶ **Example 5.13 (Extreme cases).**
 - ▶ $h = 0$: no overhead at all, completely un-informative.
 - ▶ $h = h^*$: perfectly accurate, overhead $\hat{=}$ solving the problem in the first place.
- ▶ **Observation 5.14.** We need to trade off the accuracy of h against the overhead for computing it.

- ▶ **Definition 5.15.** Let Π be a problem with states S and actions A . We say that a heuristic function 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 $s \in S$ and $a \in A$.
- ▶ 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.

- Assertion (Consistency implies Admissibility) Let Π be a problem, and let h be a heuristic function 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 state.

1.

1.1. base case: $h(s) = 0$ by definition of heuristic functions, so $h(s) \leq h^*(s)$ as desired. \square

1.2. step case:

1.2.1. We assume that $h(s') \leq h^*(s)$ for all states s' with a cheapest goal path of length n .

1.2.2. Let s be a state whose cheapest goal path has length $n+1$ and the first transition is $o = (s, s')$.

1.2.3. By consistency, we have $h(s) - h(s') \leq c(o)$ and thus $h(s) \leq h(s') + c(o)$.

1.2.4. By construction, $h^*(s)$ has a cheapest goal path of length n and thus, by induction hypothesis $h(s') \leq h^*(s')$.

1.2.5. By construction, $h^*(s) = h^*(s') + c(o)$.

1.2.6. Together this gives us $h(s) \leq h^*(s)$ as desired. \square

- Consistency is a sufficient condition for admissibility

(easier to check)

Properties of Heuristic Functions: Examples

- ▶ **Example 5.16.** Straight line distance is admissible and consistent by the triangle inequality.
If you drive 100km, then the straight line distance to Rome can't decrease by more than 100km.
- ▶ **Observation:** In practice, **admissible heuristics** are typically **consistent**.
- ▶ **Example 5.17 (An admissible, but inconsistent heuristic).**
In the problem of traveling to Rome, let $h(\text{Munich}) = 300$ and $h(\text{Innsbruck}) = 100$.
- ▶ **Inadmissible heuristics:** typically arise as approximations of admissible heuristics that are too costly to compute. (see later)

5.3 A-Star Search

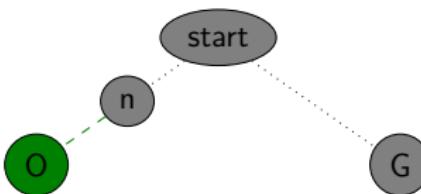
- ▶ **Idea:** Avoid expanding paths that are already expensive (make use of actual cost)
The simplest way to combine heuristic and path cost is to simply add them.
- ▶ **Definition 5.18.** The **evaluation function** for A*-search is given by $f(n) = g(n) + h(n)$, where $g(n)$ is the path cost for n and $h(n)$ is the estimated cost to goal from n .
- ▶ Thus $f(n)$ is the estimated total cost of path through n to goal
- ▶ **Definition 5.19.** Best-First-Search with evaluation function $g+h$ is called **A* search**.

A* Search: Optimality

► **Theorem 5.20.** A* search with admissible heuristic is optimal.

► **Proof:** We show that sub-optimal nodes are never selected by A*

1. Suppose a suboptimal goal G has been generated then we are in the following situation:



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

$$f(G) = g(G) \quad \text{since } h(G) = 0$$

$$g(G) > g(O) \quad \text{since } G \text{ suboptimal}$$

$$g(O) = g(n) + h^*(n) \quad n \text{ on optimal path}$$

$$g(n) + h^*(n) \geq g(n) + h(n) \quad \text{since } h \text{ is admissible}$$

$$g(n) + h(n) = f(n)$$

3. Thus, $f(G) > f(n)$ and A* never selects G for expansion.

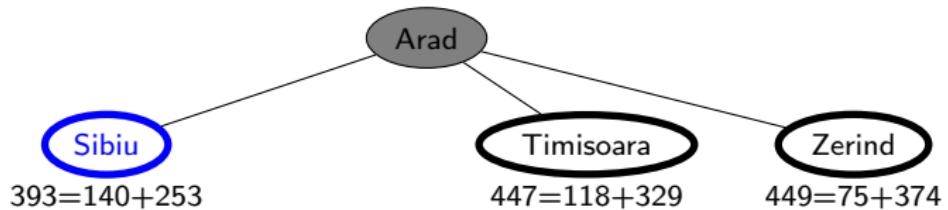
□

A* Search Example

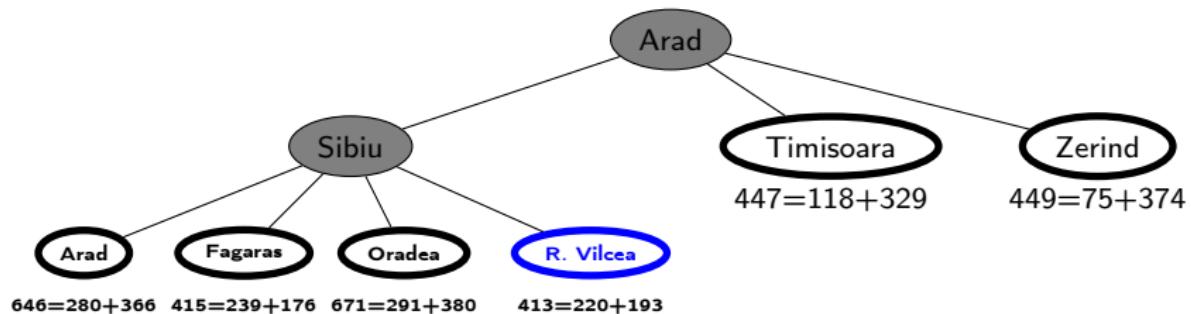
Arad

$366 = 0 + 366$

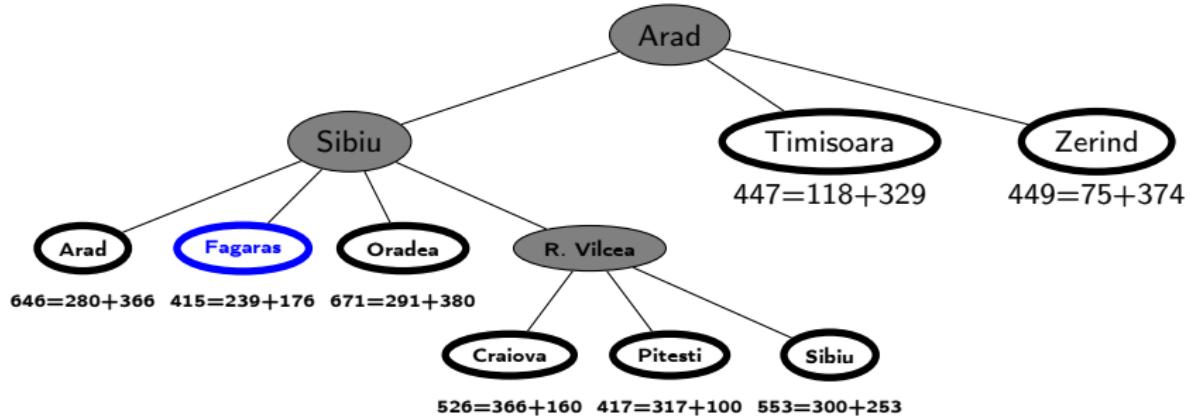
A* Search Example



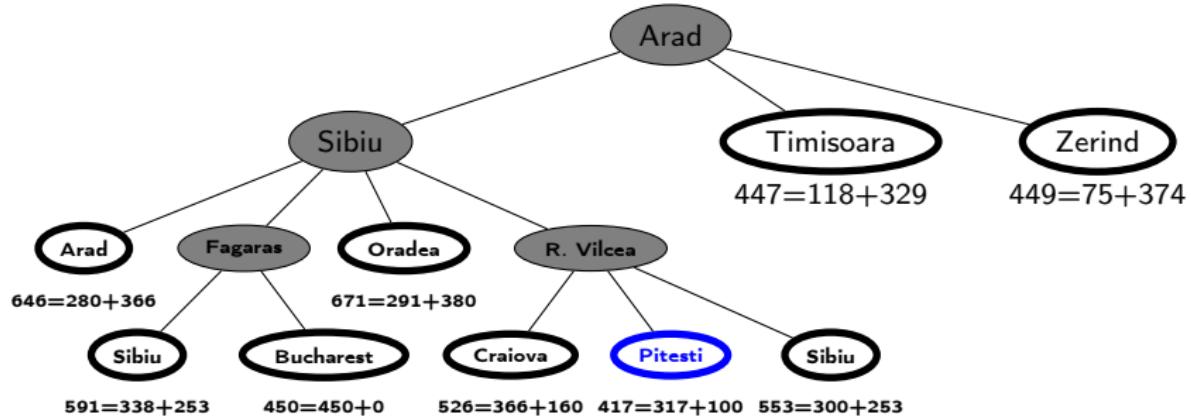
A* Search Example



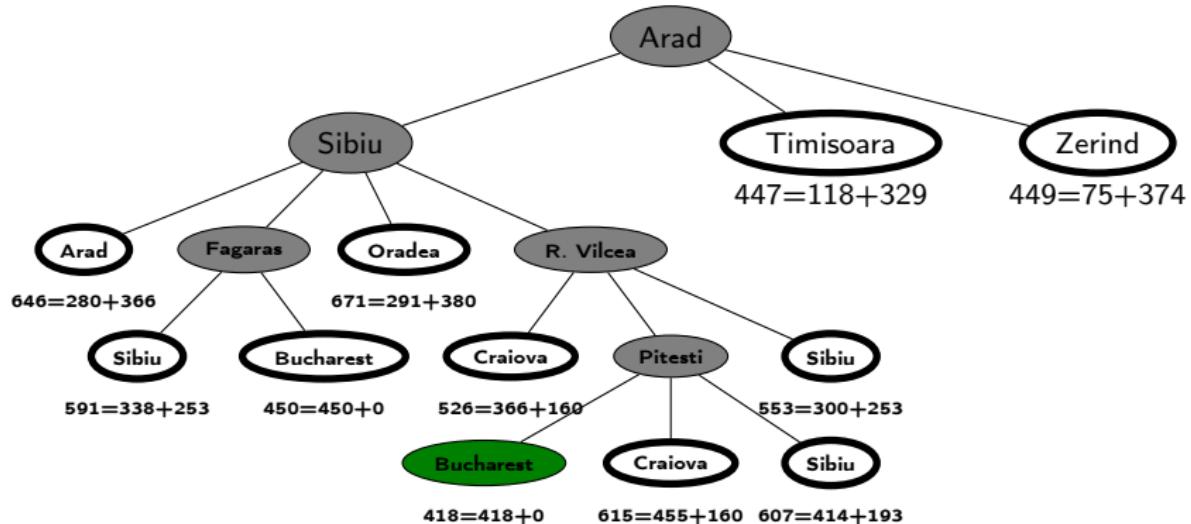
A* Search Example



A* Search Example

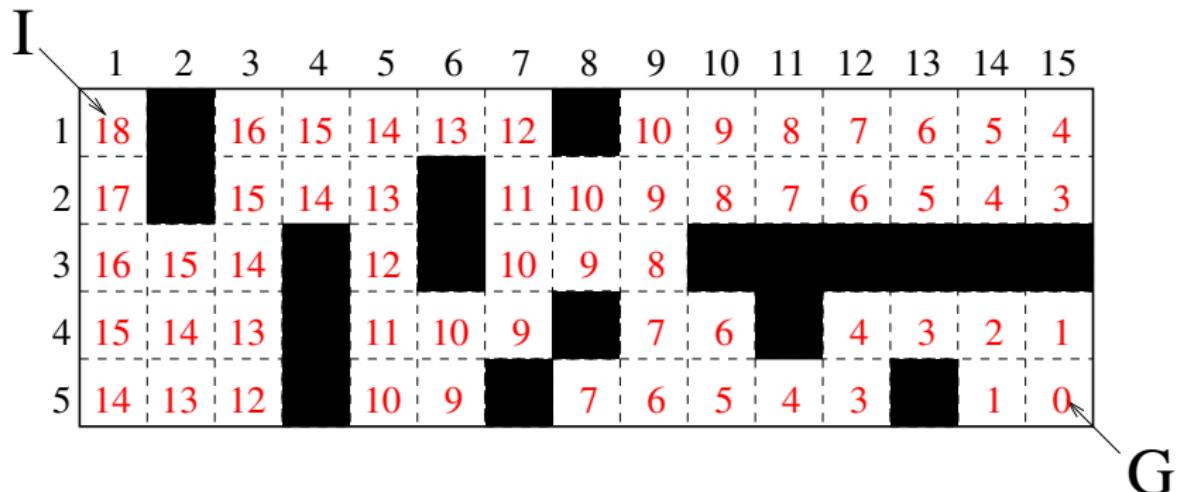


A* Search Example



Additional Observations (Not Limited to Path Planning)

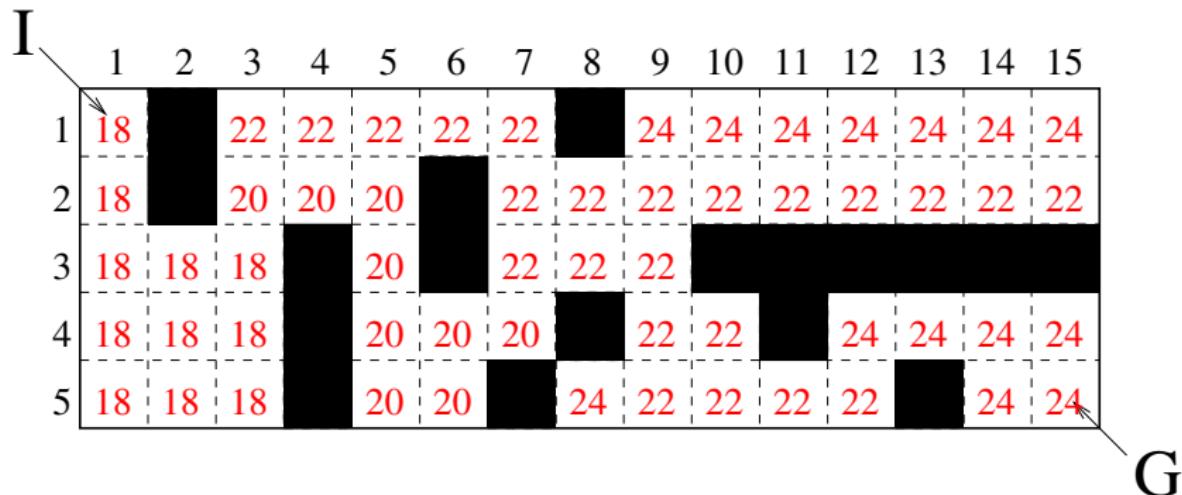
- ▶ Example 5.21 (Greedy best-first search, “good case”).



We will find a solution with little search.

Additional Observations (Not Limited to Path Planning)

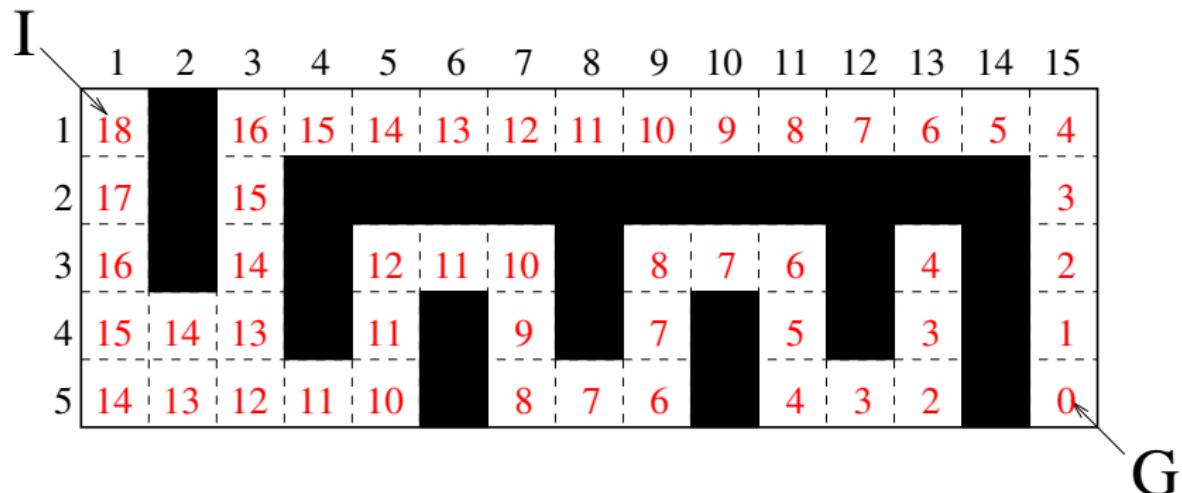
- ▶ Example 5.22 (A^* ($g+h$), “good case”).



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

Additional Observations (Not Limited to Path Planning)

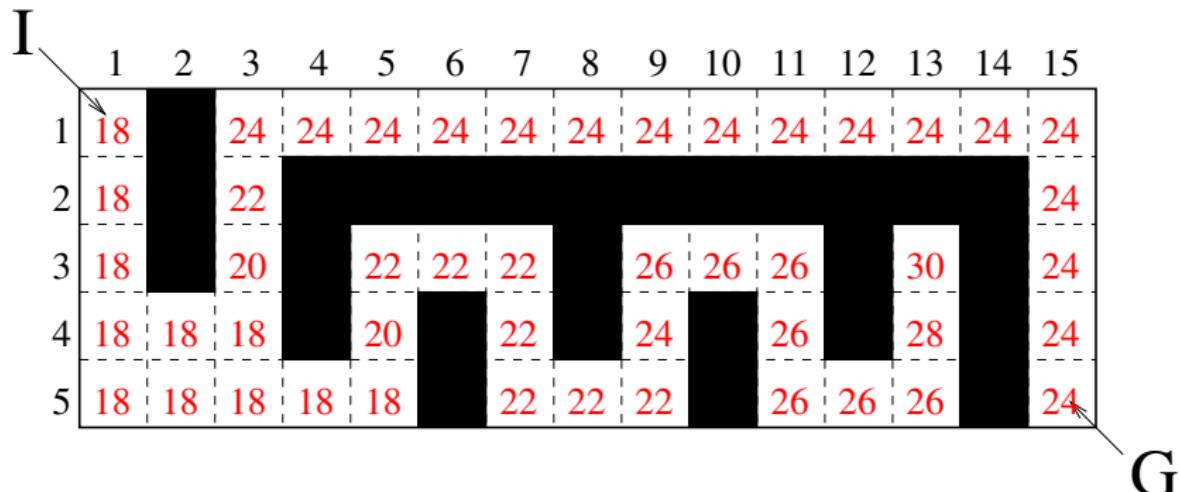
- ▶ Example 5.23 (Greedy best-first search, “bad case”).



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

Additional Observations (Not Limited to Path Planning)

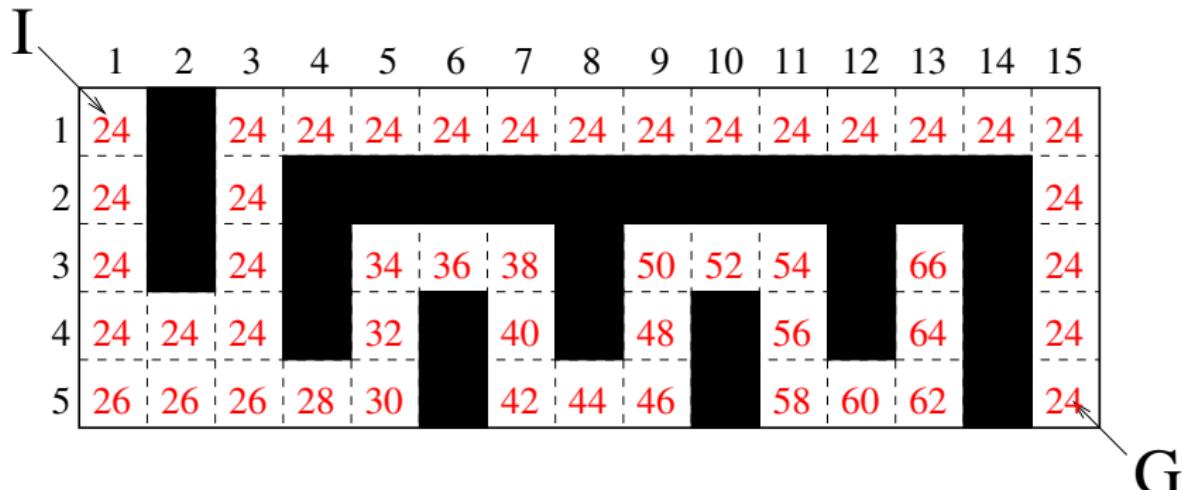
- ▶ Example 5.24 (A^* ($g+h$), “bad case”).



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

Additional Observations (Not Limited to Path Planning)

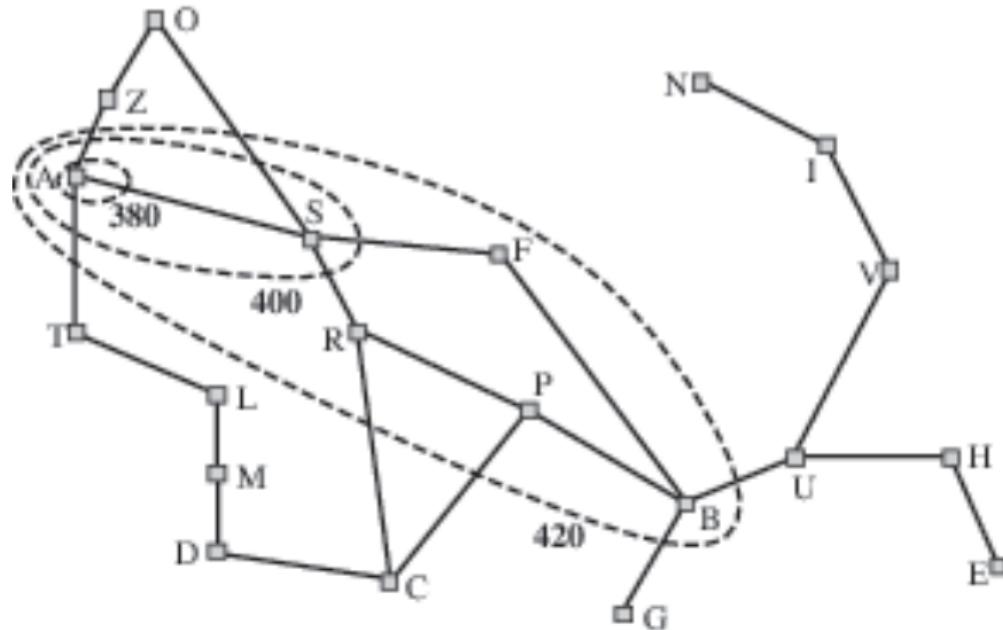
- ▶ Example 5.25 (A^* ($g+h$) using h^*).



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

A^* search: f -contours

- A^* gradually adds “ f -contours” of nodes



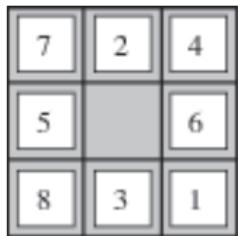
A^* search: Properties

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

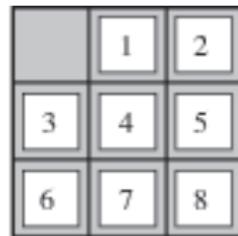
- ▶ A^* expands all (some/no) nodes with $f(n) < h^*(n)$
- ▶ The run-time depends on how good we approximated the real cost h^* with h .

5.4 Finding Good Heuristics

Admissible heuristics: Example 8-puzzle



Start State



Goal State

- ▶ **Example 5.26.** Let $h_1(n)$ be the number of misplaced tiles in node n ($h_1(S) = 6$)
- ▶ **Example 5.27.** Let $h_2(n)$ be the total Manhattan distance from desired location of each tile. ($h_2(S) = 2+0+3+1+0+1+3+4 = 14$)
- ▶ **Observation 5.28 (Typical search costs).** (IDS \cong iterative deepening search)

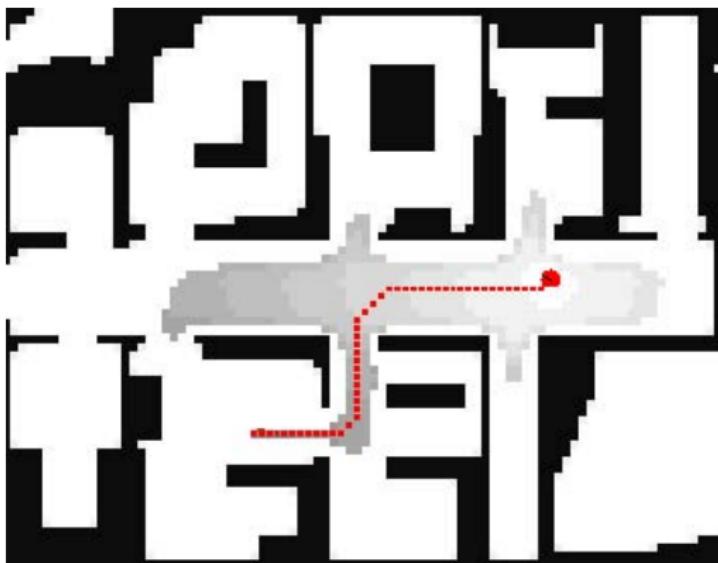
nodes explored	IDS	$A^*(h_1)$	$A^*(h_2)$
$d = 14$	3,473,941	539	113
$d = 24$	too many	39,135	1,641

- ▶ **Definition 5.29.** Let h_1 and h_2 be two admissible heuristics we say that h_2 **dominates** h_1 if $h_2(n) \geq h_1(n)$ for all n .
- ▶ **Theorem 5.30.** If h_2 dominates h_1 , then h_2 is better for search than h_1 .

- ▶ Finding good admissible heuristics is an art!
- ▶ Idea: Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.
- ▶ **Example 5.31.** If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then we get heuristic h_1 .
- ▶ **Example 5.32.** If the rules are relaxed so that a tile can move to any adjacent square, then we get heuristic h_2 . (Manhattan distance)
- ▶ **Definition 5.33.** Let $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ be a search problem, then we call a search problem $\mathcal{P}' := \langle \mathcal{S}, \mathcal{A}', \mathcal{T}', \mathcal{I}', \mathcal{G}' \rangle$ a **relaxed problem** (wrt. \mathcal{P} ; or simply **relaxation** of \mathcal{P}), iff $\mathcal{A} \subseteq \mathcal{A}'$, $\mathcal{T} \subseteq \mathcal{T}'$, $\mathcal{I} \subseteq \mathcal{I}'$, and $\mathcal{G} \subseteq \mathcal{G}'$.
- ▶ **Lemma 5.34.** If \mathcal{P}' relaxes \mathcal{P} , then every solution for \mathcal{P} is one for \mathcal{P}' .
- ▶ **Key point:** The optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem

Empirical Performance: A^* in Path Planning

- ▶ Example 5.35 (Live Demo vs. Breadth-First Search).



See <http://qiao.github.io/PathFinding.js/visual/>

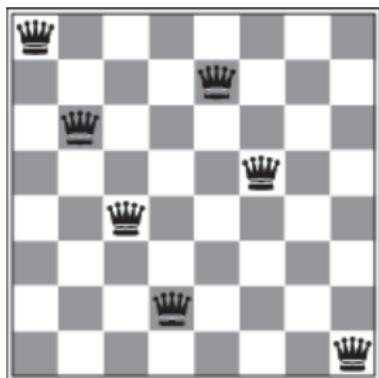
- ▶ Difference to Breadth-first Search?: That would explore all grid cells in a *circle* around the initial state!

6 Local Search

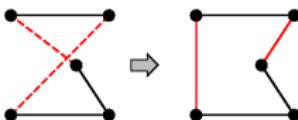
Systematic Search vs. Local Search

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

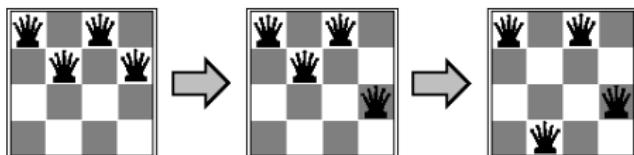
- ▶ **Idea:** Sometimes the path to the solution is irrelevant.
- ▶ **Example 6.5 (8 Queens Problem).** Place 8 queens on a chess board, so that no two queens threaten each other.
- ▶ This problem has various solutions (the one on the right isn't one of them)
- ▶ **Definition 6.6.** A **local search** algorithm is a search algorithm that operates on a single state, the **current state** (rather than multiple paths).
(advantage: constant space)
- ▶ Typically local search algorithms only move to **successor** of the current state, and do not retain search paths.
- ▶ Applications include: integrated circuit design, factory-floor layout, job-shop scheduling, portfolio management, fleet deployment, . . .



- ▶ **Definition 6.7 (Traveling Salesman Problem).** Find shortest trip through set of cities such that each city is visited exactly once.
- ▶ **Idea:** Start with any complete tour, perform pairwise exchanges



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



Hill-climbing (gradient ascent/descent)

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

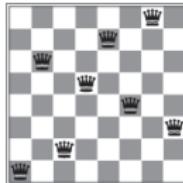
```
procedure Hill-Climbing (problem) /* a state that is a local minimum */
    local current, neighbor /* nodes */
    current := Make-Node(Initial-State[problem])
    loop
        neighbor := <a highest-valued successor of current>
        if Value[neighbor] < Value[current] return [current] end if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

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

Example Hill-Climbing with 8 Queens

- ▶ **Idea:** Consider $h \triangleq$ number of queens that threaten each other.
- ▶ **Example 6.10.** An 8-queens state with heuristic cost estimate $h = 17$ showing h -values for moving a queen within its column
- ▶ **Problem:** The state space has local minima. e.g. the board on the right has $h = 1$ but every **successor** has $h > 1$.

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



Hill-climbing

- ▶ **Problem:** Depending on initial state, can get stuck on local maxima/minima and plateaux.

- ▶ “Hill-climbing search is like climbing Everest in thick fog with amnesia”.

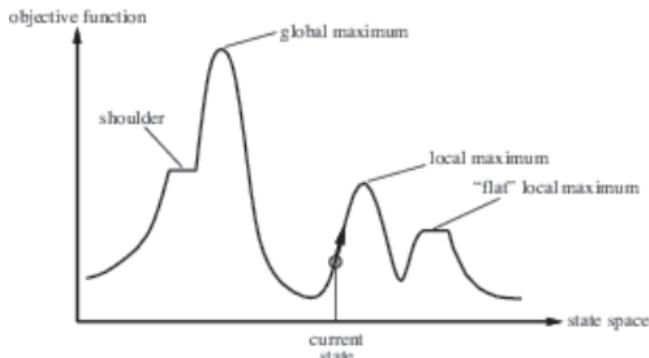
- ▶ **Idea:** Escape local maxima by allowing some “bad” or random moves.

- ▶ **Example 6.11.** local search, simulated annealing...

- ▶ **Properties:** All are incomplete, non-optimal.

- ▶ Sometimes performs well in practice

(if (optimal) solutions are dense)



Simulated annealing (Idea)

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



- ▶ Annealing is the process of heating steel and let it cool gradually to give it time to grow an optimal crystal structure.
- ▶ Simulated Annealing is like shaking a ping-pong ball occasionally on a bumpy surface to free it. (so it does not get stuck)
- ▶ Devised by Metropolis et al., 1953, for physical process modelling
- ▶ Widely used in VLSI layout, airline scheduling, etc.

Simulated annealing (Implementation)

► The algorithm

```
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  $\infty$ 
        T := schedule[t]
        if T = 0 return current end if
        next := <a randomly selected successor of current>
         $\Delta(E) := \text{Value}[next] - \text{Value}[current]$ 
        if  $\Delta(E) > 0$  current := next
        else
            current := next <only with probability>  $e^{\Delta(E)/T}$ 
        end if
    end for
end procedure
```

- a problem 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 \implies 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 .

- Is this necessarily an interesting guarantee?

- ▶ **Idea:** Keep k states instead of 1; choose top k of all their successor.
- ▶ 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 states end up on same local hill
- ▶ **Idea:** Choose k successor randomly, biased towards good ones.(Observe the close analogy to natural selection!)

Genetic algorithms (very briefly)

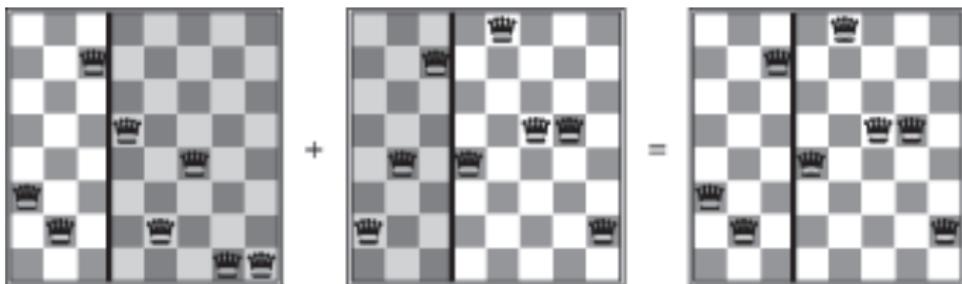
- Idea: Use local beam search (keep a population of k) randomly modify population (mutation) generate successors from pairs of states (sexual reproduction) optimize a fitness function (survival of the fittest)



► Example 6.13.

Genetic algorithms (continued)

- ▶ **Problem:** Genetic Algorithms require states encoded as strings (GPs use programs)
- ▶ Crossover helps iff substrings are meaningful components
- ▶ **Example 6.14 (Evolving 8 Queens).**



- ▶ GAs \neq evolution: e.g., real genes encode replication machinery!

Chapter 8 Adversarial Search for Game Playing

1 Introduction



“Adversarial search” = 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.

► **Restrictions:**

- Game states discrete, number of game states 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 steps).

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.

- ▶ **Definition 1.1 (Game State Space).** A **game state space** is a 6-tuple $\Theta = \langle S, A, T, I, S^T, u \rangle$ where:
 - ▶ **states** S , **actions** A , deterministic **transition relation** T , **initial state** I . As in classical **search problems**, except:
 - ▶ S is the disjoint union of S^{Max} , S^{Min} , and S^T .
 - ▶ A is the disjoint union of A^{Max} and A^{Min} .
 - ▶ For $a \in A^{\text{Max}}$, if $s \xrightarrow{a} s'$ then $s \in S^{\text{Max}}$ and $s' \in (S^{\text{Min}} \cup S^T)$.
 - ▶ For $a \in A^{\text{Min}}$, if $s \xrightarrow{a} s'$ then $s \in S^{\text{Min}}$ and $s' \in (S^{\text{Max}} \cup S^T)$.
 - ▶ S^T is the set of **terminal states**.
 - ▶ $u: S^T \rightarrow \mathbb{R}$ is the **utility function**.
- ▶ **Definition 1.2 (Commonly used terminology).** **position** $\hat{=}$ **state**, **end state** $\hat{=}$ **terminal state**, **move** $\hat{=}$ **action**.
- ▶ 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* do that here.

Why Games are Hard to Solve: I

- ▶ What is a “solution” here?
- ▶ **Definition 1.3.** Let Θ be a game state space, and let $X \in \{\text{Max}, \text{Min}\}$. A **strategy** for X is a function $\sigma^X : S^X \rightarrow A^X$ so that a is applicable to s whenever $\sigma^X(s) = a$.
- ▶ We don't know how the opponent will react, and need to prepare for all possibilities.
- ▶ **Definition 1.4.** A strategy is **optimal** if it yields the best possible utility for X assuming perfect opponent play (not formalized here).
- ▶ In (almost) all games, computing a strategy is infeasible. Instead, compute the next move “on demand”, given the current game state.

Why Games are hard to solve II

- ▶ Number of reachable states in Chess: 10^{40} .
- ▶ Number of reachable states in Go: 10^{100} .
- ▶ It's worse even: Our algorithms here look at search **trees** (game trees), no duplicate checking. Chess: $35^{100} \approx 10^{154}$. Go: $200^{300} \approx 10^{690}$.

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” $\hat{=}$ **natural language**.

Specialized vs. General Game Playing

- ▶ And which game descriptions do computers use?
 - ▶ Explicit: Only in illustrations.
 - ▶ Blackbox/API: Assumed description in **(This Chapter)**
 - ▶ Method of choice for all those game players out there in the market (Chess computers, video game opponents, you name it).
 - ▶ Programs designed for, and specialized to, a particular game.
 - ▶ Human knowledge is key: **evaluation functions** (see later), opening databases (Chess!!), end 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

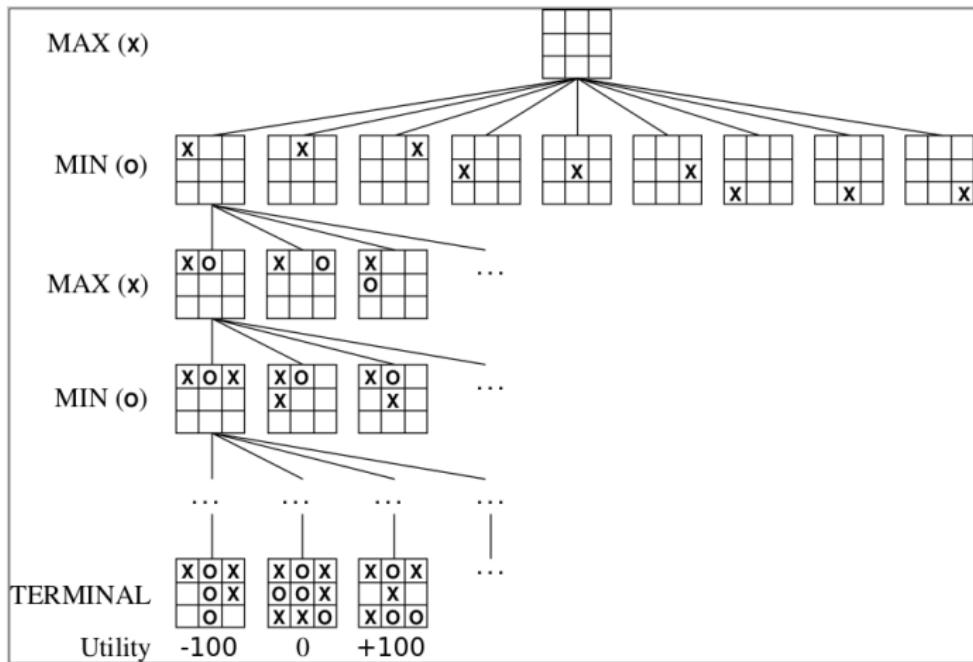
- ▶ **Minimax Search:** How to compute an optimal strategy?
 - ▶ Minimax is the canonical (and easiest to understand) algorithm for *solving* games, i.e., computing an optimal strategy.
- ▶ **Evaluation Functions:** But what if we don't have the time/memory to solve the entire game?
 - ▶ Given limited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

2 Minimax Search

“Minimax”?

- ▶ We want to compute an **optimal strategy** for player “Max”.
 - ▶ In other words: “**We are Max, and our opponent is Min.**”
- ▶ Recall:
 - ▶ We compute the **strategy offline**, before the game begins. During the game, whenever it’s our turn, we just lookup the corresponding action.
 - ▶ **Max** attempts to *maximize* the utility $u(s)$ of the terminal state that will be reached during play.
 - ▶ **Min** attempts to *minimize* $u(s)$.
- ▶ So what?
 - ▶ The computation alternates between minimization and maximization \leadsto hence “minimax”.

Example Tic-Tac-Toe

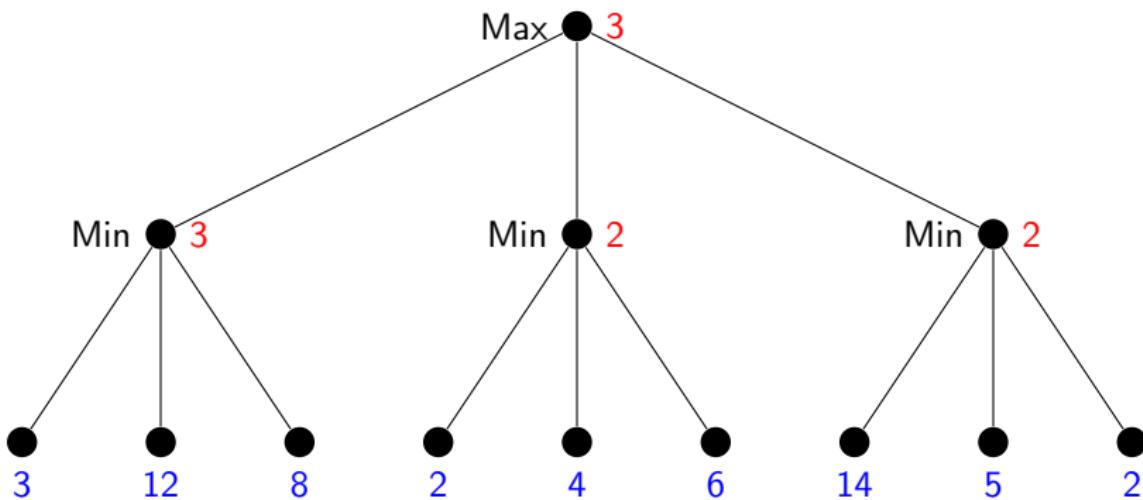


- ▶ Game tree, current player marked on the left.
- ▶ Last row: terminal positions with their utility.

► We max, we min, we max, we min ...

1. Depth-first search in game tree, with Max in the root.
2. Apply utility function to terminal positions.
3. Bottom-up for each inner node n in the tree, compute the utility $u(n)$ of n as follows:
 - If it's Max's turn: Set $u(n)$ to the maximum of the utilities of n 's successor nodes.
 - If it's Min's turn: Set $u(n)$ 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 Minimax.

The Minimax Algorithm: Pseudo-Code

- **Definition 2.1.** The **minimax algorithm** (often just called **minimax**) is given by the following function whose input is a state $s \in S^{\text{Max}}$, in which Max is to move.

function Minimax–Decision(s) **returns** an action

$v := \text{Max–Value}(s)$

return an action yielding value v **in** the previous **function** call

function Max–Value(s) **returns** a utility value

if Terminal–Test(s) **then return** $u(s)$

$v := -\infty$

for each $a \in \text{Actions}(s)$ **do**

$v := \max(v, \text{Min–Value}(\text{ChildState}(s, a)))$

return v

function Min–Value(s) **returns** a utility value

if Terminal–Test(s) **then return** $u(s)$

$v := +\infty$

for each $a \in \text{Actions}(s)$ **do**

$v := \min(v, \text{Max–Value}(\text{ChildState}(s, a)))$

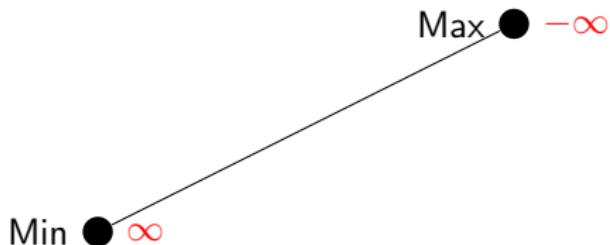
return v

Minimax: Example, Now in Detail

Max ● $-\infty$

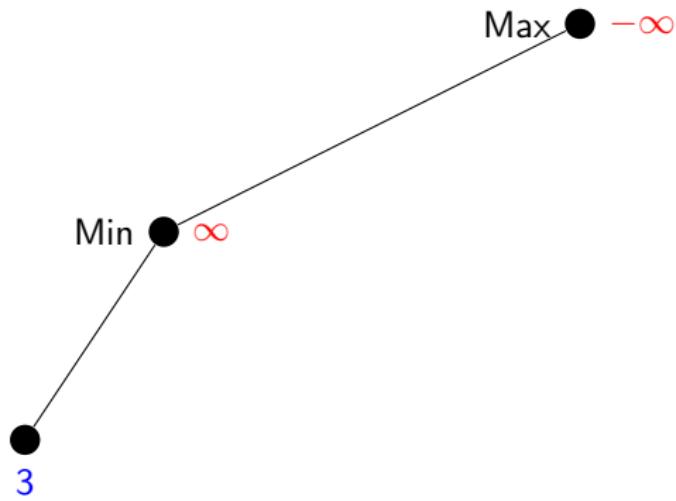
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



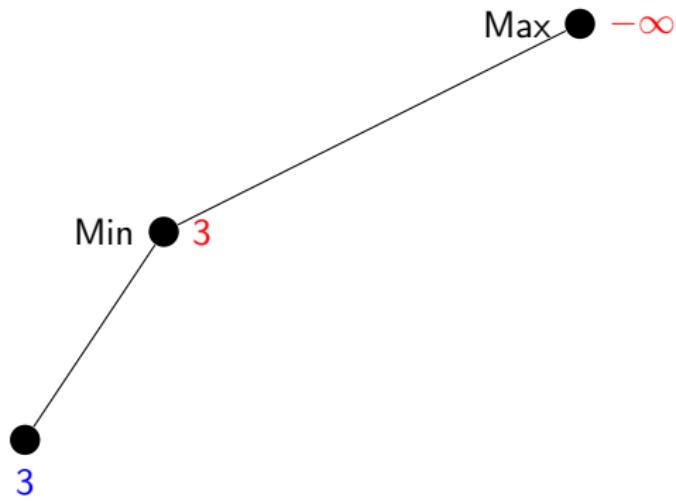
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



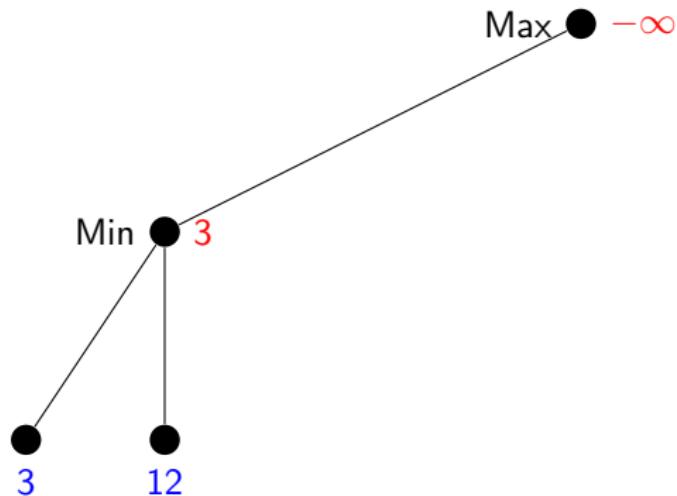
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



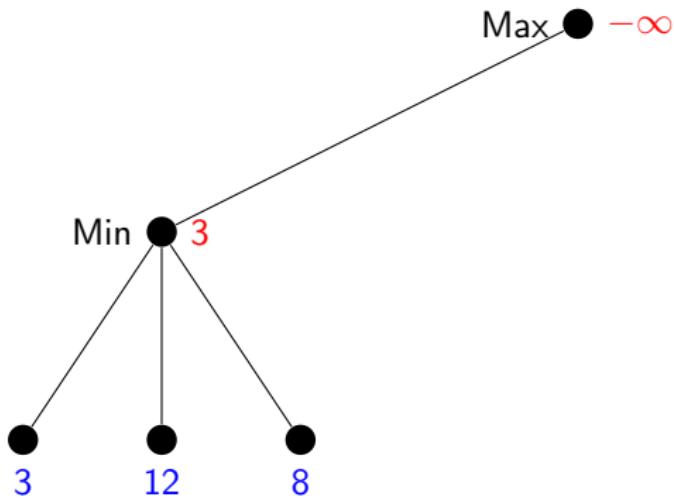
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



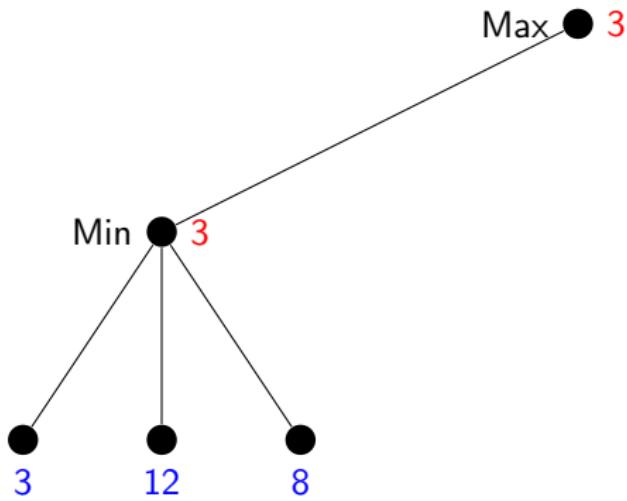
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



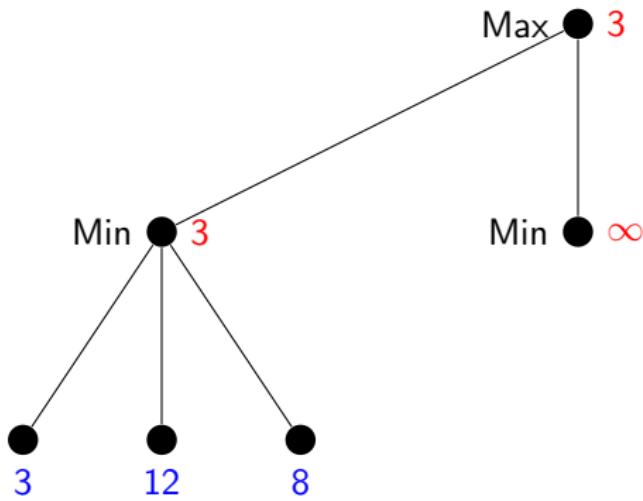
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



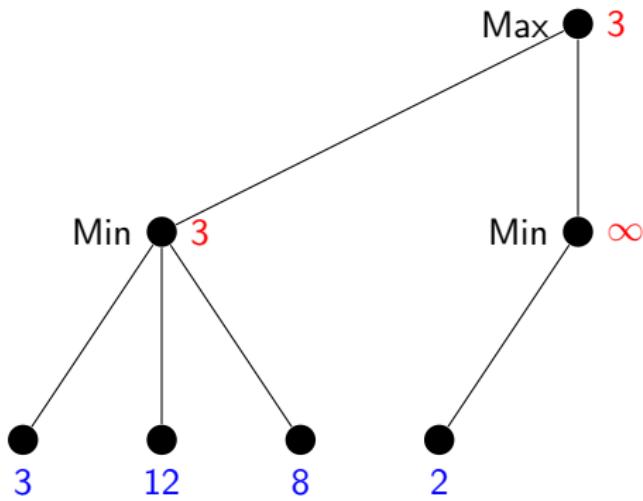
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



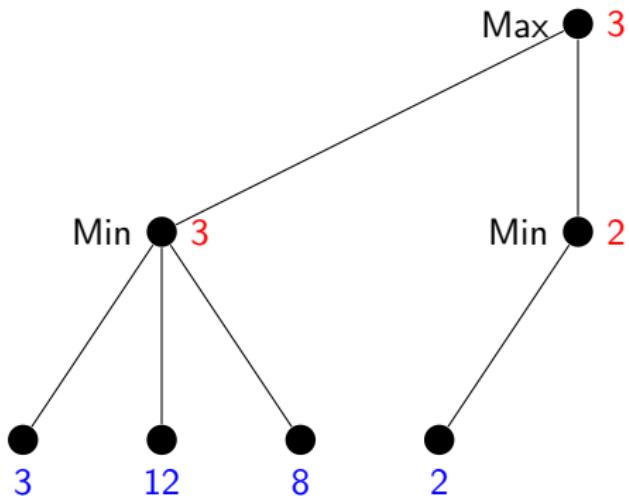
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



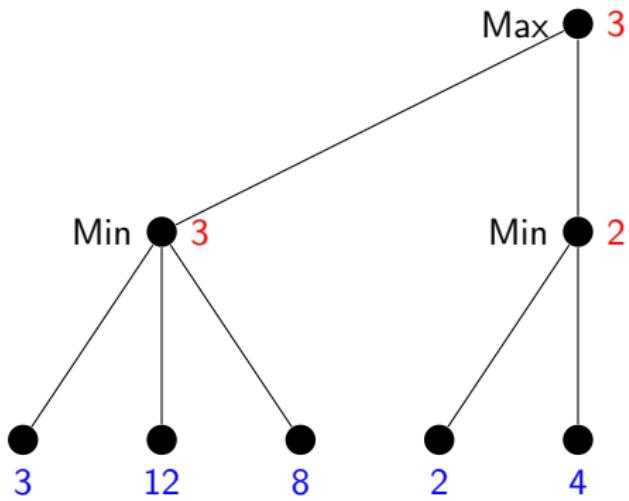
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



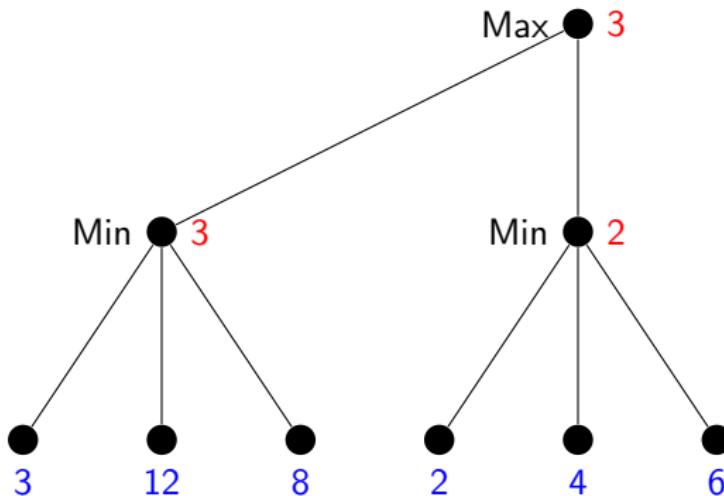
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



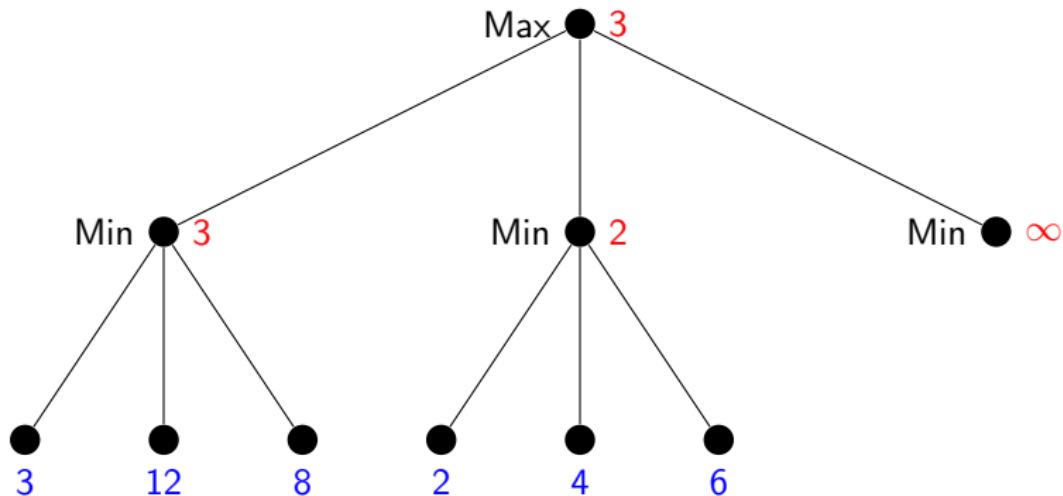
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



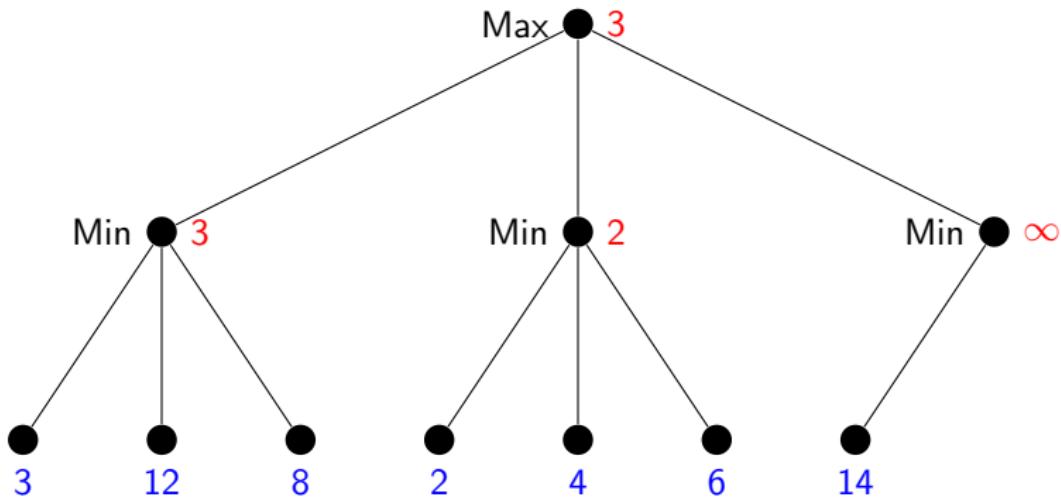
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



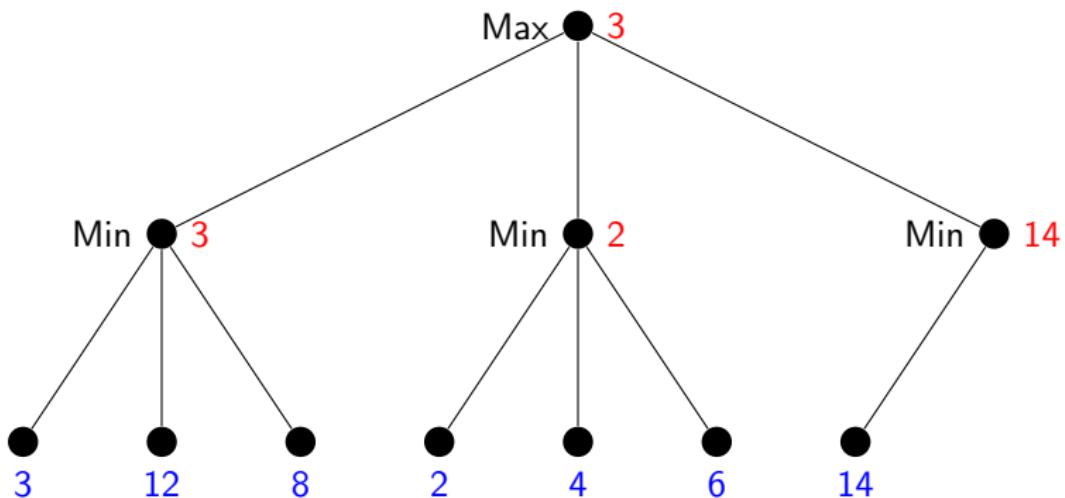
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



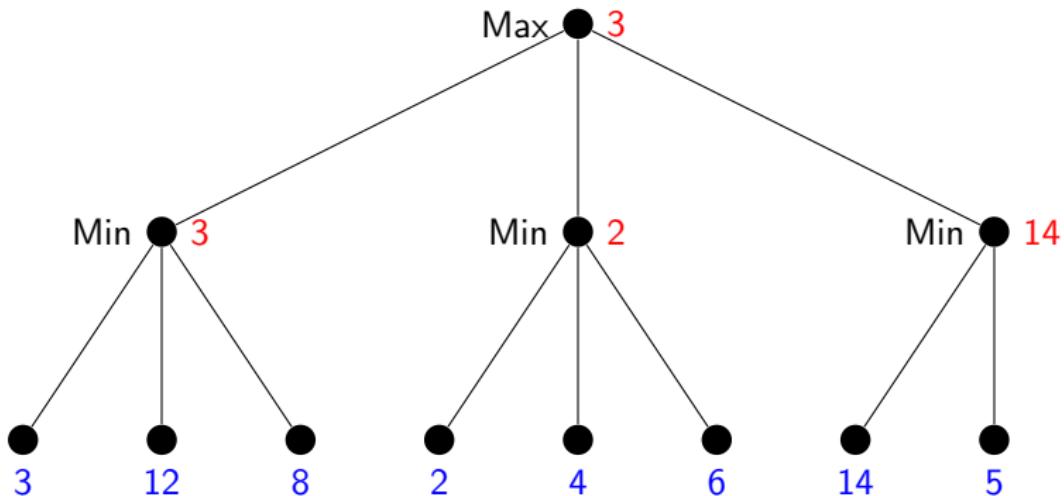
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



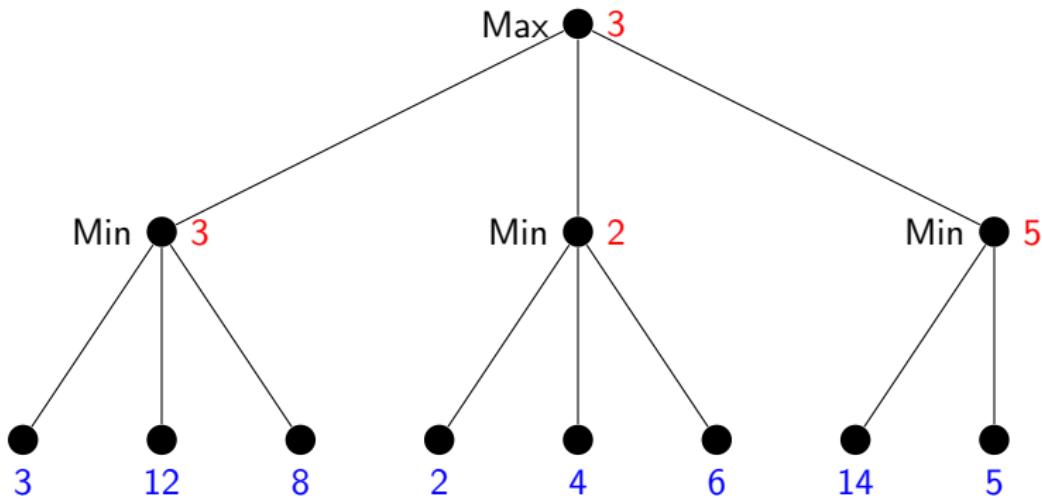
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



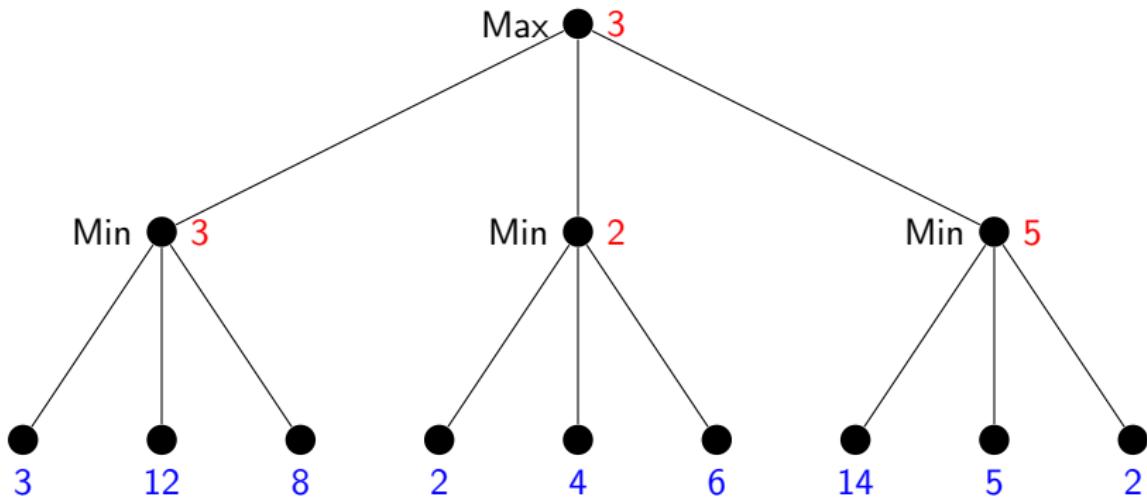
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



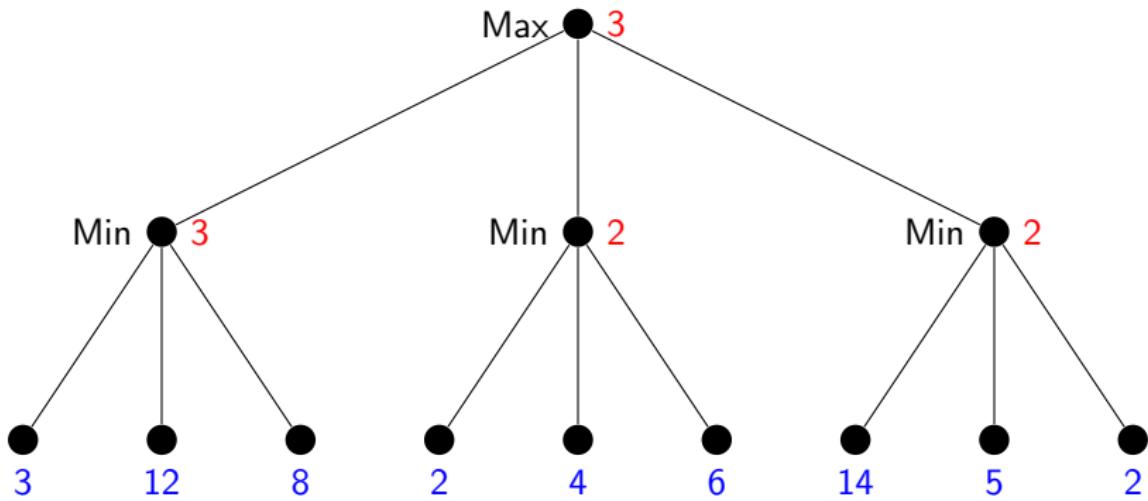
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



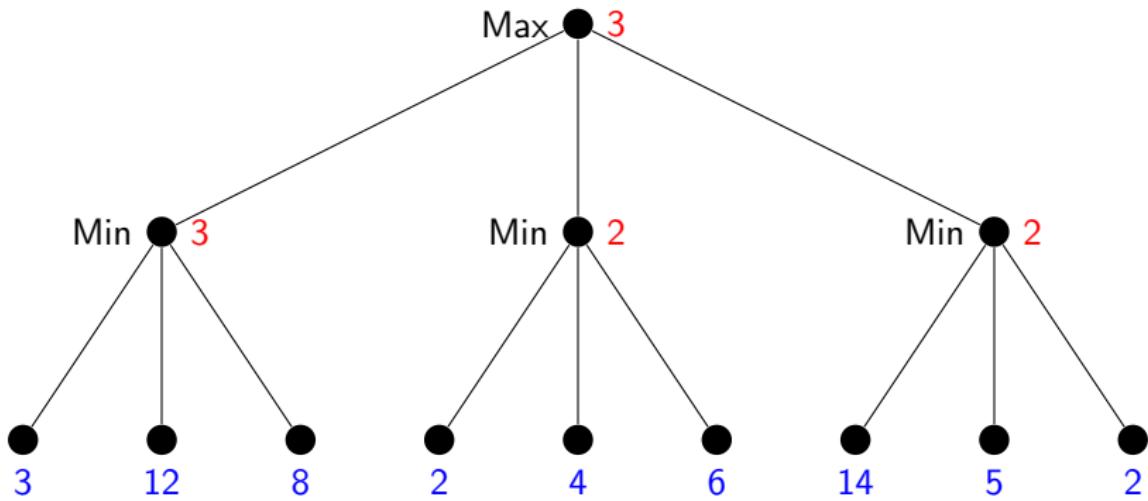
- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



- ▶ So which action for Max is returned?

Minimax: Example, Now in Detail



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

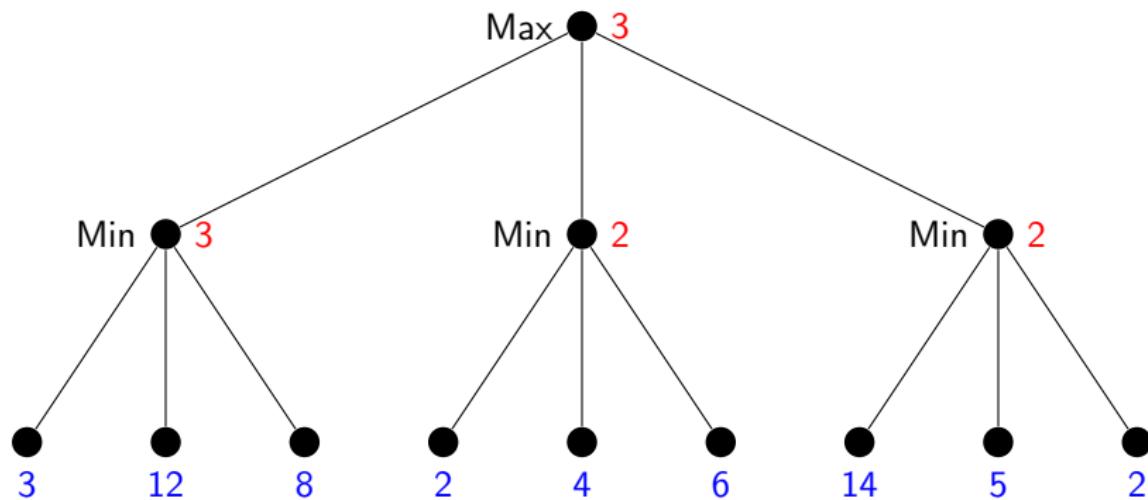
- ▶ Minimax advantages:
 - ▶ Minimax is the simplest possible (reasonable) game search algorithm.
(If any of you sat down, prior to this lecture, to implement a Tic-Tac-Toe player, chances are you either looked this up on Wikipedia, or invented it in the process.)
 - ▶ Returns an optimal action, assuming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

3 Evaluation Functions

Evaluation Functions for Minimax

- ▶ **Problem:** Game tree too big so search through in **minimax**.
- ▶ **Solution:** We impose a **search depth limit** (also called **horizon**) d , and apply an evaluation function to the non-terminal **cut off states**, i.e. states s with $\text{dp}(s) > d$.
- ▶ **Definition 3.1.** An **evaluation function** f maps game states to numbers:
 - ▶ $f(s)$ is an estimate of the actual value of s (as would be computed by unlimited-depth Minimax for s).
 - ▶ If cut-off state is terminal: Just use 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**)

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



- ▶ Blue: Evaluation function f , applied to the cut-off states at $d = 2$.
- ▶ Red: Utilities of inner nodes, as computed by Minimax using d, f .



- ▶ Evaluation function in Chess:
 - ▶ **Material:** Pawn (Bauer) 1, Knight (Springer) 3, Bishop (Läufer) 3, Rook (Turm) 5, Queen (Dame) 9.
 - ▶ 3 points advantage \sim 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)

- ▶ A common approach is to use a **weighted linear function** for f :

$$w_1 f_1 + w_2 f_2 + \cdots + w_n f_n$$

where the w_i are the **weights**, and the f_i are the **features**.

- ▶ **Problem:** How to obtain these weighted linear functions?
 - ▶ Weights w_i can be learned automatically.
 - ▶ The features f_i , however, have to be designed by human experts.
- ▶ **Note:**
 - ▶ Very fast, very simplistic.
 - ▶ Can be computed **incrementally**: In transition $s \xrightarrow{a} s'$, adapt $f(s)$ to $f(s')$ by considering only those features whose values have changed.

The Horizon Problem

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



Black to move

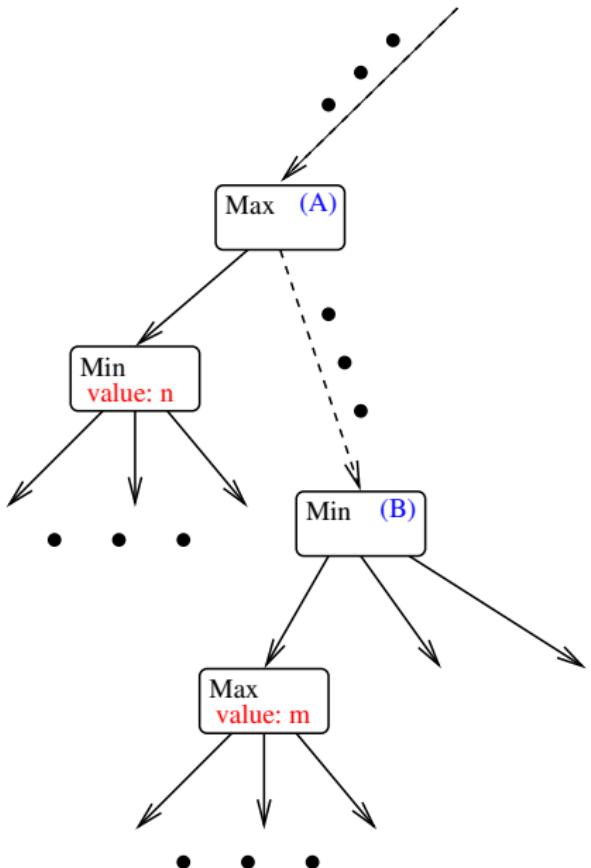
- ▶ Who's gonna win here?
 - ▶ White wins (Pawn cannot be prevented from becoming a queen.)
 - ▶ Black has a +4 advantage in material, so if we cut-off here then our evaluation function will say “−100, black wins”.
 - ▶ The loss for black is “beyond our horizon” unless we search extremely deeply: Black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

So, How Deeply to Search?

- ▶ **Goal:** In given time, search as deeply as possible.
- ▶ **Problem:** Very difficult to predict search runtime.
- ▶ **Solution:** Iterative deepening.
 - ▶ Search with depth limit $d = 1, 2, 3, \dots$
 - ▶ Time's up: Return result of deepest completed search.
- ▶ **Better Solution:** Quiescence search
 - ▶ Dynamically adapted d .
 - ▶ Search more deeply in “unquiet” positions, where value of evaluation function changes a lot in neighboring states.
 - ▶ Example 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.

4 Alpha-Beta Search

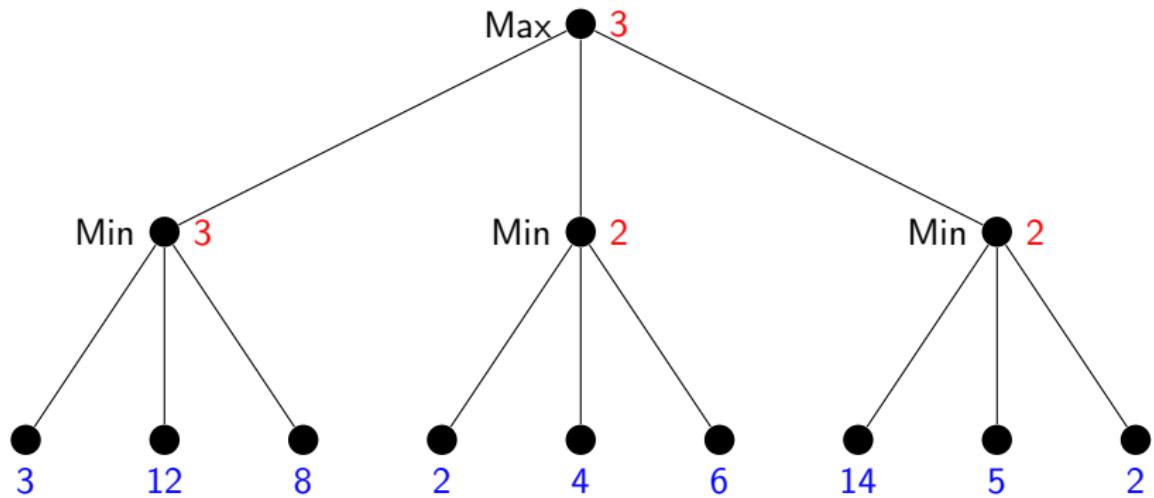
When We Already Know We Can Do Better Than This



- ▶ Say $n > m$.
- ▶ By choosing to go to the left in search node (A), Max already can get utility at least n in this part of the game.
- ▶ So, if “later on” (further down in the same sub-tree), 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).

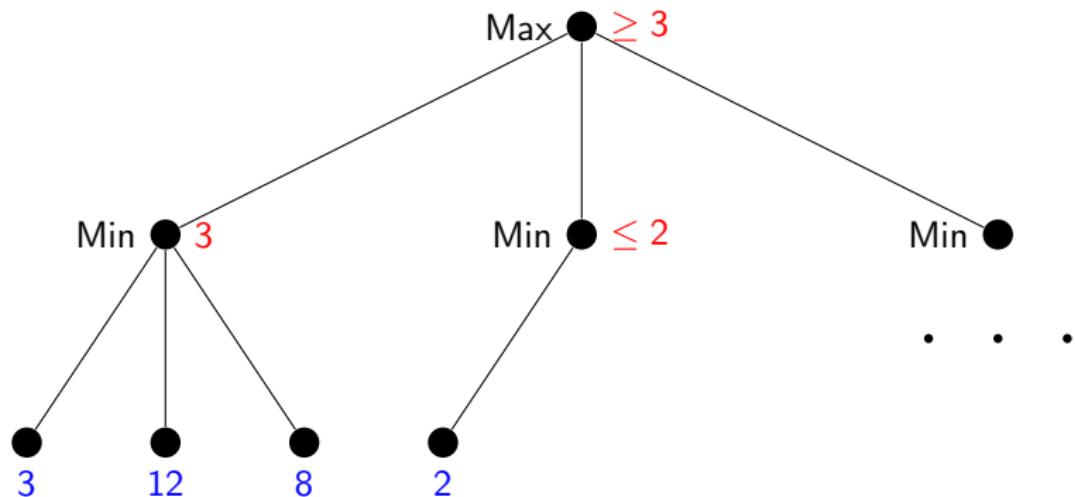
Alpha Pruning: Basic Idea

► 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.



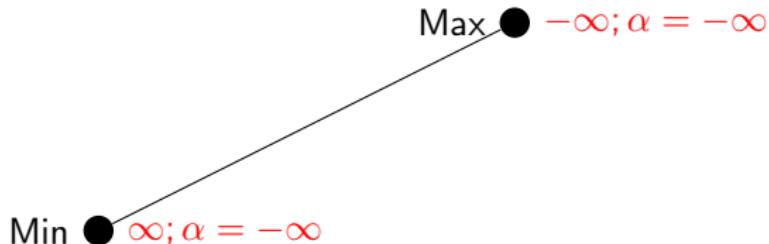
Alpha Pruning

- ▶ 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 .

Max $\bullet -\infty; \alpha = -\infty$

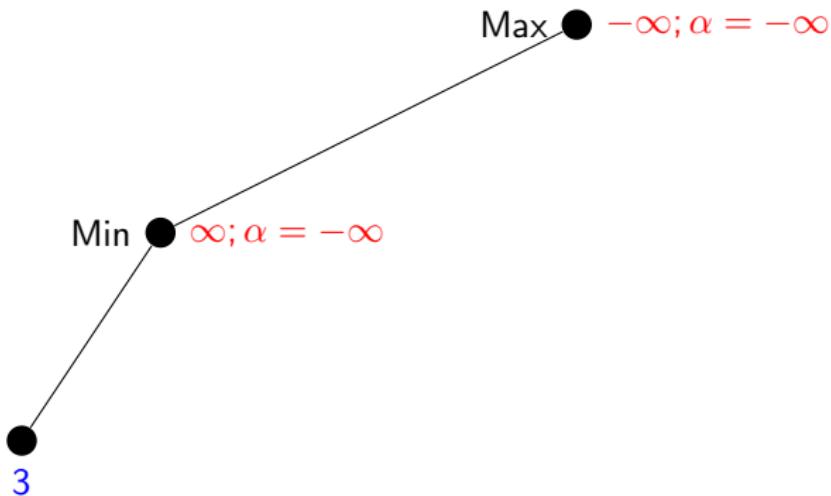
Alpha Pruning

- ▶ 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 .



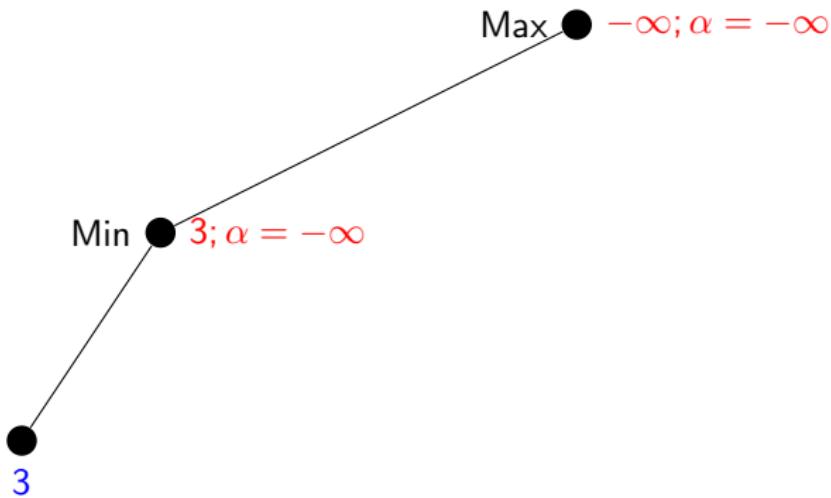
Alpha Pruning

- ▶ 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 .



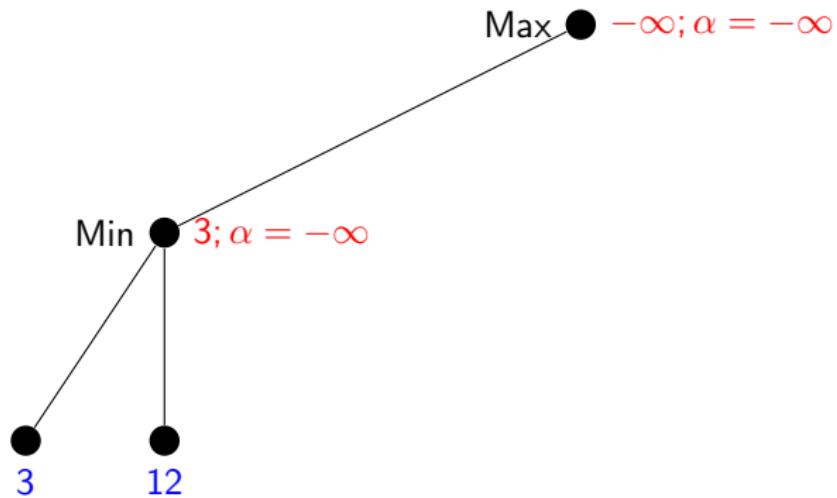
Alpha Pruning

- ▶ 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 .



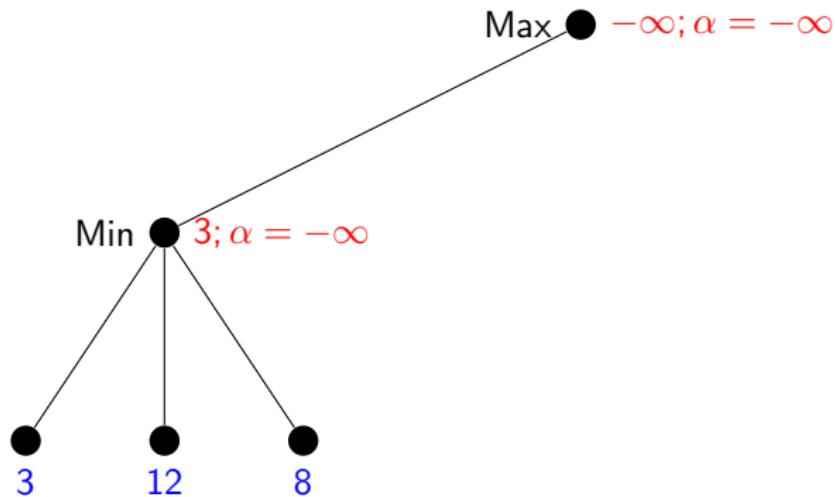
Alpha Pruning

- ▶ 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 .



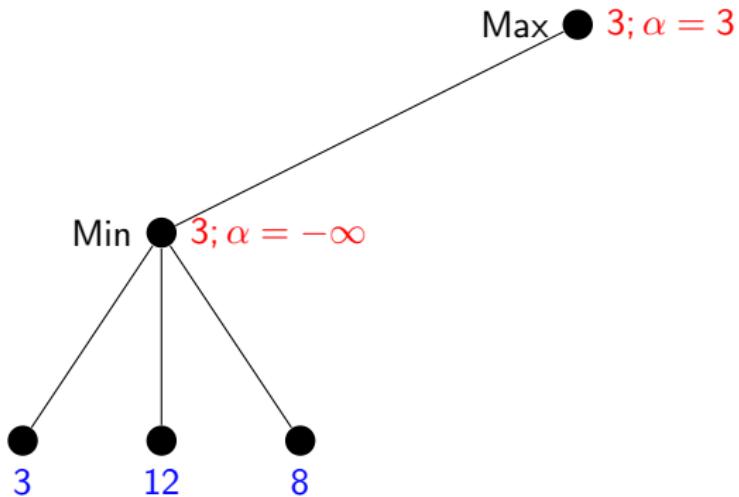
Alpha Pruning

- ▶ 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 .



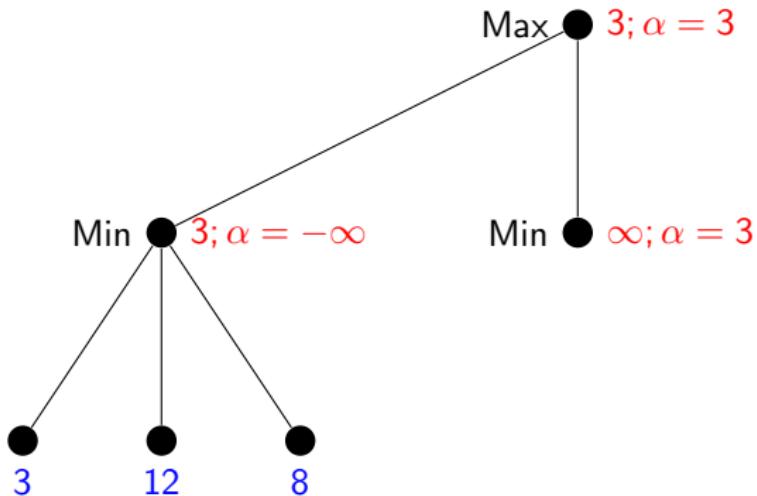
Alpha Pruning

- ▶ 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 .



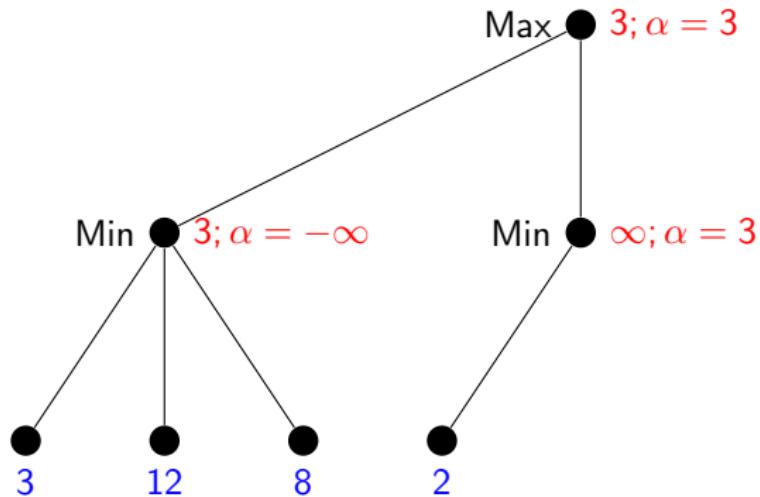
Alpha Pruning

- ▶ 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 .



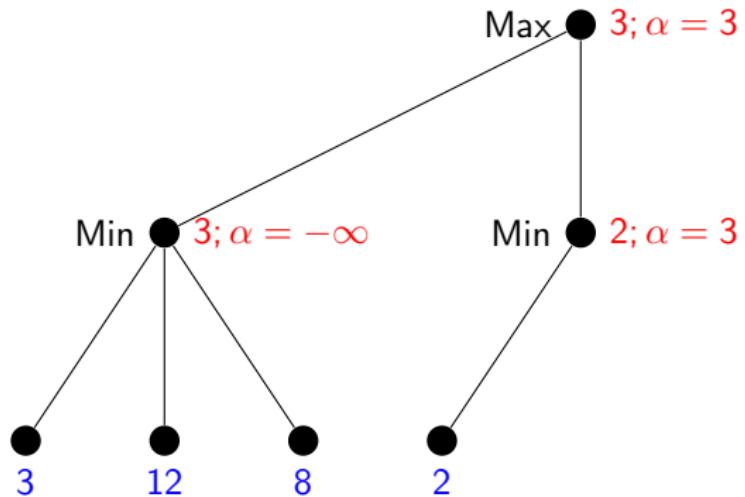
Alpha Pruning

- ▶ 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 .



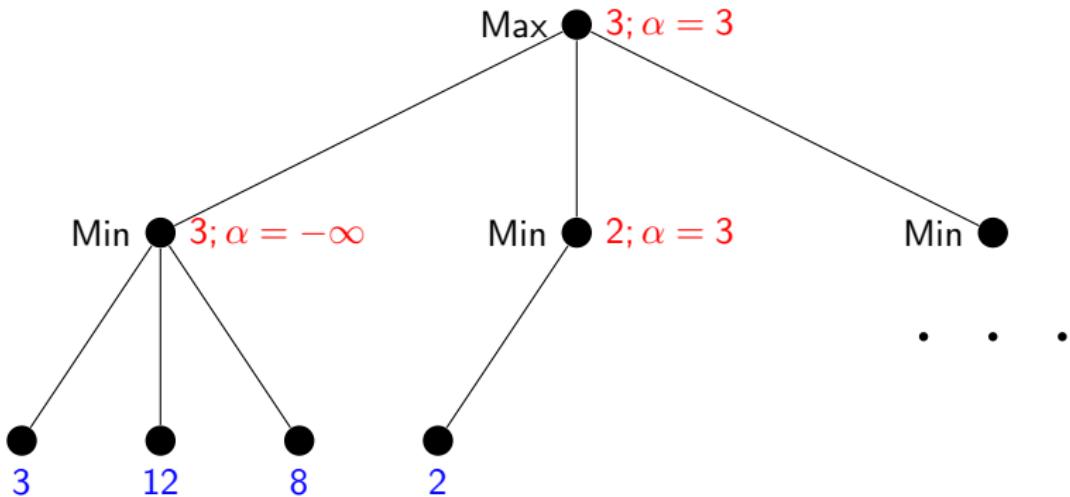
Alpha Pruning

- ▶ 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 .



Alpha Pruning

- ▶ 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 $\leq \alpha$, then stop considering n . (Pruning out its remaining successors.)

- ▶ 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 $\leq \alpha$, then stop considering n . (Pruning out its remaining successors.)
- ▶ Idea: We can use a dual method for Min:
 - ▶ What is β : For each search node n , the lowest 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 $\geq \beta$, then stop considering n . (Pruning out its remaining successors.)
- ▶ ...and of course we can use both together!

Alpha-Beta Search: Pseudo-Code

- **Definition 4.1.** The **alphabeta search** algorithm is given by the following pseudo-code

```
function Alpha-Beta-Search ( $s$ ) returns an action
     $v := \text{Max-Value}(s, -\infty, +\infty)$ 
    return an action yielding value  $v$  in the previous function call

function Max-Value( $s, \alpha, \beta$ ) returns a utility value
    if Terminal-Test( $s$ ) then return  $u(s)$ 
     $v := -\infty$ 
    for each  $a \in \text{Actions}(s)$  do
         $v := \max(v, \text{Min-Value}(\text{ChildState}(s, a), \alpha, \beta))$ 
         $\alpha := \max(\alpha, v)$ 
        if  $v \geq \beta$  then return  $v$  /* Here:  $v \geq \beta \Leftrightarrow \alpha \geq \beta */$ 
    return  $v$ 

function Min-Value( $s, \alpha, \beta$ ) returns a utility value
    if Terminal-Test( $s$ ) then return  $u(s)$ 
     $v := +\infty$ 
    for each  $a \in \text{Actions}(s)$  do
         $v := \min(v, \text{Max-Value}(\text{ChildState}(s, a), \alpha, \beta))$ 
         $\beta := \min(\beta, v)$ 
        if  $v \leq \alpha$  then return  $v$  /* Here:  $v \leq \alpha \Leftrightarrow \alpha \geq \beta */$ 
    return  $v$ 
```

$\hat{=}$ Minimax (slide 182) + α/β book-keeping and pruning.

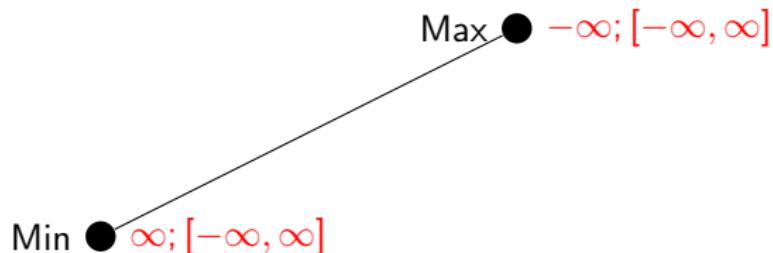
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$

Max ● $-\infty; [-\infty, \infty]$

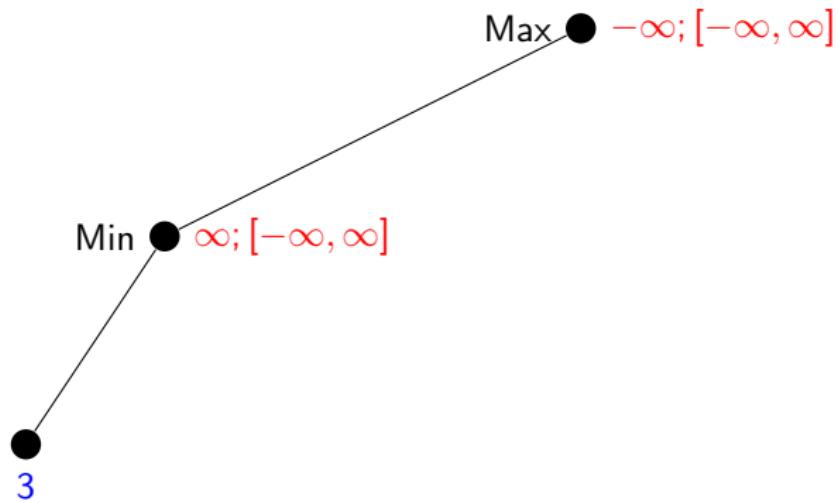
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



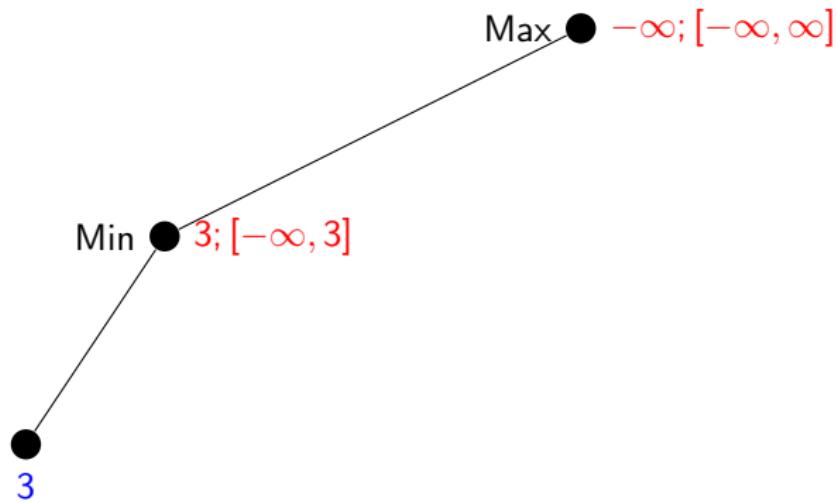
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



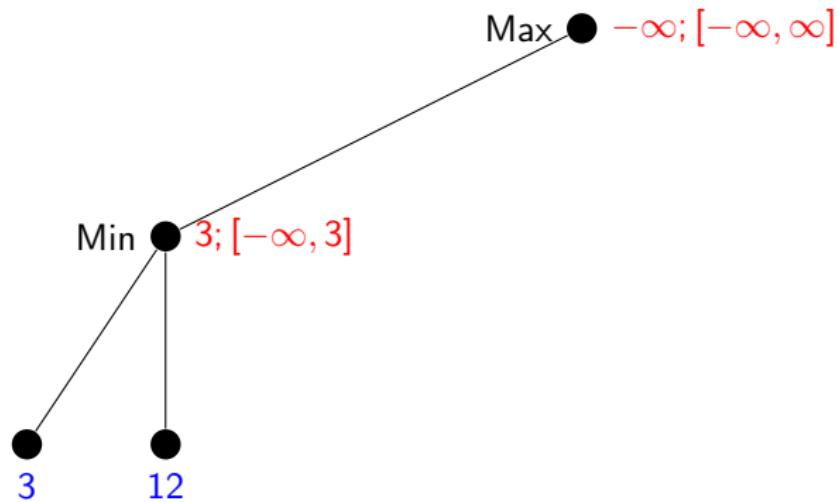
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



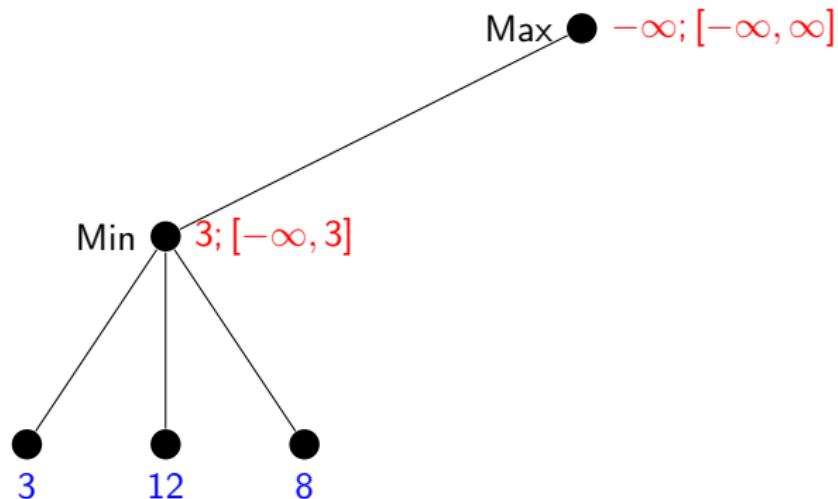
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



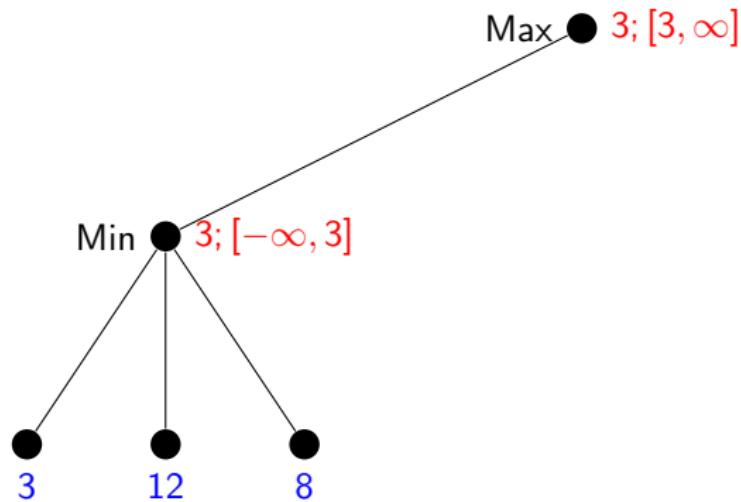
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



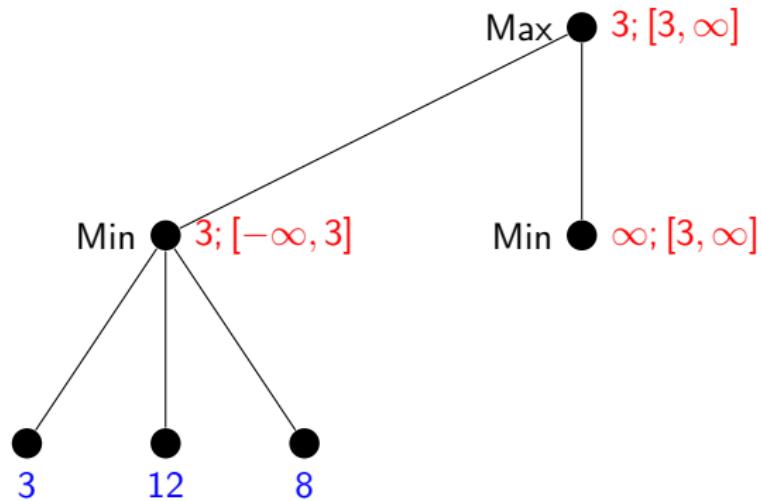
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



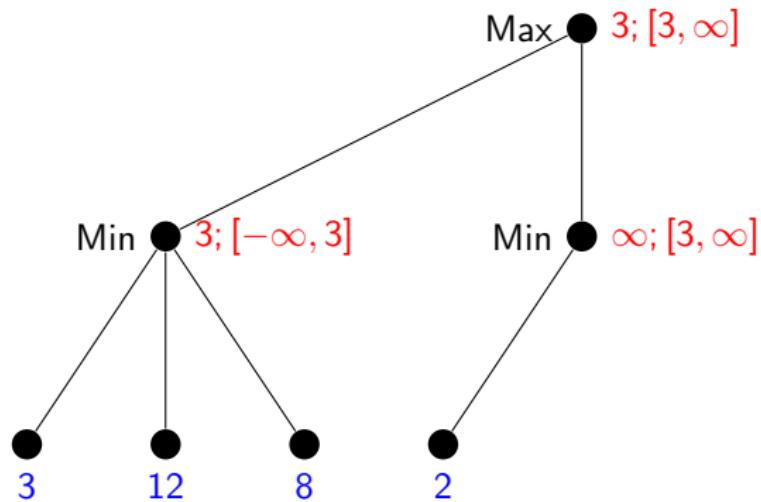
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



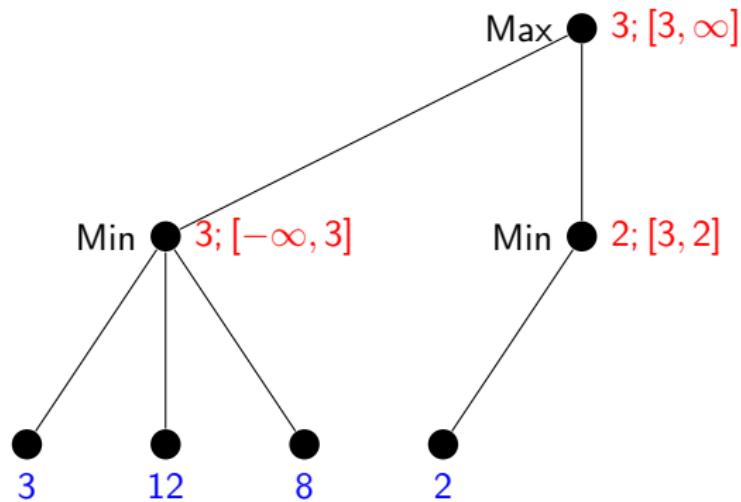
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



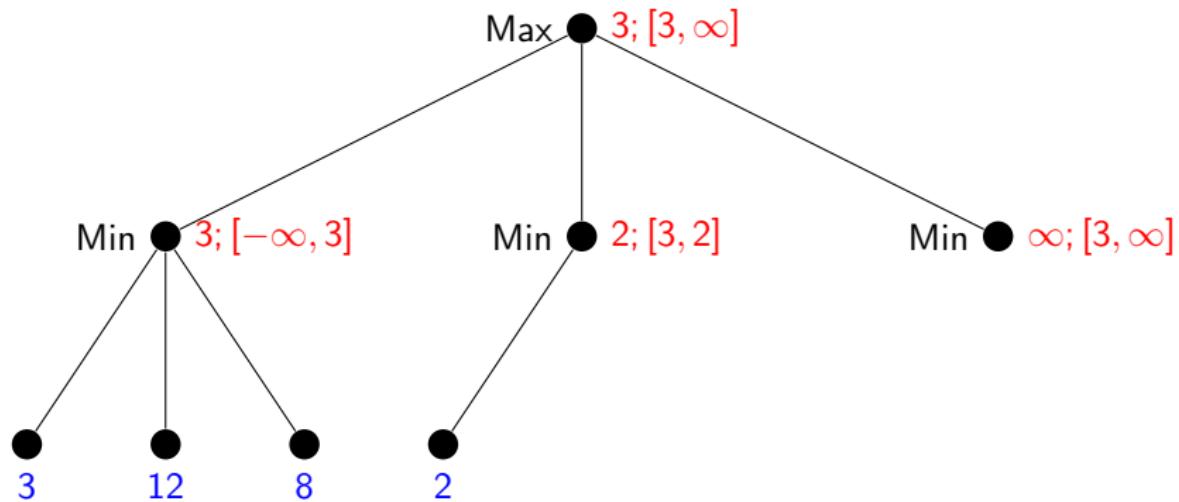
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



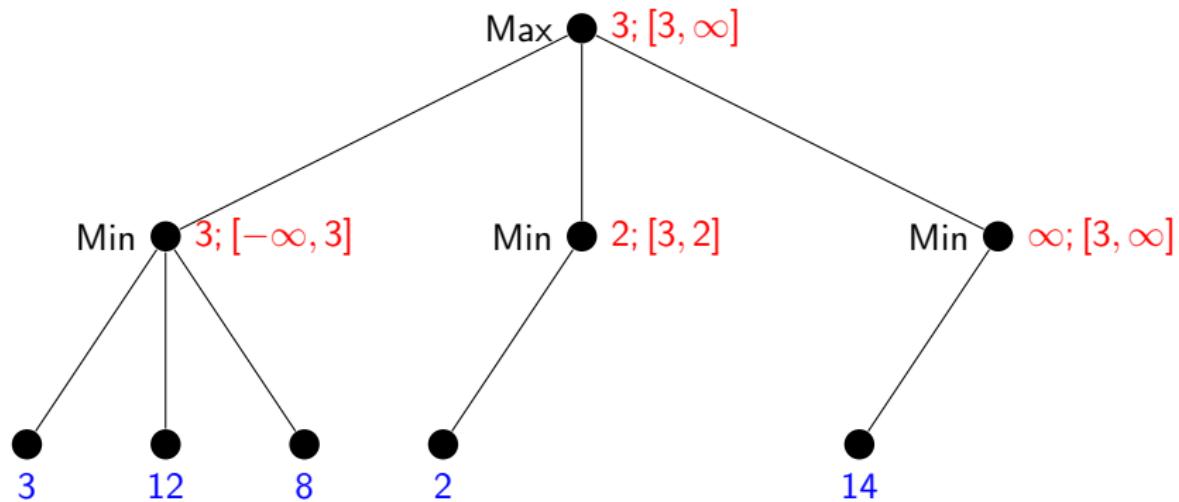
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



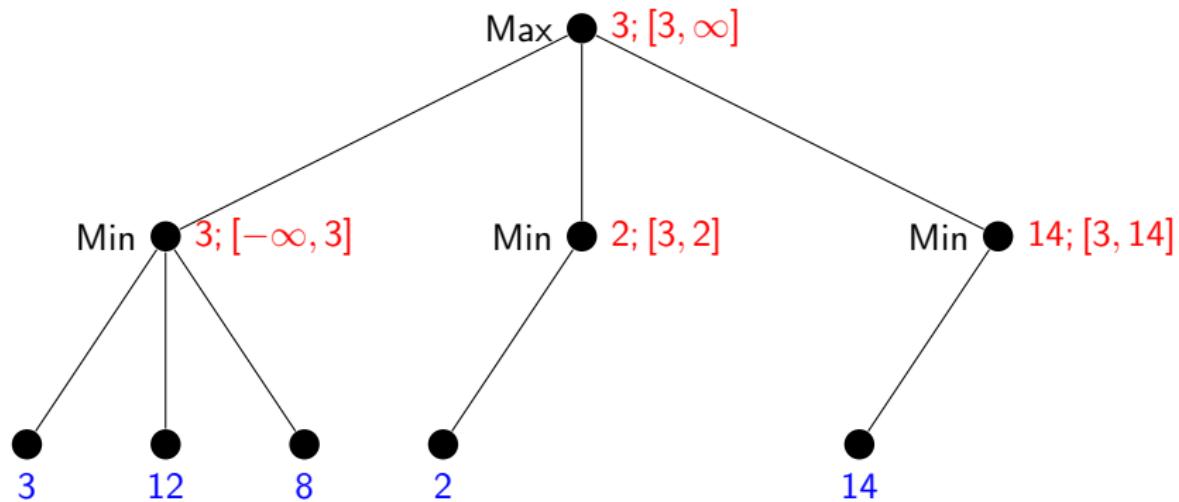
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



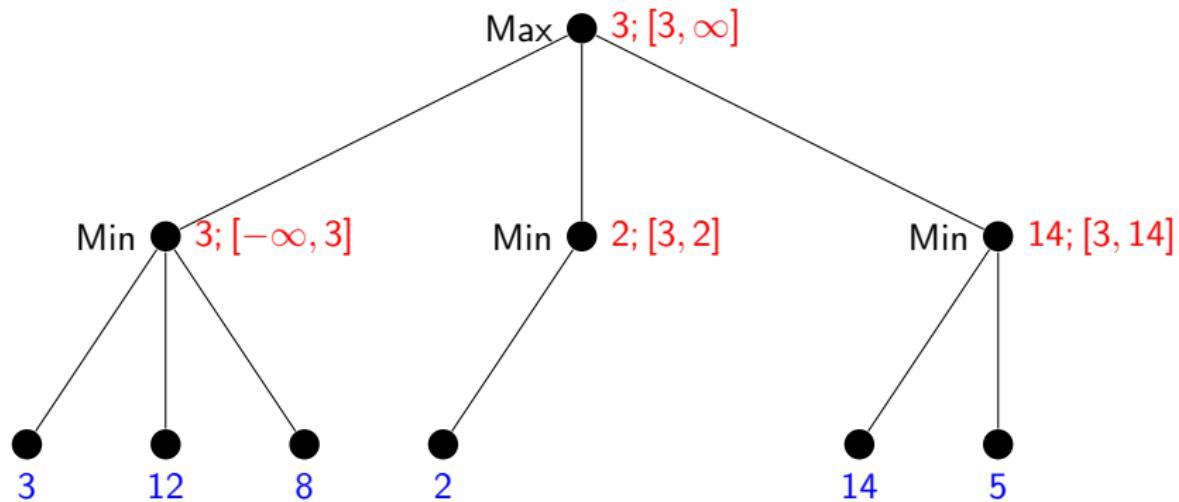
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



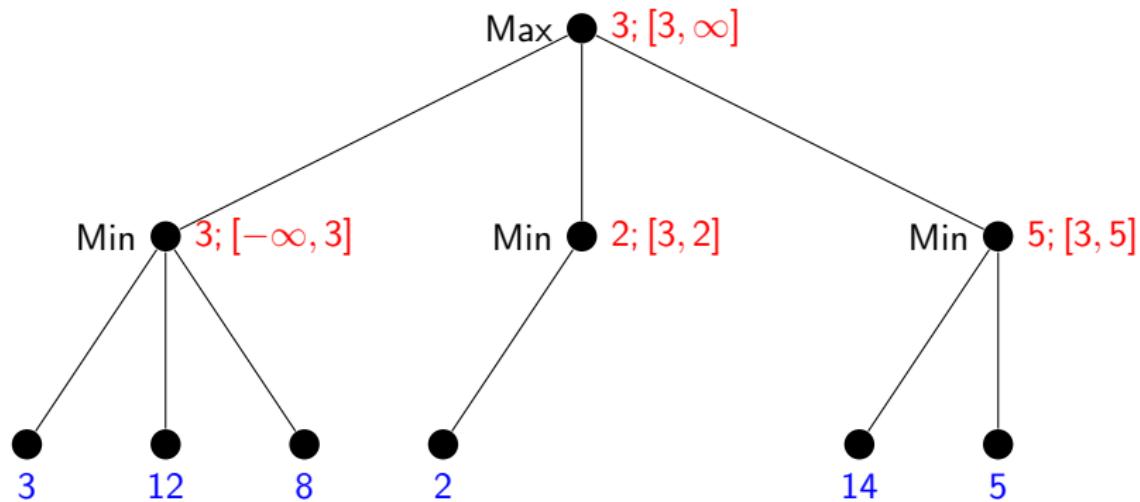
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



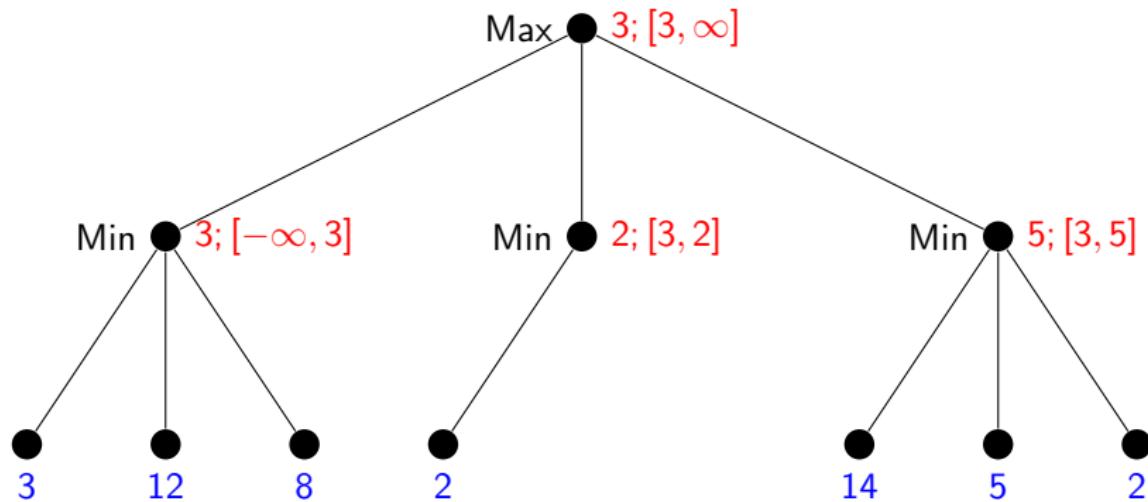
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



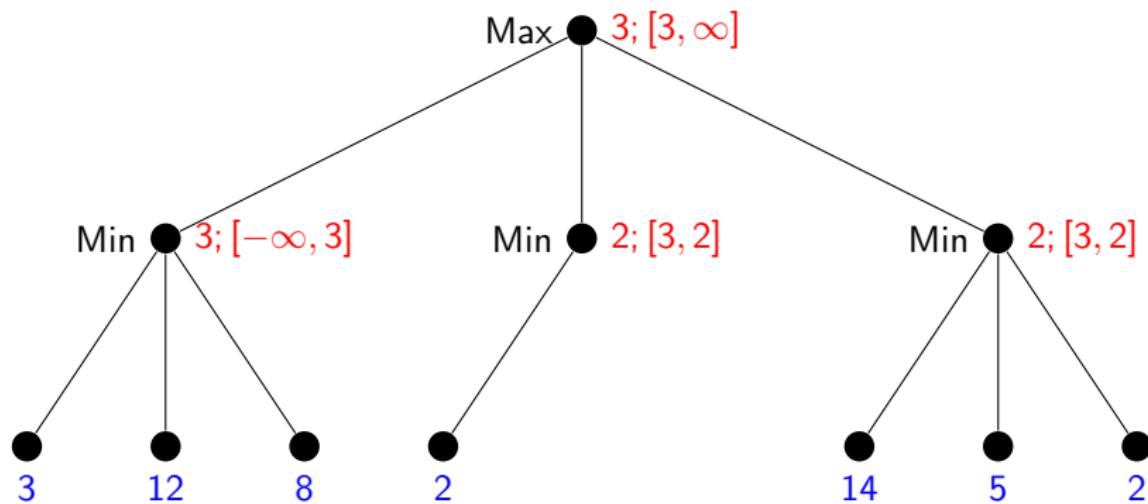
Alpha-Beta Search: Example

- ▶ Notation: $v; [\alpha, \beta]$



Alpha-Beta Search: Example

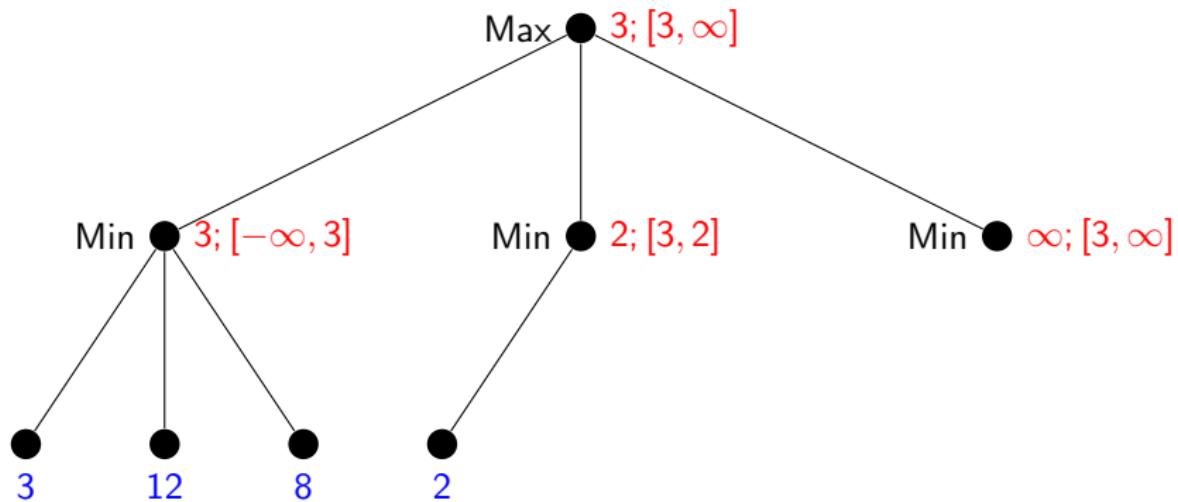
- ▶ Notation: $v; [\alpha, \beta]$



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

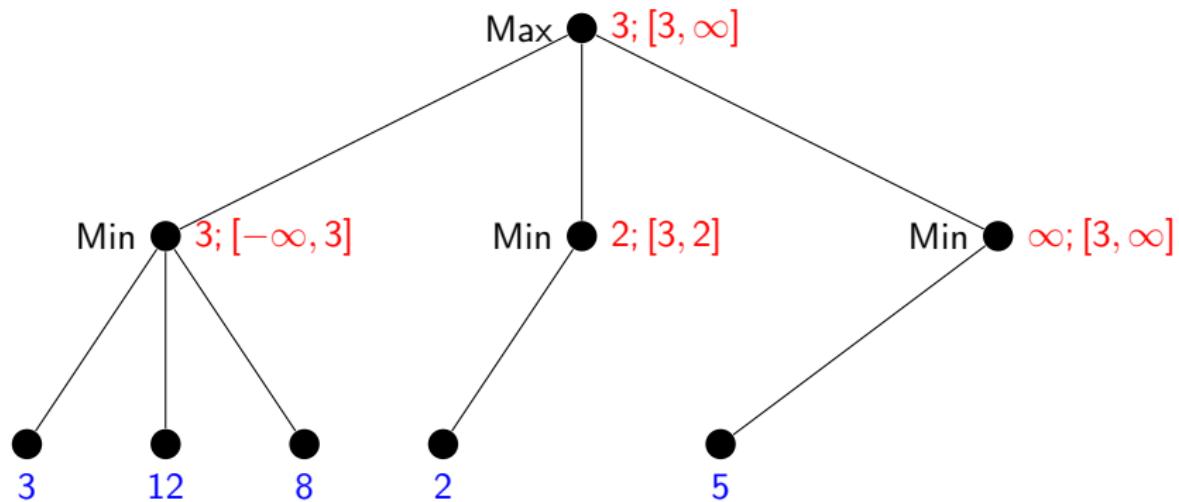
Alpha-Beta Search: Modified Example

- ▶ Showing off some actual β pruning:



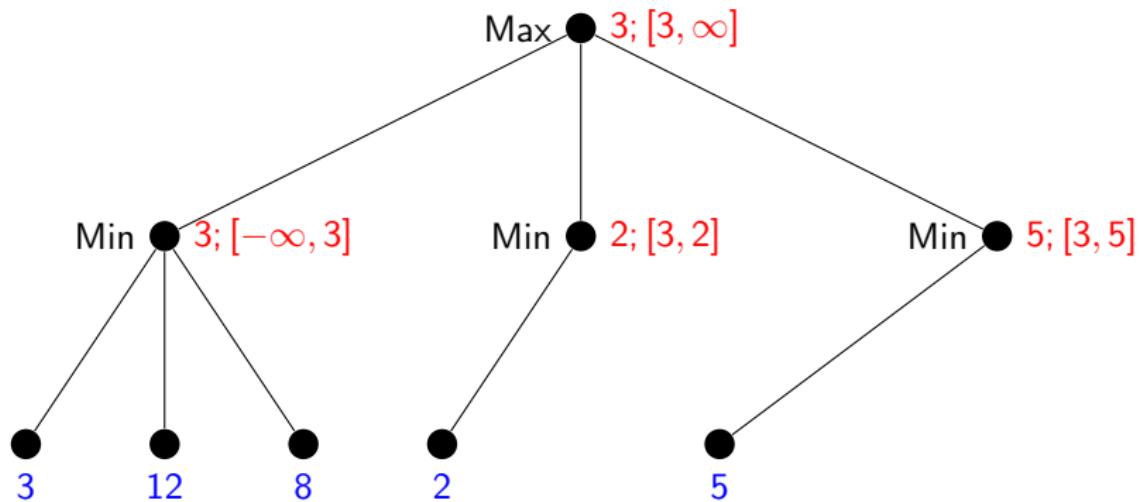
Alpha-Beta Search: Modified Example

- ▶ Showing off some actual β pruning:



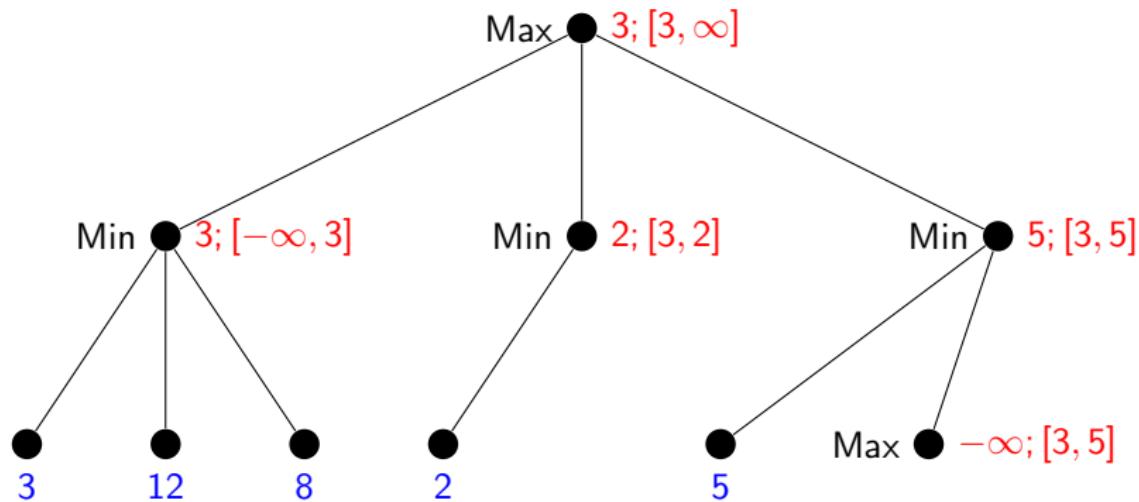
Alpha-Beta Search: Modified Example

- ▶ Showing off some actual β pruning:



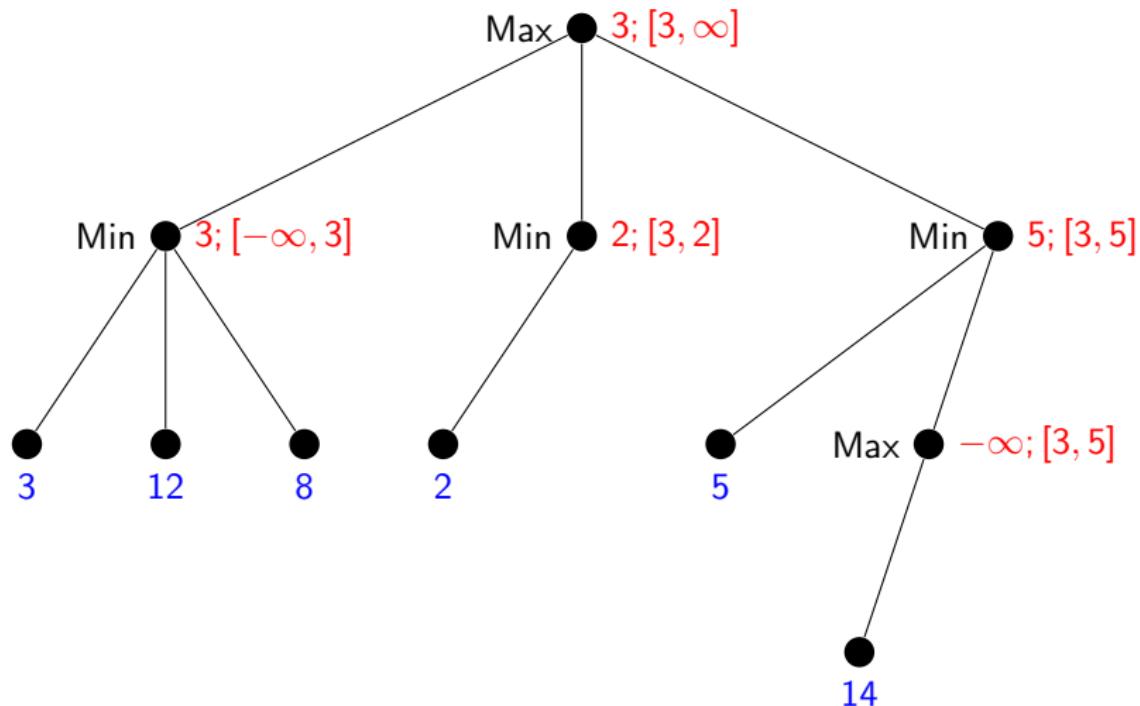
Alpha-Beta Search: Modified Example

- ▶ Showing off some actual β pruning:



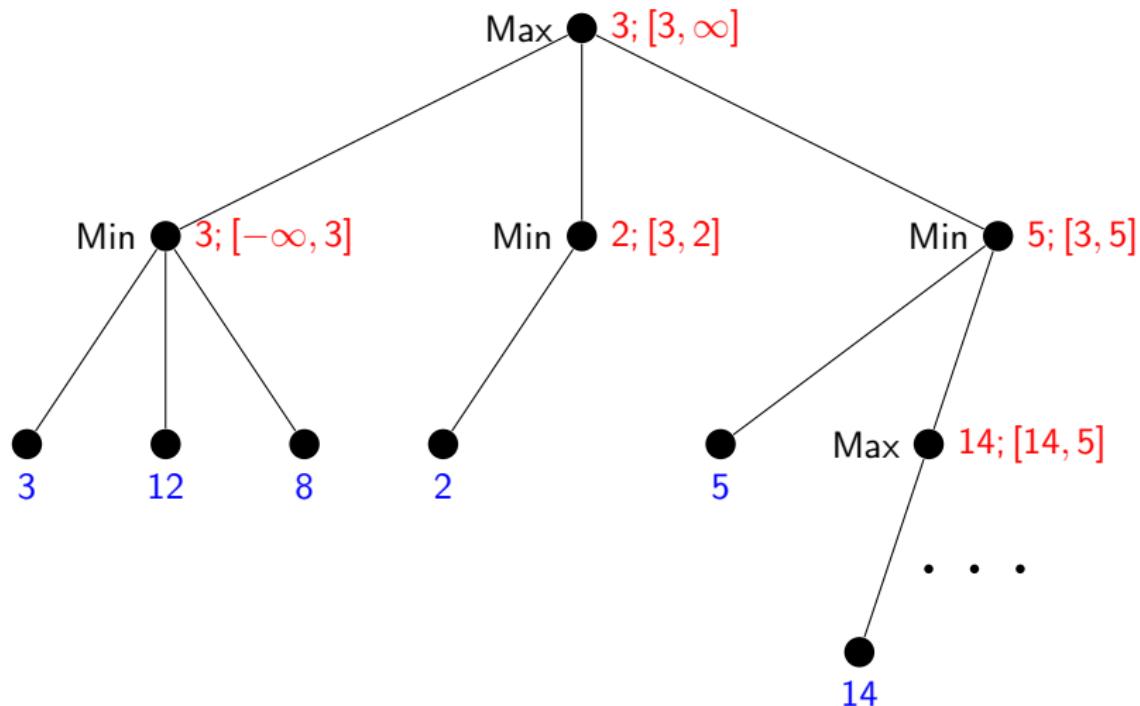
Alpha-Beta Search: Modified Example

- ▶ Showing off some actual β pruning:



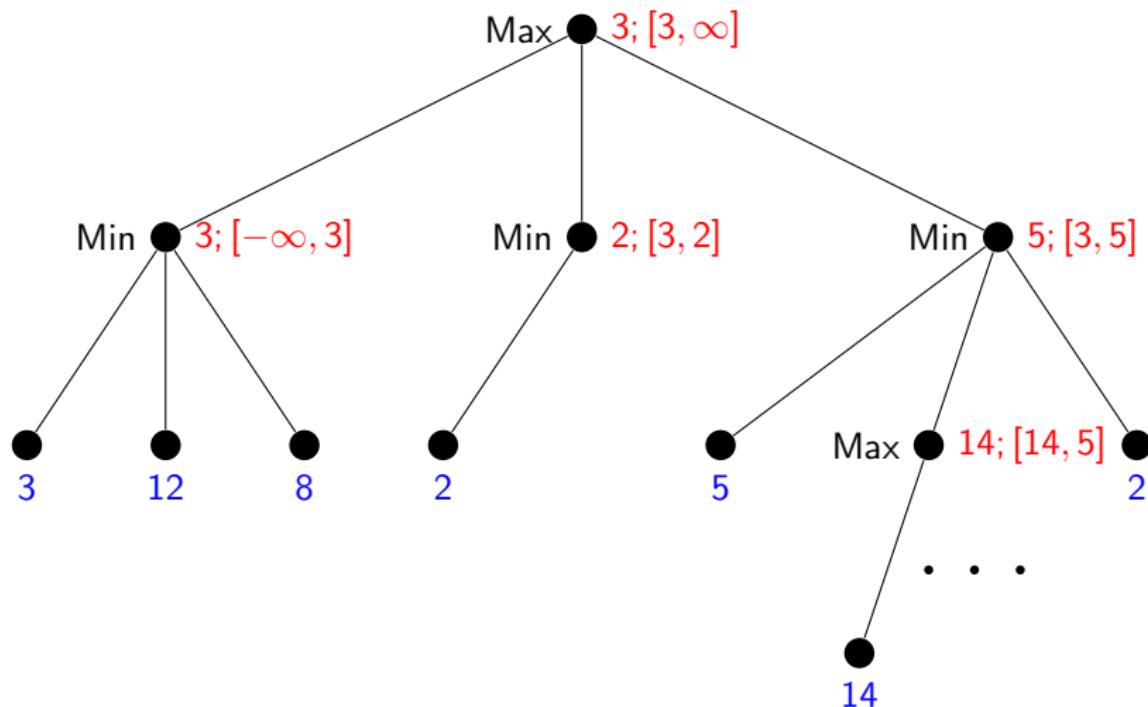
Alpha-Beta Search: Modified Example

- ▶ Showing off some actual β pruning:



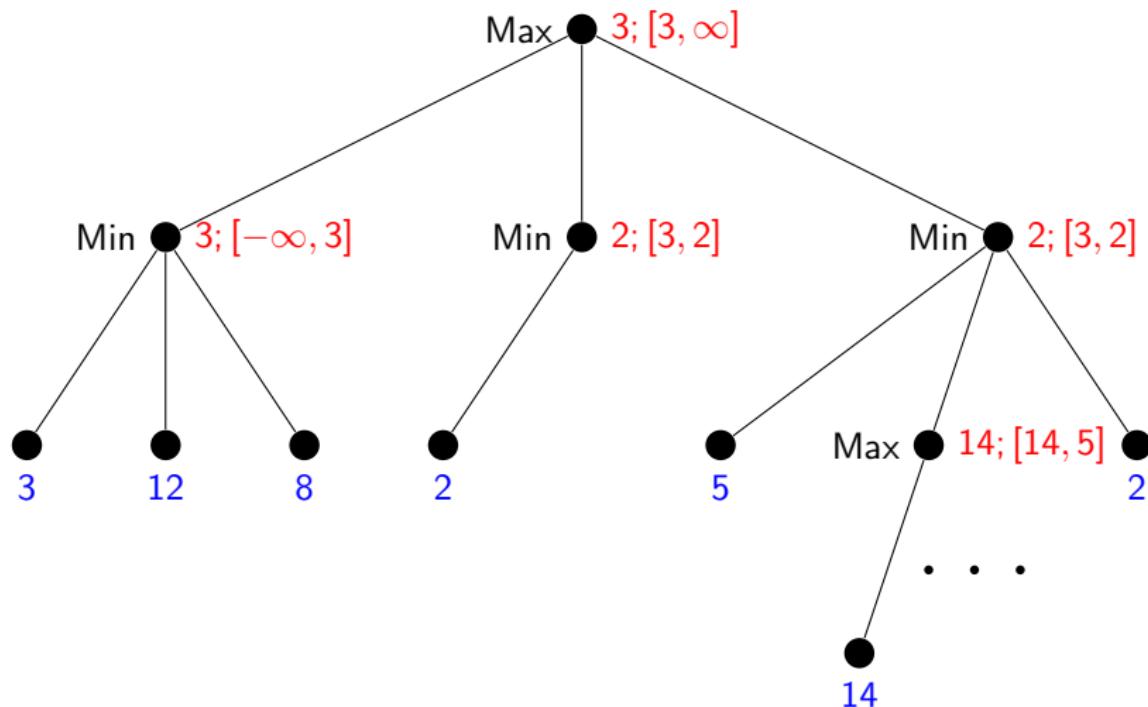
Alpha-Beta Search: Modified Example

- ▶ Showing off some actual β pruning:



Alpha-Beta Search: Modified Example

- ▶ Showing off some actual β pruning:



How Much Pruning Do We Get?

- ▶ Choosing the best moves first yields most pruning in alpha-beta search.
 - ▶ The maximizing moves for Max, the minimizing moves for Min.
- ▶ **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 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.

And now ...



- ▶ **AlphaGo = Monte-Carlo tree search + neural networks**

Monte-Carlo Tree Search: Basic Ideas

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

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

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

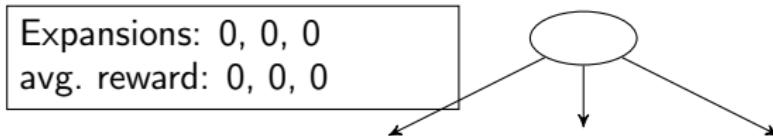
```
while time not up do
    apply actions within T to select a leaf state s'
    select action a' applicable to s', run random sample from a'
    add s' to T, update averages etc.
return an a for s with maximal average u(t)
```

When executing a , keep the part of T below a .

- ▶ Compared to alphabeta search: no exhaustive enumeration.
- ▶ **Pro:** runtime & memory.
- ▶ **Contra:** need good guidance how to “select” and “sample”.

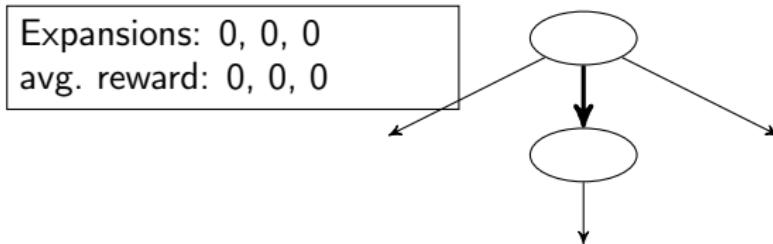
Monte-Carlo Sampling: Illustration of Sampling

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



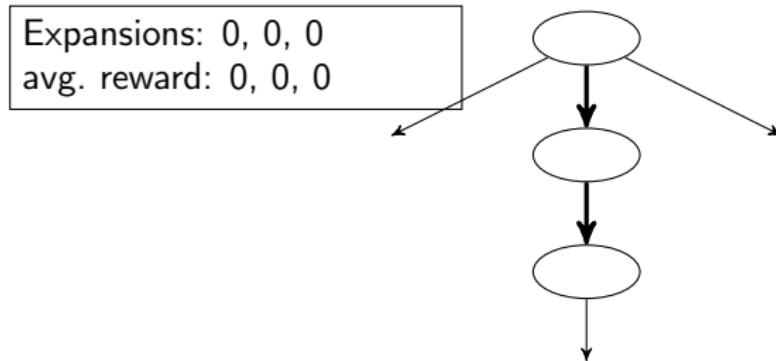
Monte-Carlo Sampling: Illustration of Sampling

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



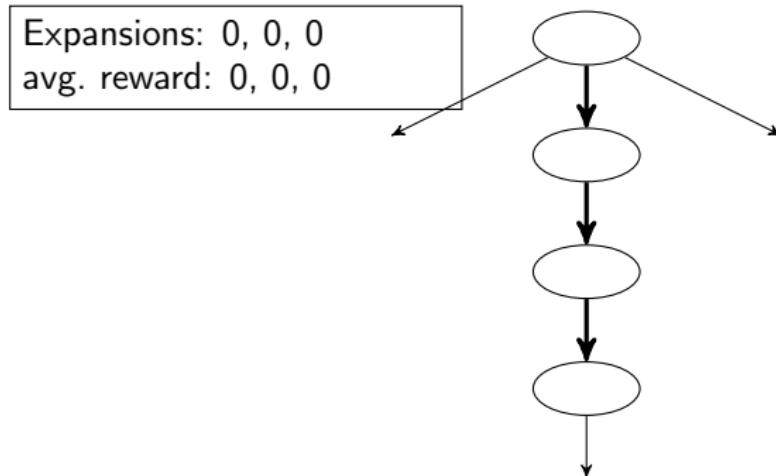
Monte-Carlo Sampling: Illustration of Sampling

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



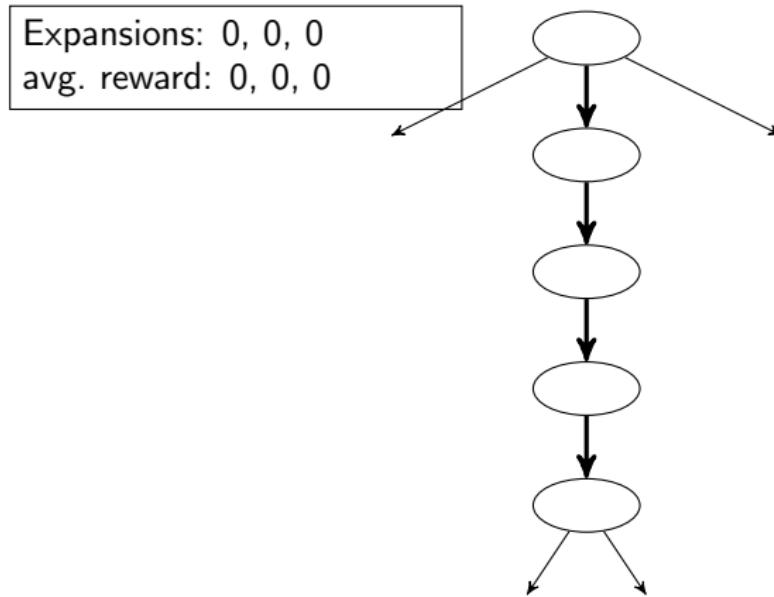
Monte-Carlo Sampling: Illustration of Sampling

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



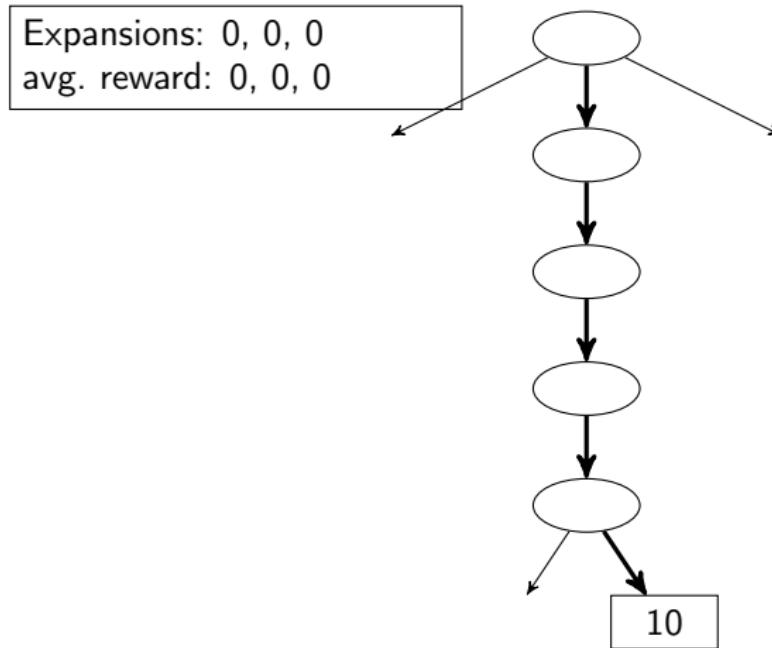
Monte-Carlo Sampling: Illustration of Sampling

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



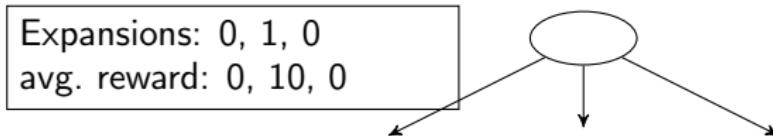
Monte-Carlo Sampling: Illustration of Sampling

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



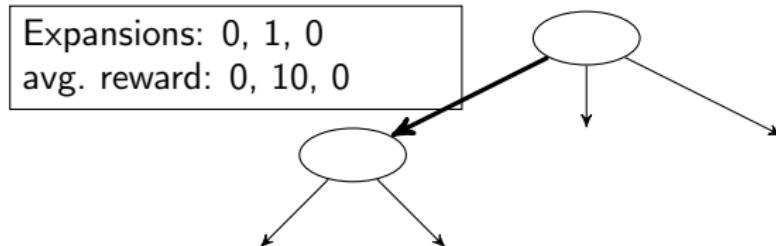
Monte-Carlo Sampling: Illustration of Sampling

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



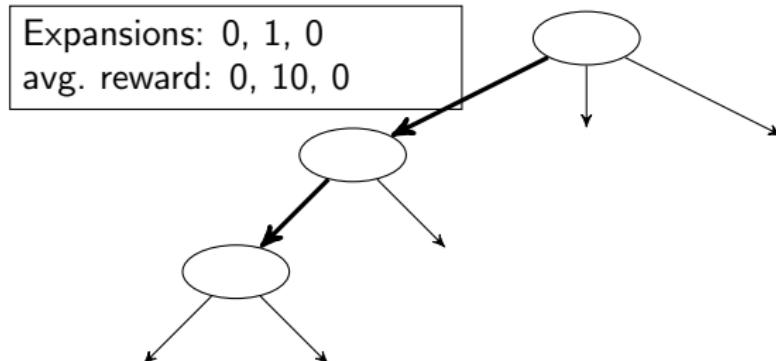
Monte-Carlo Sampling: Illustration of Sampling

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



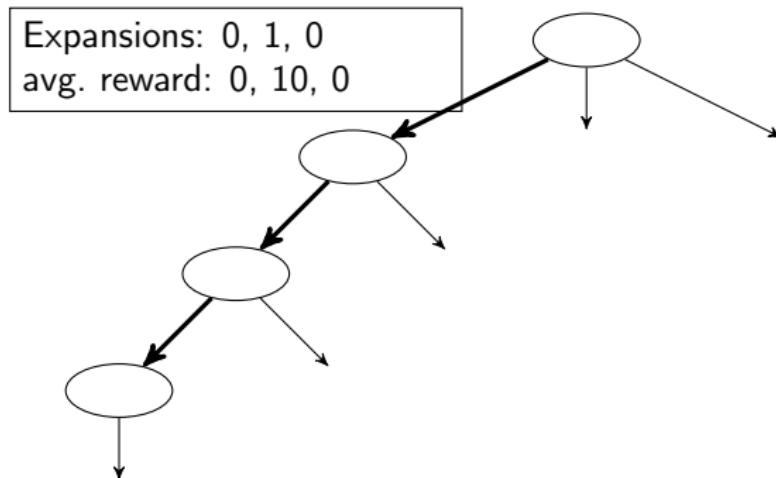
Monte-Carlo Sampling: Illustration of Sampling

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



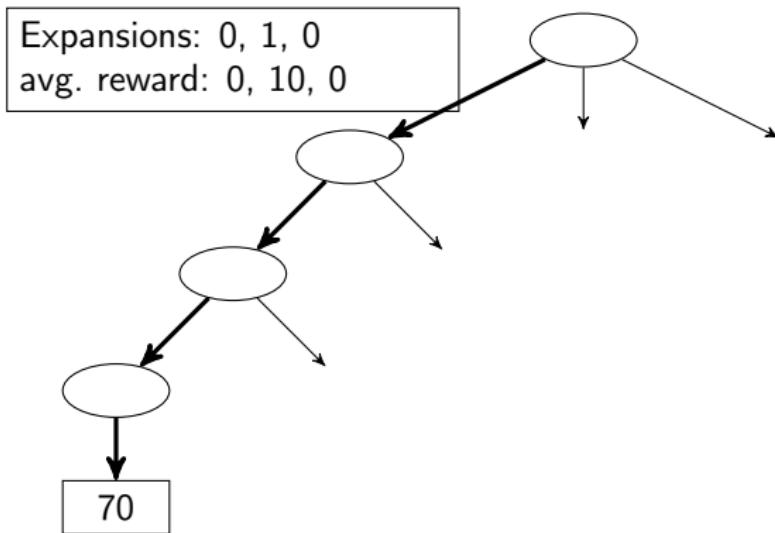
Monte-Carlo Sampling: Illustration of Sampling

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



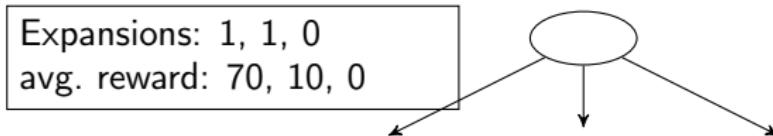
Monte-Carlo Sampling: Illustration of Sampling

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



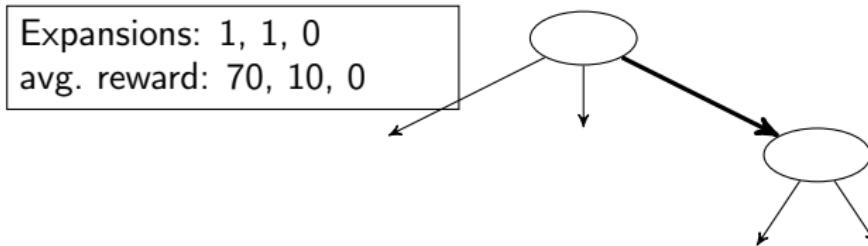
Monte-Carlo Sampling: Illustration of Sampling

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



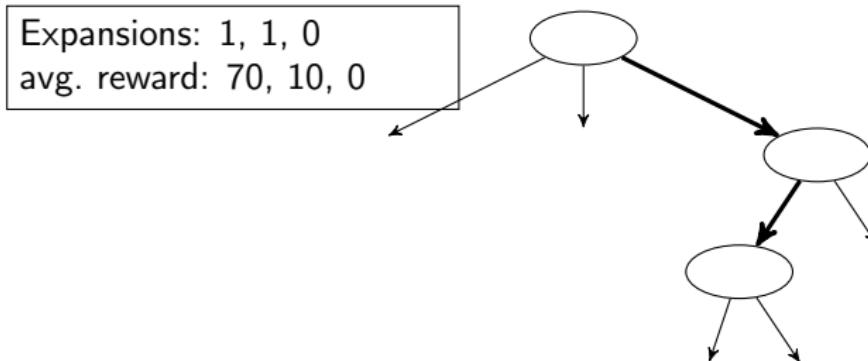
Monte-Carlo Sampling: Illustration of Sampling

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



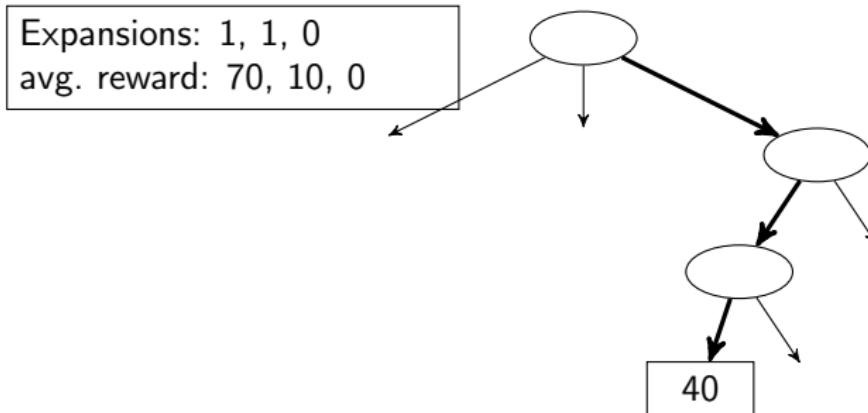
Monte-Carlo Sampling: Illustration of Sampling

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



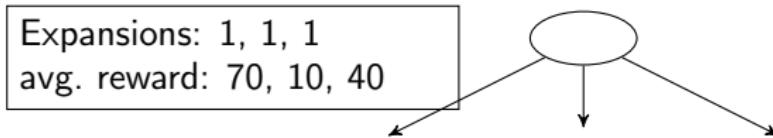
Monte-Carlo Sampling: Illustration of Sampling

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



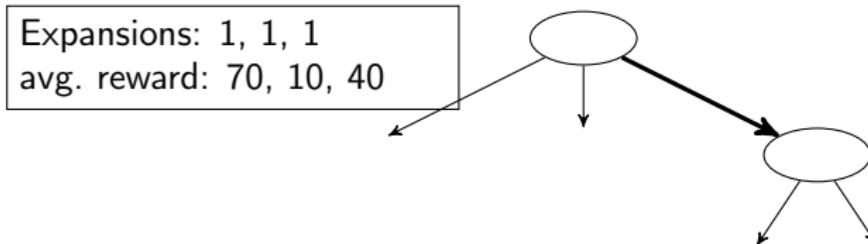
Monte-Carlo Sampling: Illustration of Sampling

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



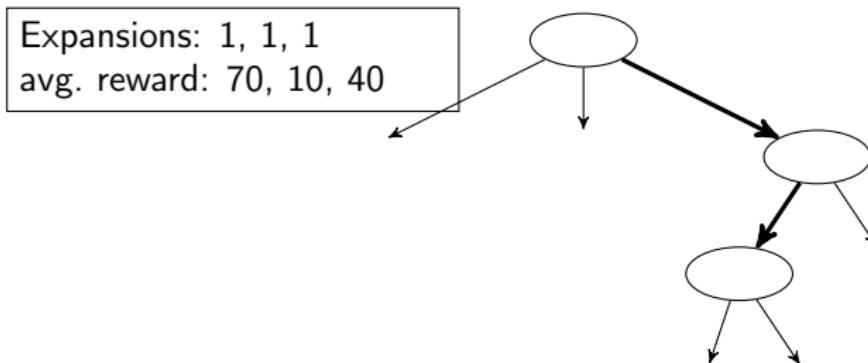
Monte-Carlo Sampling: Illustration of Sampling

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



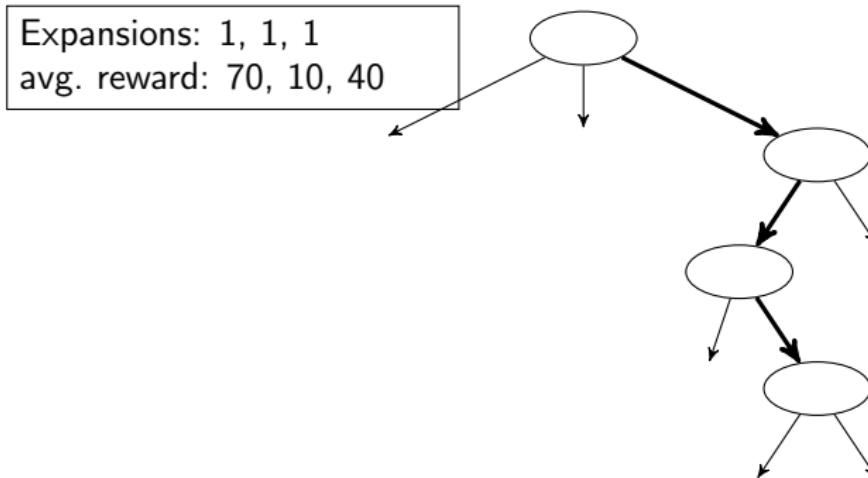
Monte-Carlo Sampling: Illustration of Sampling

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



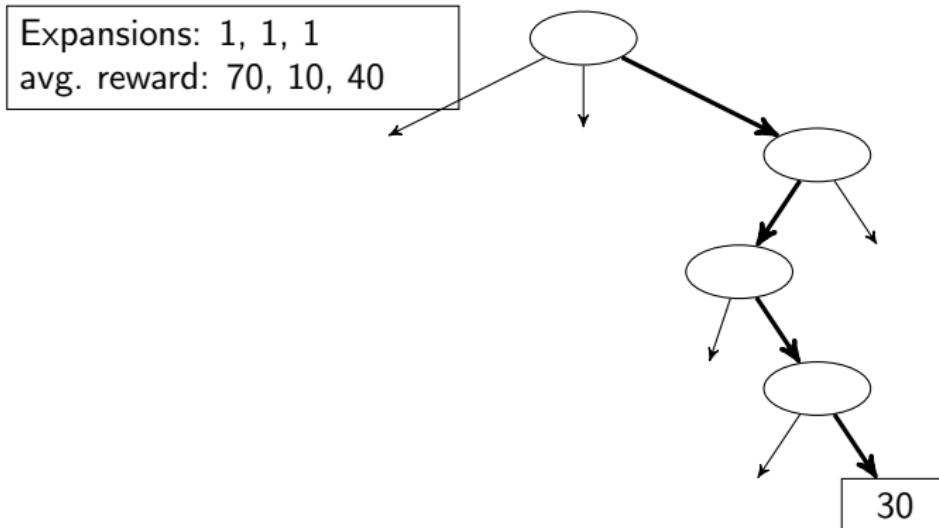
Monte-Carlo Sampling: Illustration of Sampling

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



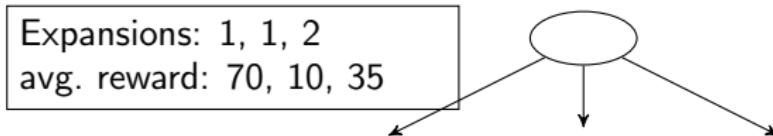
Monte-Carlo Sampling: Illustration of Sampling

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



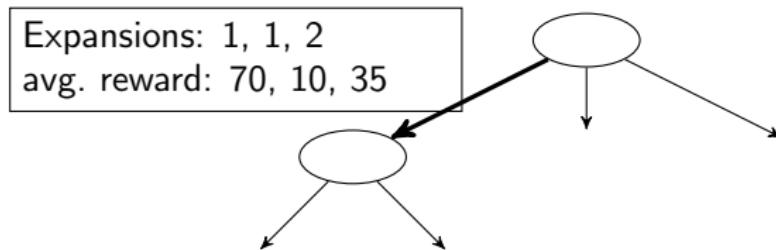
Monte-Carlo Sampling: Illustration of Sampling

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



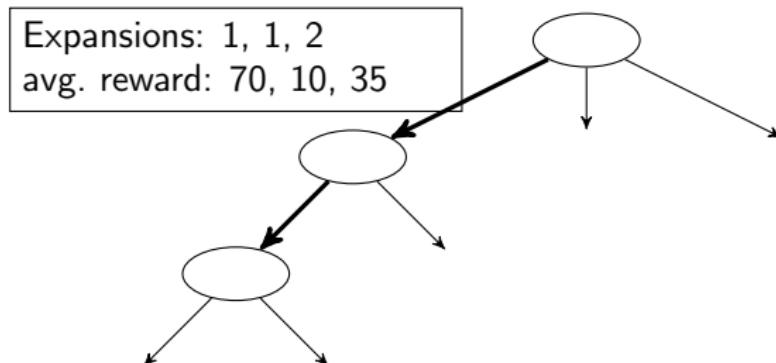
Monte-Carlo Sampling: Illustration of Sampling

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



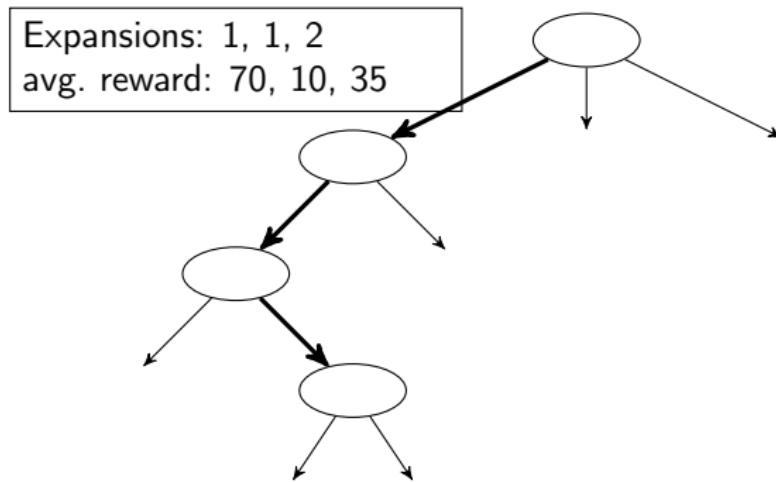
Monte-Carlo Sampling: Illustration of Sampling

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



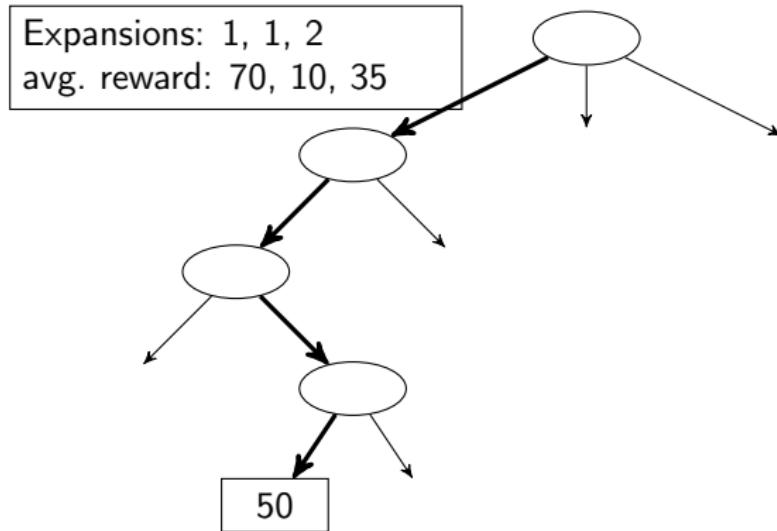
Monte-Carlo Sampling: Illustration of Sampling

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



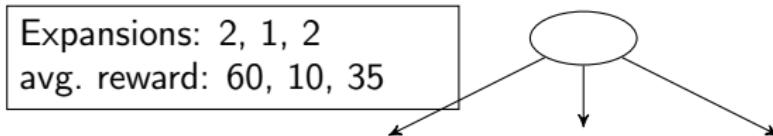
Monte-Carlo Sampling: Illustration of Sampling

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



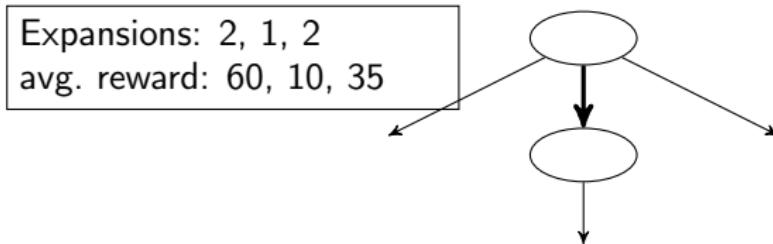
Monte-Carlo Sampling: Illustration of Sampling

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



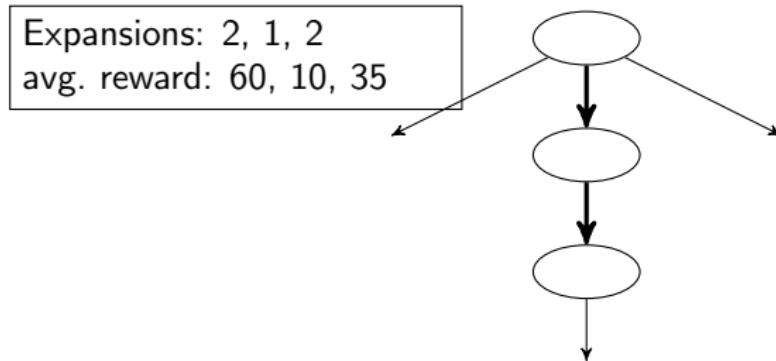
Monte-Carlo Sampling: Illustration of Sampling

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



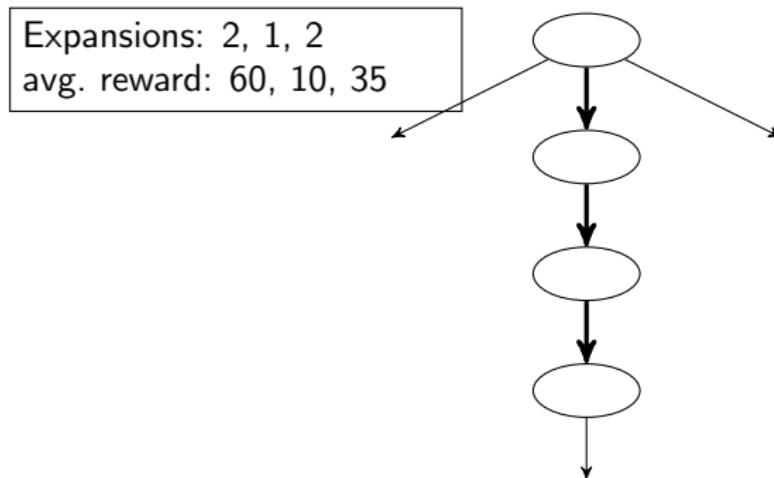
Monte-Carlo Sampling: Illustration of Sampling

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



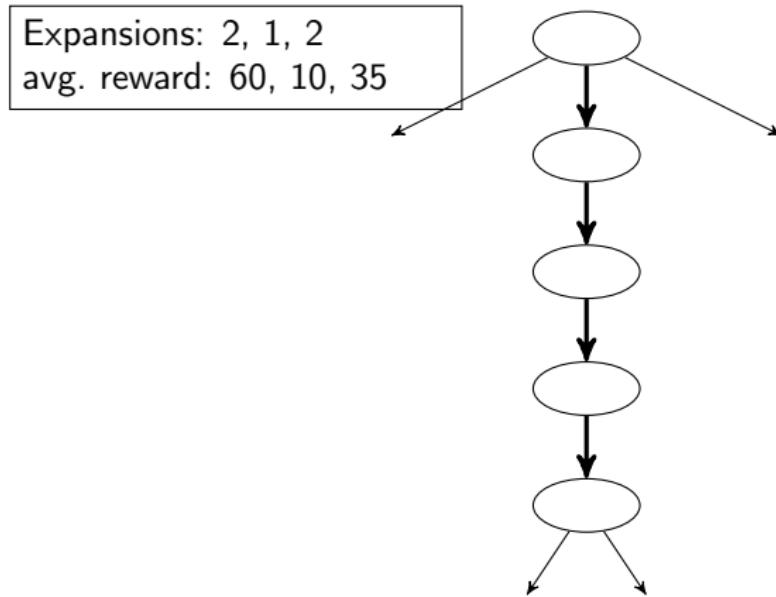
Monte-Carlo Sampling: Illustration of Sampling

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



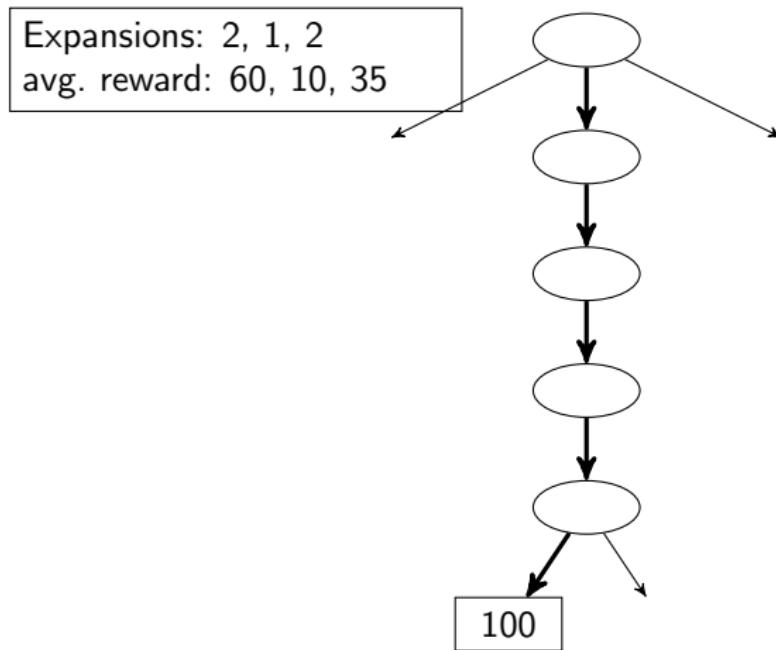
Monte-Carlo Sampling: Illustration of Sampling

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



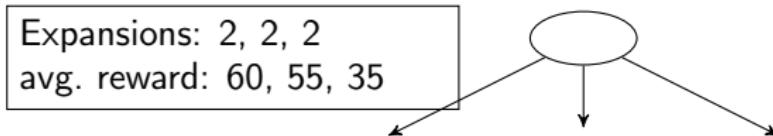
Monte-Carlo Sampling: Illustration of Sampling

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



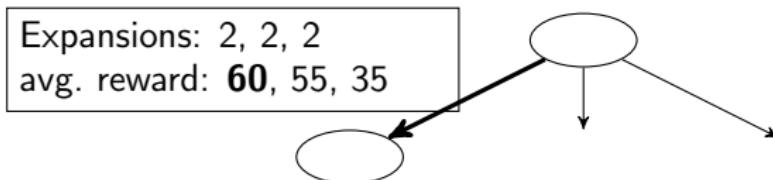
Monte-Carlo Sampling: Illustration of Sampling

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



Monte-Carlo Sampling: Illustration of Sampling

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



Monte-Carlo Sampling: Illustration of Sampling

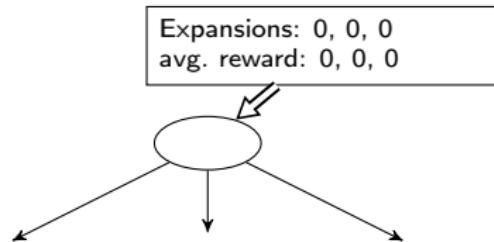
- ▶ **Idea:** Sample the search tree keeping track of the average utilities.
- ▶ **Example 5.3 (Single-player, for simplicity).** (with adversary, distinguish max/min nodes)



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

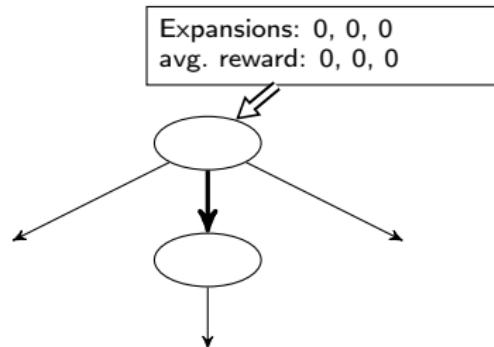
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



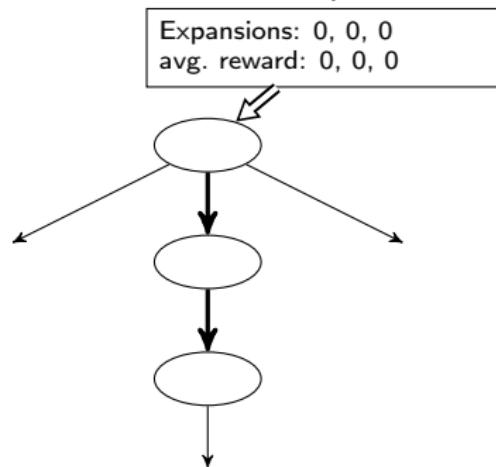
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



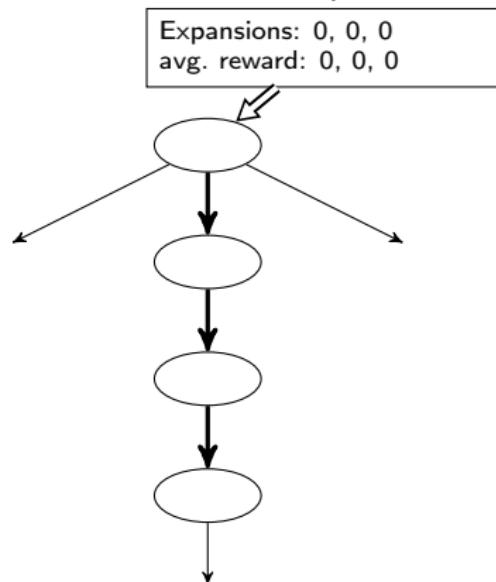
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



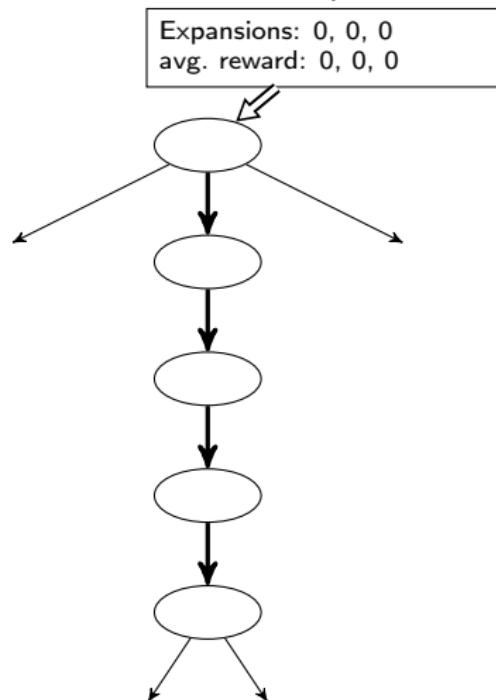
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



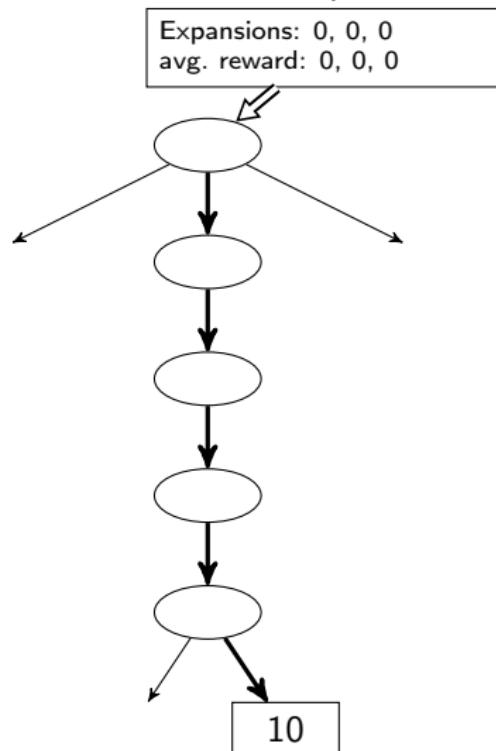
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



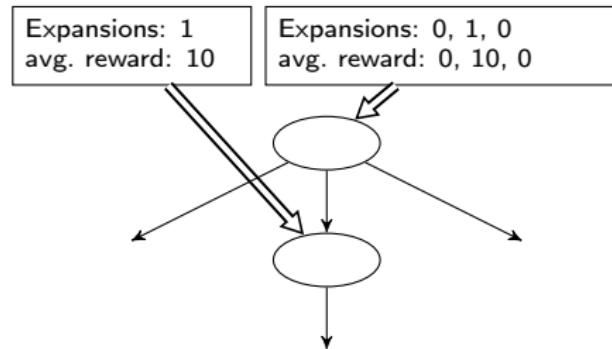
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



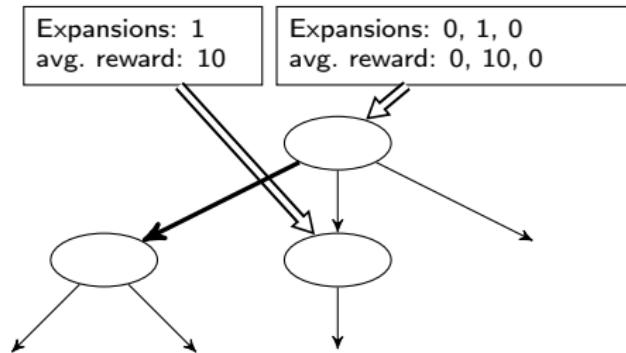
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



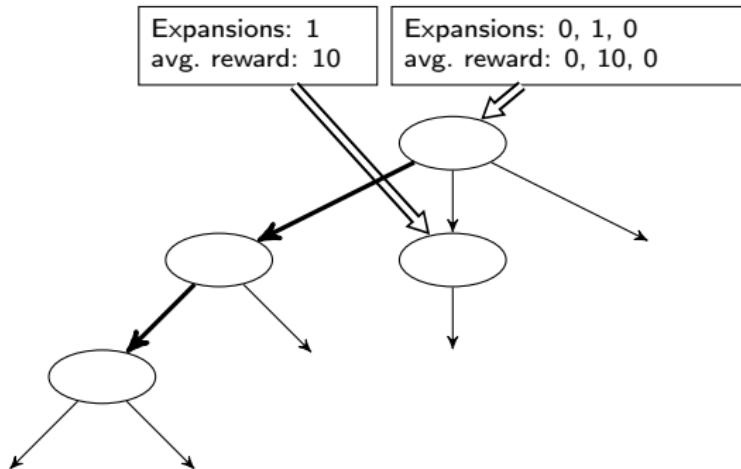
Monte-Carlo Tree Search: Building the Tree

- **Idea:** we can save work by building the tree as we go along
- **Example 5.4 (Redoing the previous example).**



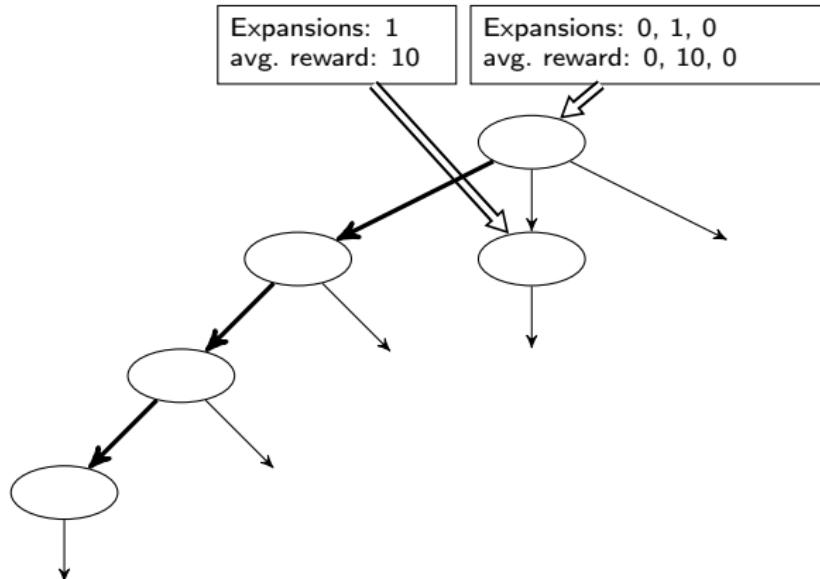
Monte-Carlo Tree Search: Building the Tree

- **Idea:** we can save work by building the tree as we go along
- **Example 5.4 (Redoing the previous example).**



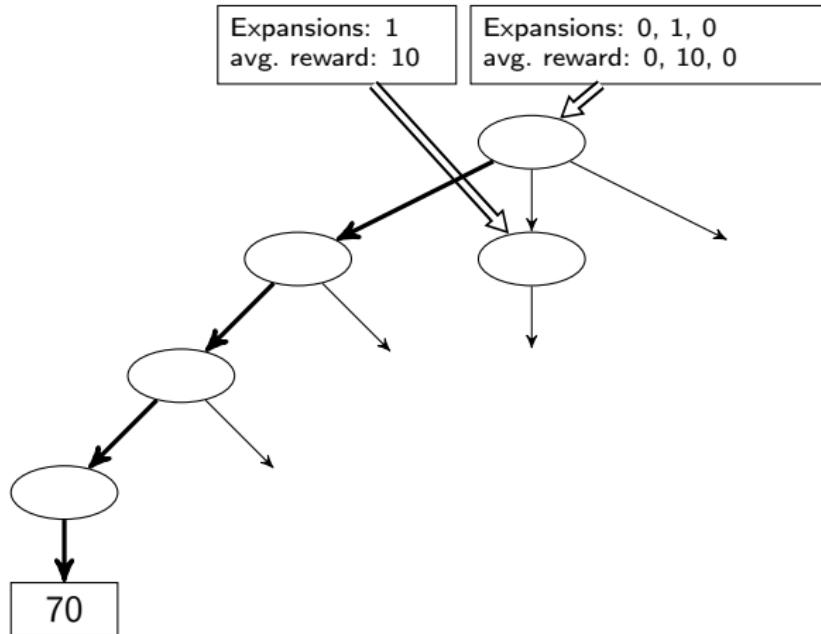
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
 - ▶ **Example 5.4 (Redoing the previous example).**



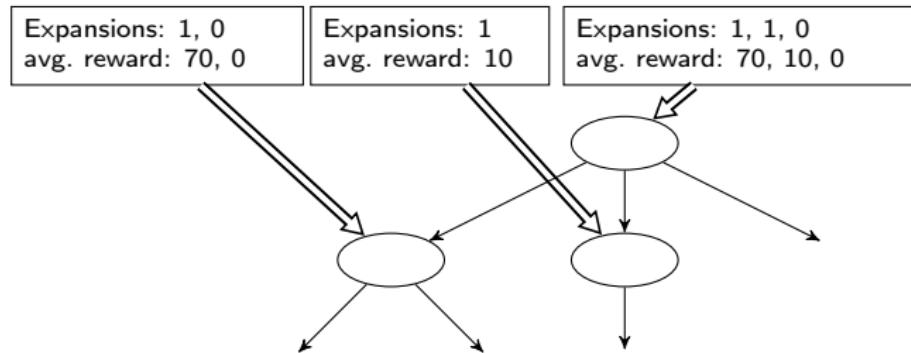
Monte-Carlo Tree Search: Building the Tree

- **Idea:** we can save work by building the tree as we go along
- **Example 5.4 (Redoing the previous example).**



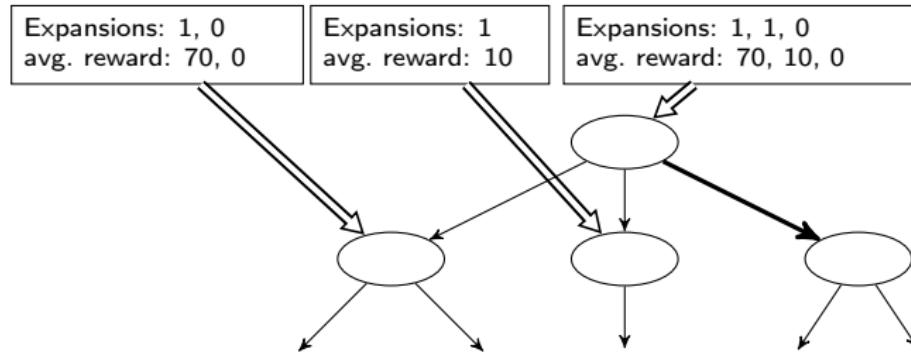
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



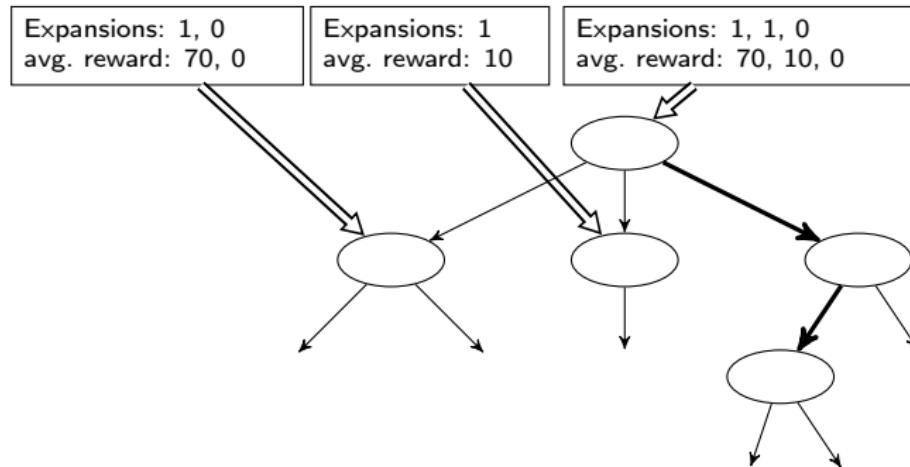
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



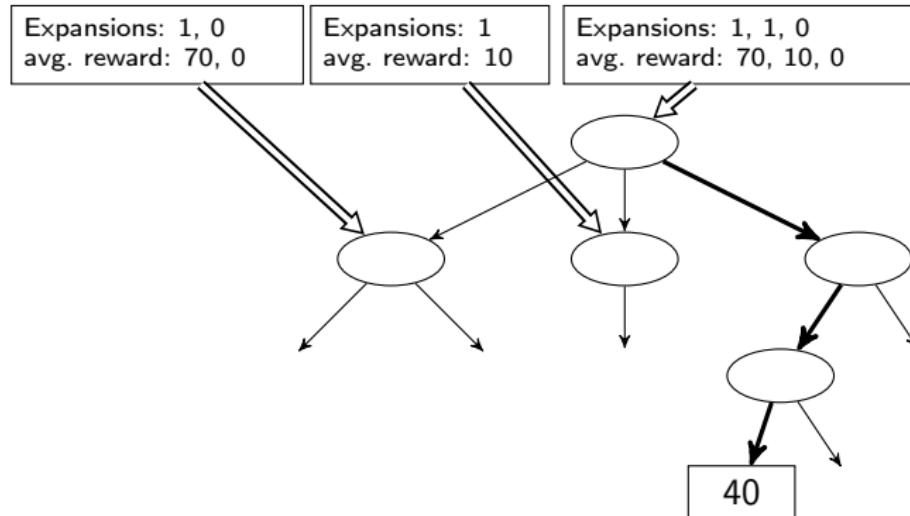
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



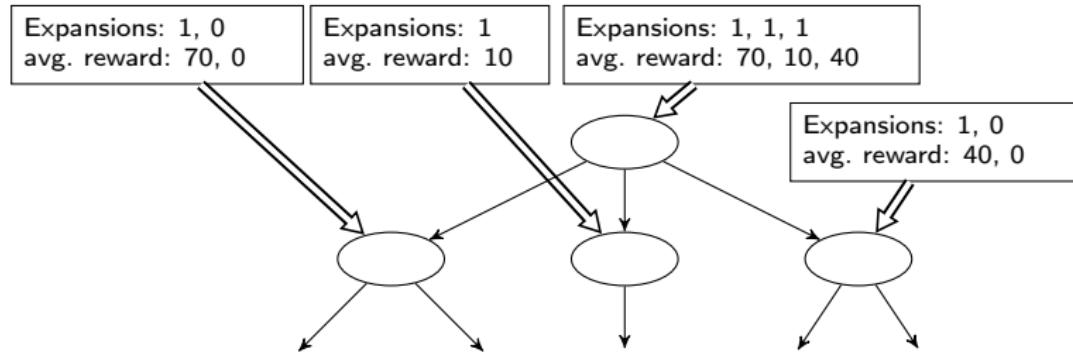
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



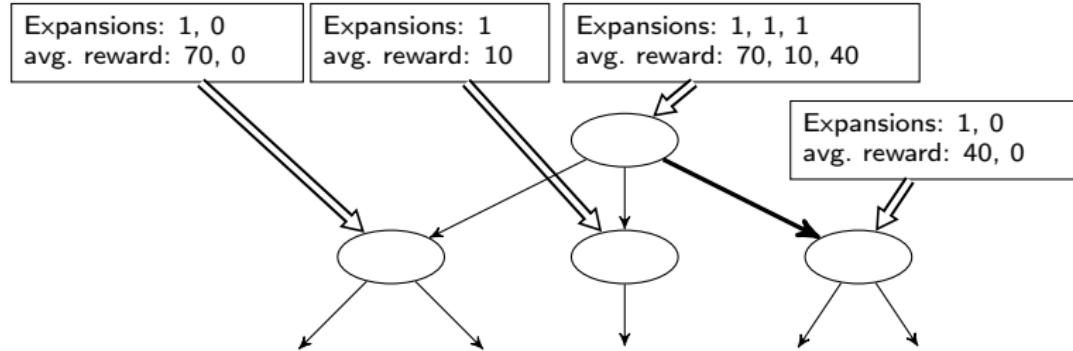
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



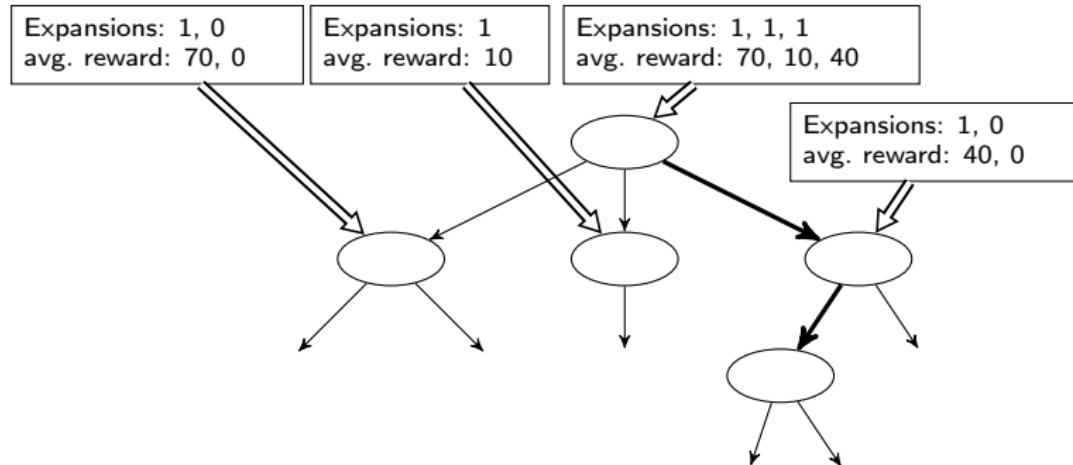
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



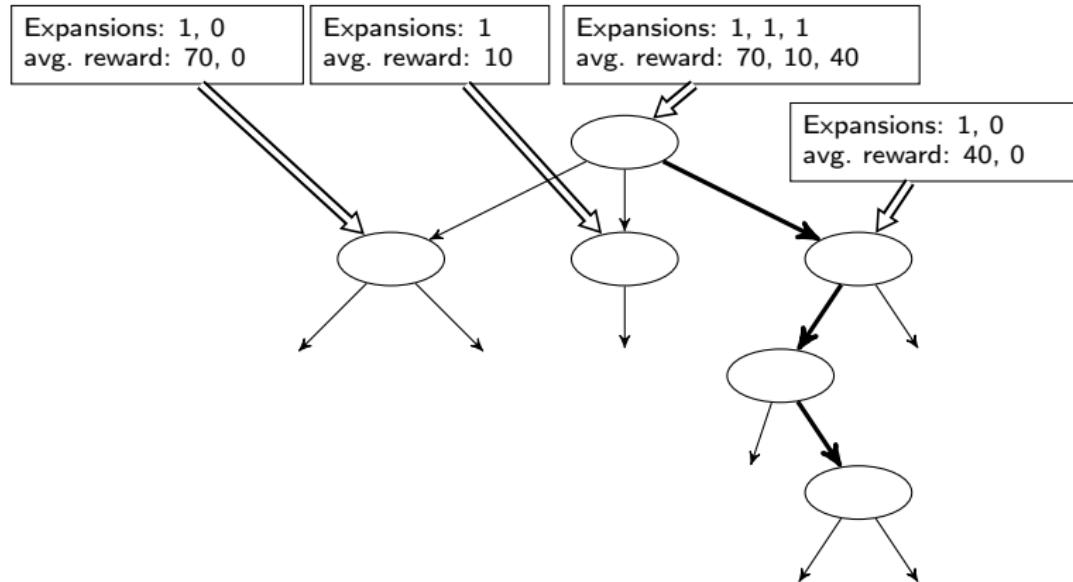
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



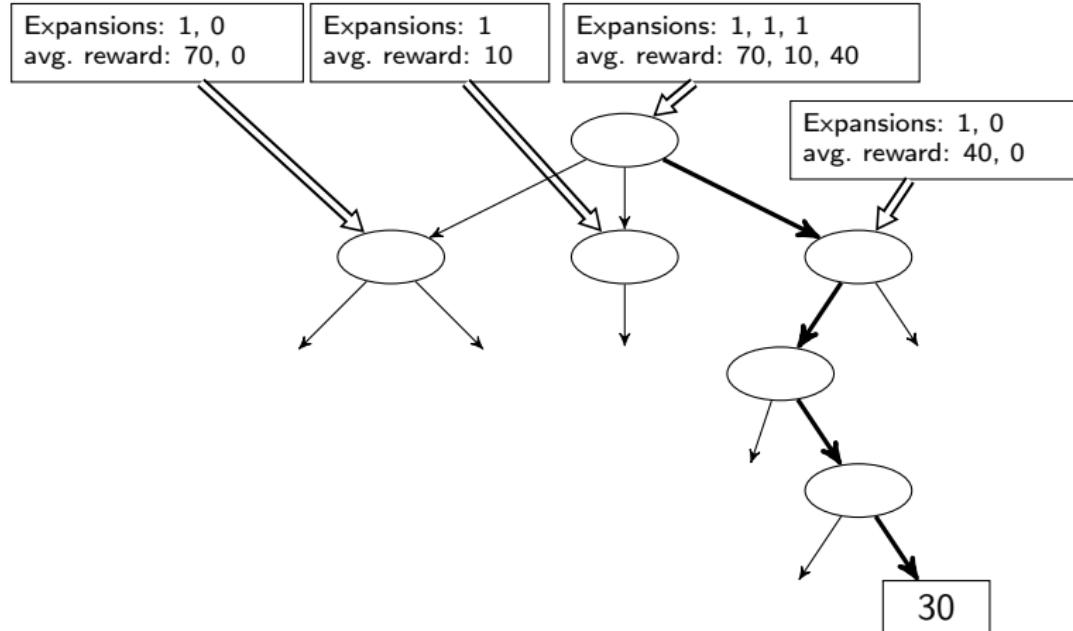
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



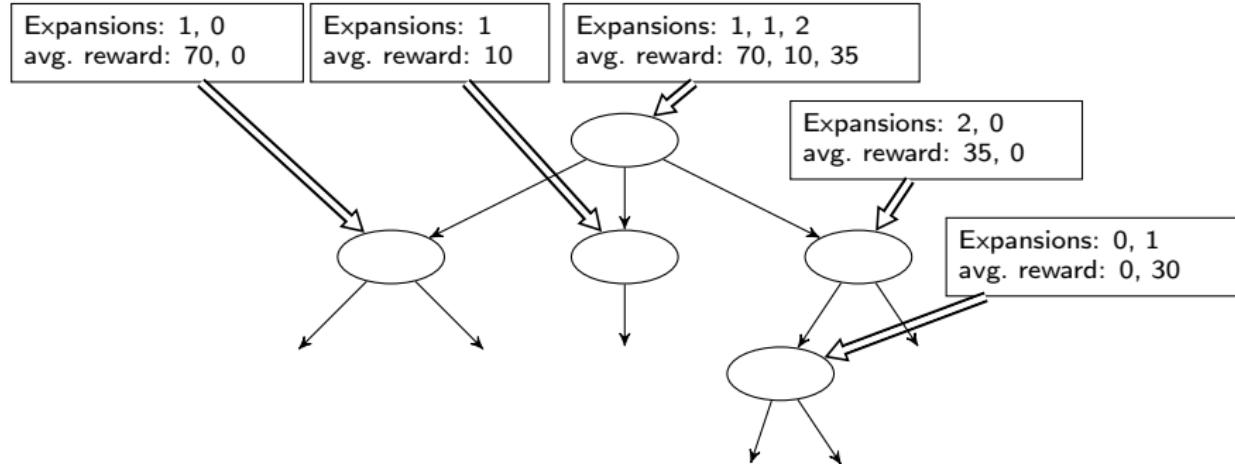
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



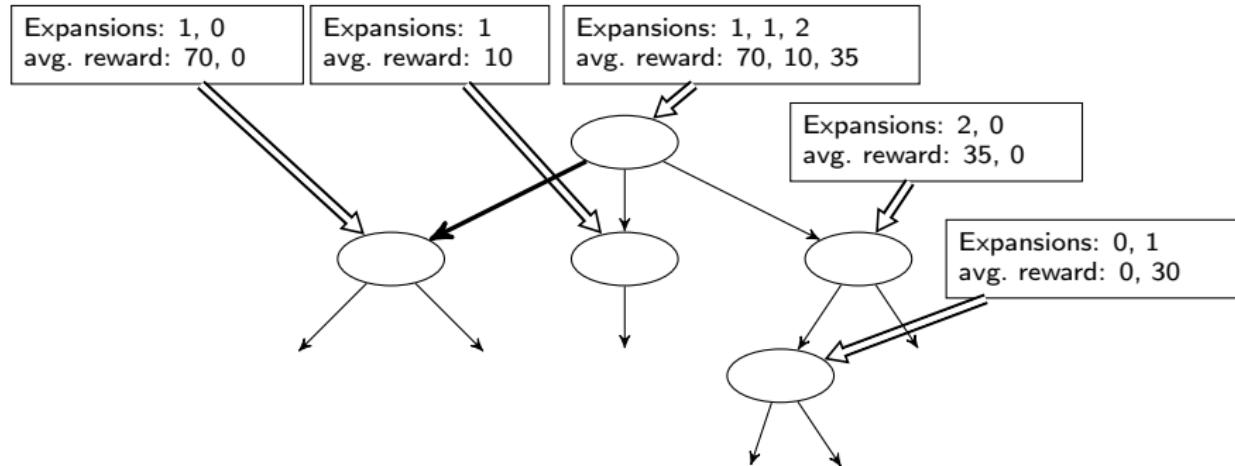
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



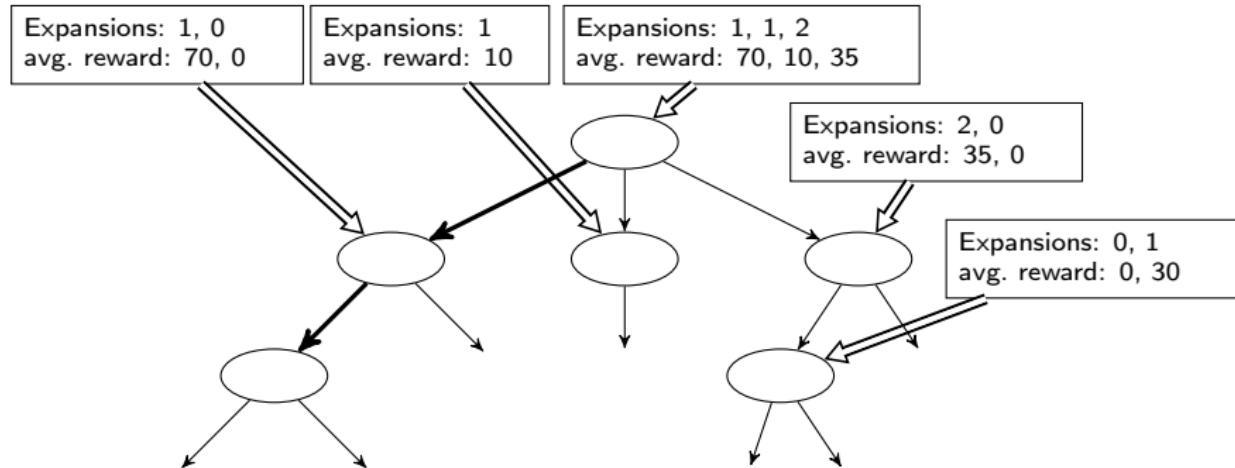
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



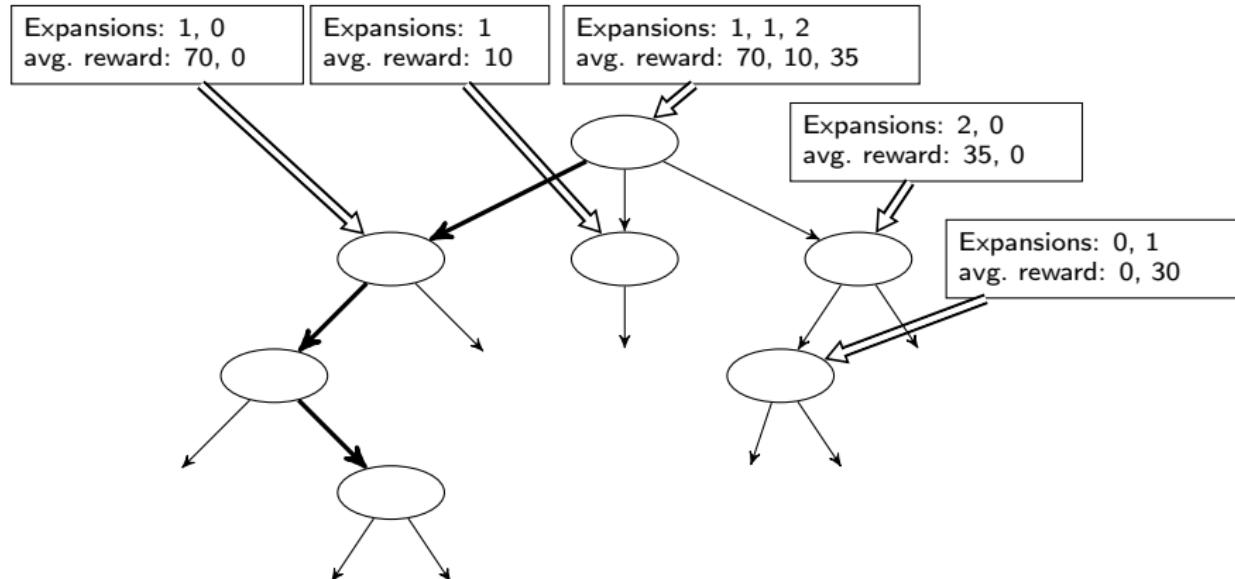
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



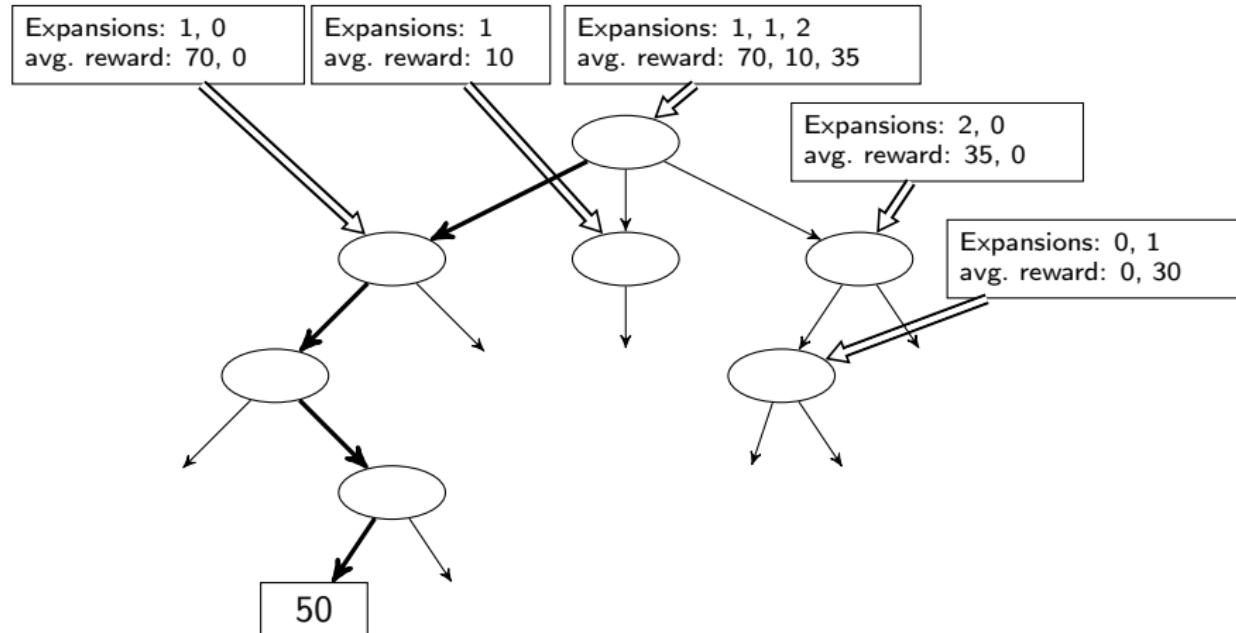
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



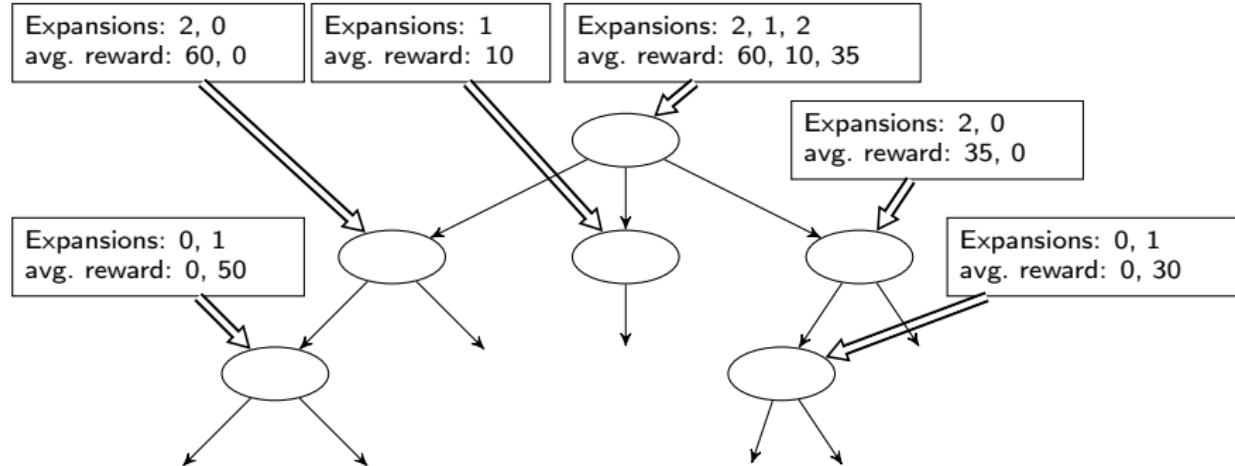
Monte-Carlo Tree Search: Building the Tree

- ▶ Idea: we can save work by building the tree as we go along
- ▶ Example 5.4 (Redoing the previous example).



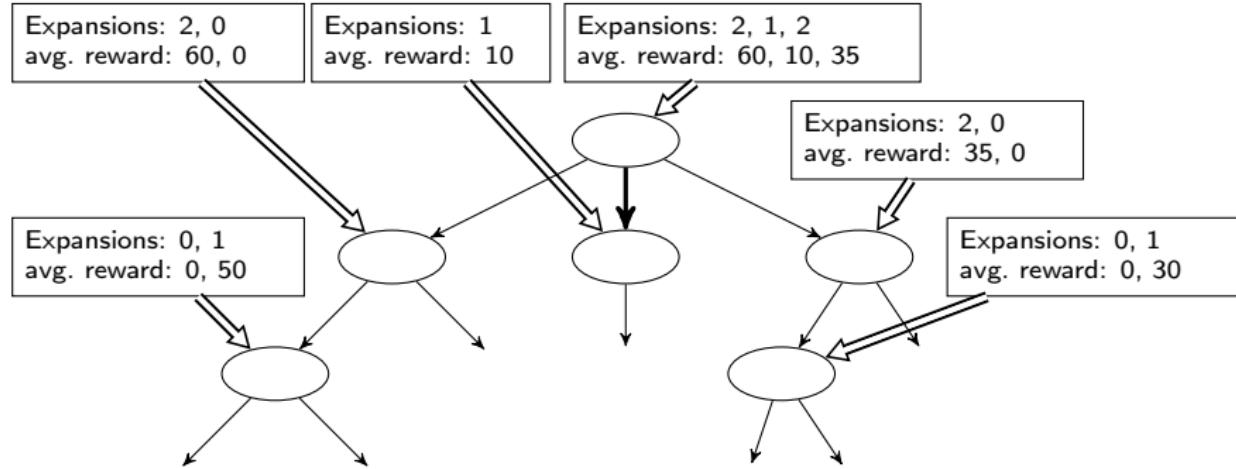
Monte-Carlo Tree Search: Building the Tree

- **Idea:** we can save work by building the tree as we go along
- **Example 5.4 (Redoing the previous example).**



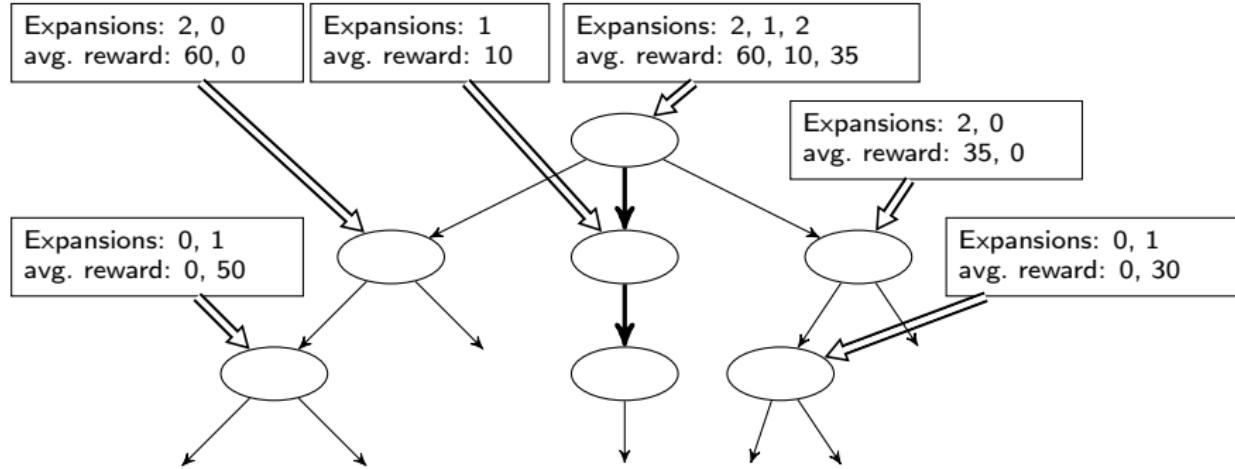
Monte-Carlo Tree Search: Building the Tree

- **Idea:** we can save work by building the tree as we go along
- **Example 5.4 (Redoing the previous example).**



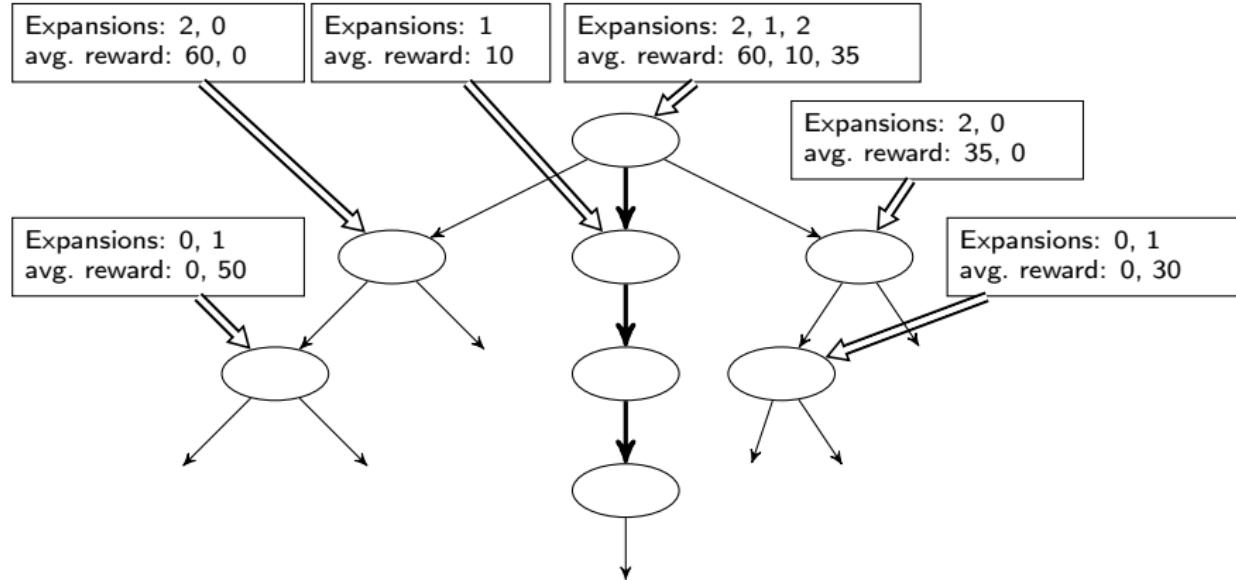
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
 - ▶ **Example 5.4 (Redoing the previous example).**



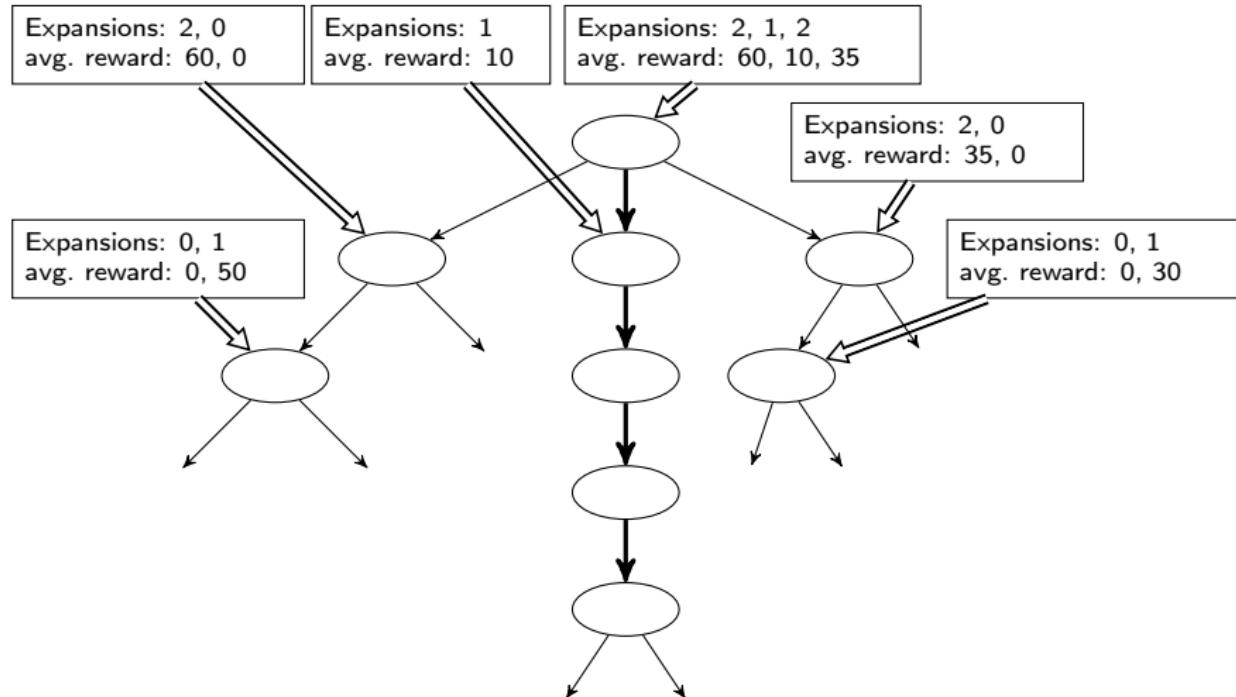
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
 - ▶ **Example 5.4 (Redoing the previous example).**



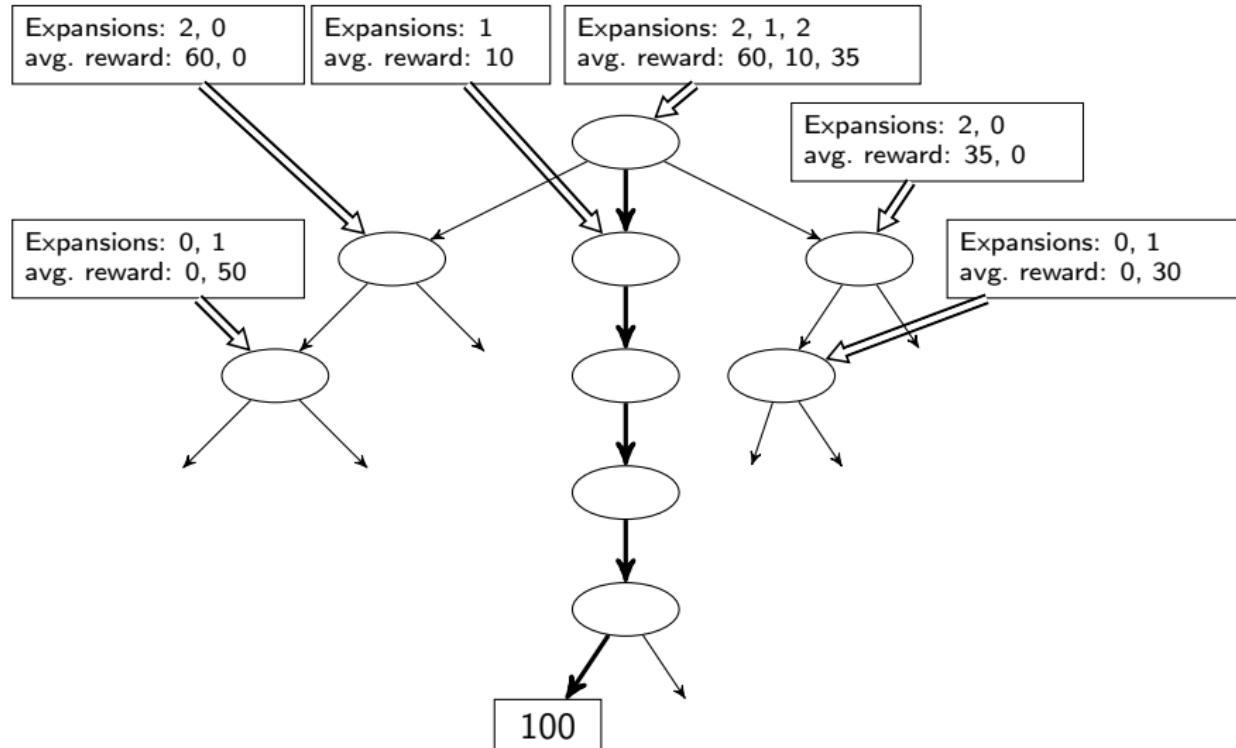
Monte-Carlo Tree Search: Building the Tree

- **Idea:** we can save work by building the tree as we go along
- **Example 5.4 (Redoing the previous example).**



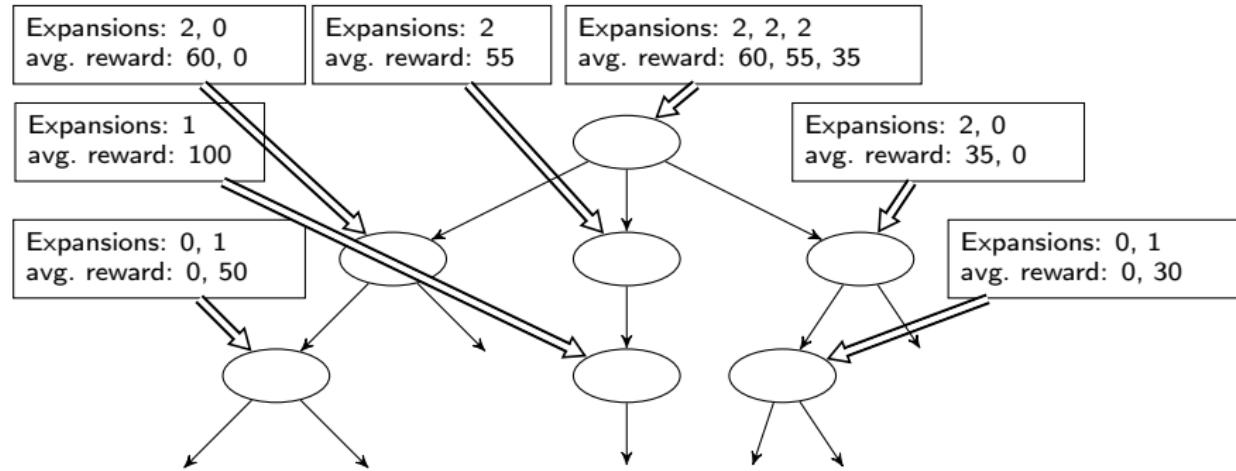
Monte-Carlo Tree Search: Building the Tree

- **Idea:** we can save work by building the tree as we go along
- **Example 5.4 (Redoing the previous example).**



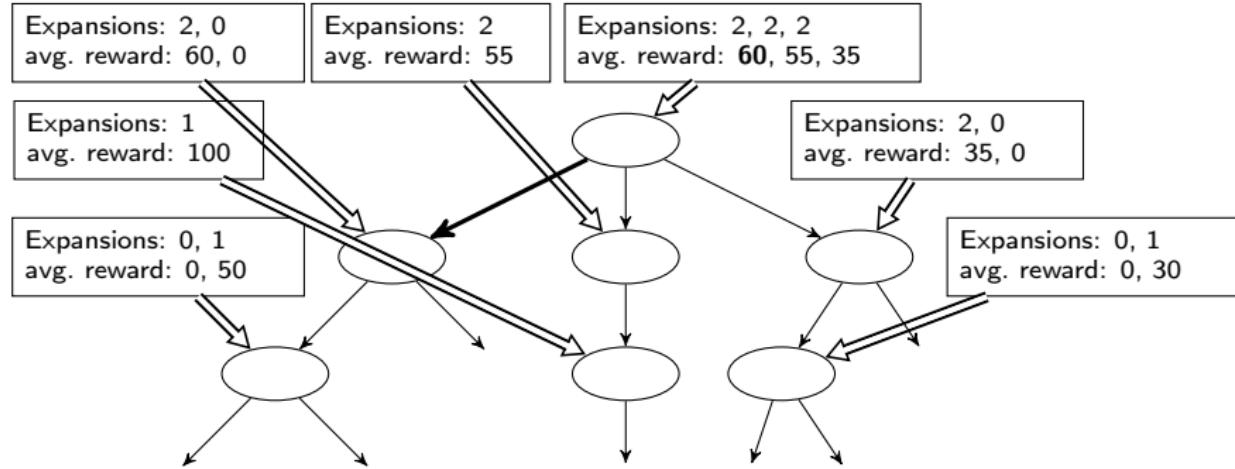
Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



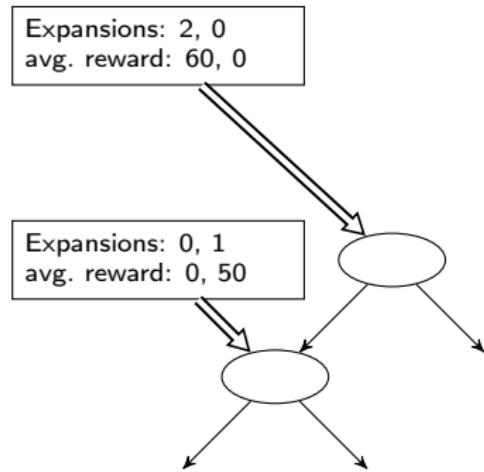
Monte-Carlo Tree Search: Building the Tree

- Idea: we can save work by building the tree as we go along
- Example 5.4 (Redoing the previous example).



Monte-Carlo Tree Search: Building the Tree

- ▶ **Idea:** we can save work by building the tree as we go along
- ▶ **Example 5.4 (Redoing the previous example).**



How to Guide the Search in MCTS?

- ▶ How to “sample”: What exactly is “random”?
- ▶ Classical formulation: balance exploitation vs. exploration.
 - ▶ Exploitation: Prefer moves that have high average already (interesting regions of state space).
 - ▶ Exploration: Prefer moves that have not been tried a lot yet (don’t overlook other, possibly better, options).
- ▶ UCT: “Upper Confidence bounds applied to Trees” [KS06].
 - ▶ Inspired by Multi-Armed Bandit (as in: Casino) problems.
 - ▶ Basically a formula defining the balance. Very popular (buzzword).
 - ▶ Recent critics (e.g. [FD14]): “Exploitation” in search is very different from the Casino, as the “accumulated rewards” are fictitious (we’re only thinking about the game, not actually playing and winning/losing all the time).

- ▶ Neural Networks:
 - ▶ 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 *AlphaGoSystem*

► Neural network training pipeline and architecture:

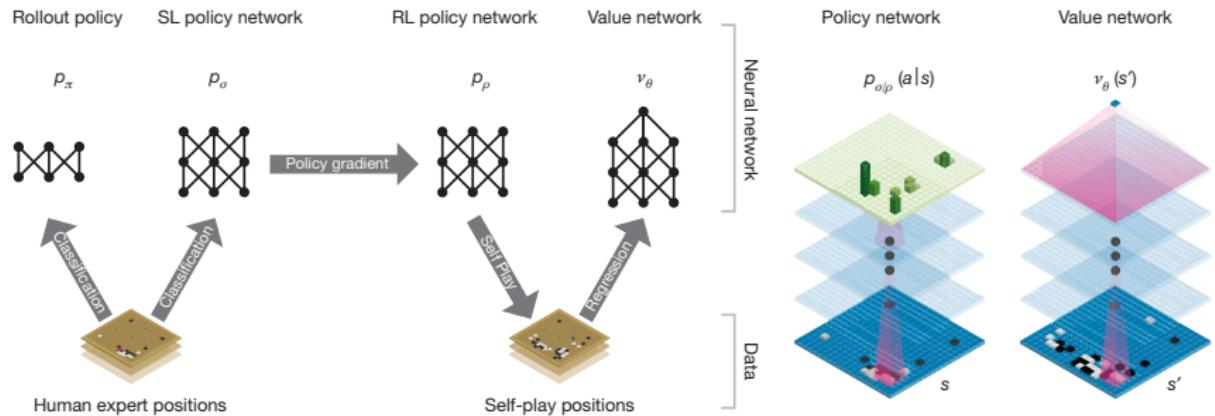


Illustration taken from [Sil+16].

- **Rollout policy p_π :** Simple but fast, \approx 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").

Neural Networks + MCTS in AlphaGo

► Monte Carlo tree search in AlphaGo:

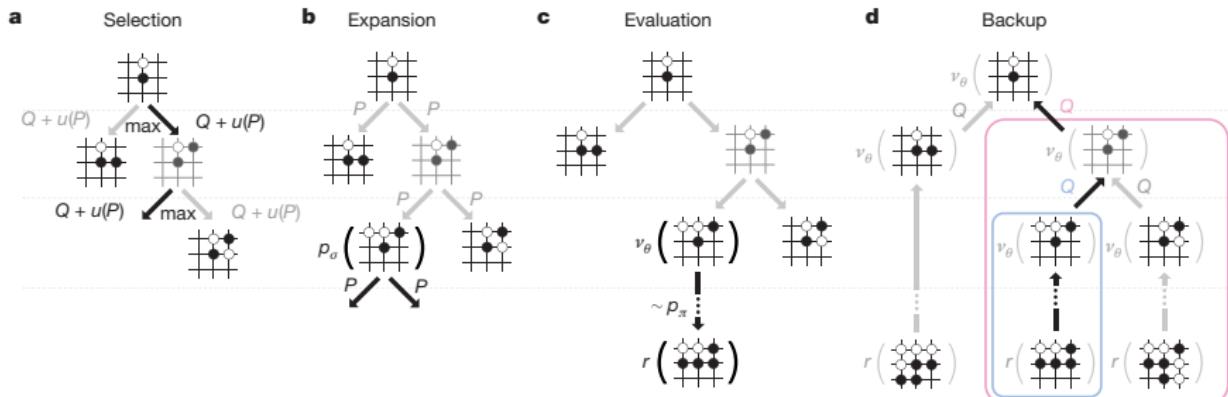


Illustration taken from [Sil+16]

- *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 .

6 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 official 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.
- Board Games are considered a “solved problem” from the AI perspective.

Computer Chess: “Deep Blue” beat Garry Kasparov in 1997



- ▶ 6 games, final score 3.5 : 2.5.
- ▶ Specialized Chess hardware, 30 nodes with 16 processors each.
- ▶ Alpha-beta search plus human knowledge (more details in a moment).
- ▶ Nowadays, standard PC hardware plays at world champion level.

- ▶ The chess machine is an ideal one to start with, since ([Claude Shannon \(1949\)](#))
 1. the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate),
 2. it is neither so simple as to be trivial nor too difficult for satisfactory solution,
 3. chess is generally considered to require “thinking” for skilful play, [...]
 4. the discrete structure of chess fits well into the digital nature of modern computers.
- ▶ Chess is the drosophila of Artificial Intelligence. ([Alexander Kronrod \(1965\)](#))

Computer Chess: Another Famous Quote

- ▶ In 1965, the Russian mathematician Alexander Kronrod said, “Chess is the Drosophila of artificial intelligence.”
However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophilae. We would have some science, but mainly we would have very fast fruit flies.(John McCarthy (1997))

7 Conclusion

- ▶ Games (2-player turn-taking zero-sum discrete and finite games) can be understood as a simple extension of classical search problems.
- ▶ Each player tries to reach a **terminal state** with the best possible **utility** (maximal vs. minimal).
- ▶ **Minimax** searches the game depth-first, max'ing and min'ing at the respective turns of each player. It yields perfect play, but takes time $\mathcal{O}(b^d)$ where b is the branching factor and d the search depth.
- ▶ Except in trivial games (Tic-Tac-Toe), Minimax needs a **depth limit** and apply an **evaluation function** to estimate the value of the cut-off states.
- ▶ **Alpha-beta search** remembers the best values achieved for each player elsewhere in the tree already, and prunes out sub-trees that won't be reached in the game.
- ▶ **Monte-Carlo tree search (MCTS)** samples game branches, and averages the findings. **AlphaGo** controls this using **neural networks**: evaluation function ("value network"), and action filter ("policy network").

Chapter 9 Constraint Satisfaction Problems

1 Constraint Satisfaction Problems: Motivation

A (Constraint Satisfaction) Problem

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

Fußballkalender

Saison 2012/2013



Juli 2012		August 2012		September 2012		Oktober 2012		November 2012		Dezember 2012		Januar 2013		Februar 2013		März 2013		April 2013		Mai 2013		Juni 2013	
1		1	1	1		1		1		1		1		1		1		1		1		1	
2		2		2		2		2		2		2		2		2		2		2		2	
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4		4		4		4		4		4		4		4		4		4		4		4	
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6		6		6		6		6		6		6		6		6		6		6		6	
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8		8		8		8		8		8		8		8		8		8		8		8	
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
11		11		11		11		11		11		11		11		11		11		11		11	
12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
13		13		13		13		13		13		13		13		13		13		13		13	
14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
16		16		16		16		16		16		16		16		16		16		16		16	
17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
18		18		18		18		18		18		18		18		18		18		18		18	
19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19
20		20		20		20		20		20		20		20		20		20		20		20	
21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21
22		22		22		22		22		22		22		22		22		22		22		22	
23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23
24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24
25		25		25		25		25		25		25		25		25		25		25		25	
26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26
27		27		27		27		27		27		27		27		27		27		27		27	
28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28
29		29		29		29		29		29		29		29		29		29		29		29	
30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
31		31		31		31		31		31		31		31		31		31		31		31	
DFB-Pokal, Champions-League, Europa-League, Länderspiele																							

Constraint satisfaction problems (CSPs)

- ▶ Standard search problem: **state** is a “black box”—any old data structure that supports goal test, eval, successor
- ▶ **Definition 1.2.** A **constraint satisfaction problem (CSP)** is a **search problem**, where the **states** are given by a **finite set** $V := \{X_1, \dots, X_n\}$ of **variables** and **domains** $\{D_v | v \in V\}$ and the **goal test** is a set of **constraints** specifying allowable combinations of values for subsets of variables.
- ▶ **Definition 1.3.** A **CSP** is **satisfiable**, iff it has a **solution** – a function from **variables** v to elements of their domains D_v that satisfy all **constraints**.
- ▶ **Remark 1.4.** We are using **factored** representation for world states now.
- ▶ Simple example of a *formal representation language*
- ▶ Allows useful *general-purpose* algorithms with more power than standard **tree search algorithm**.

Another Constraint Satisfaction Problem

- **Example 1.5 (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	

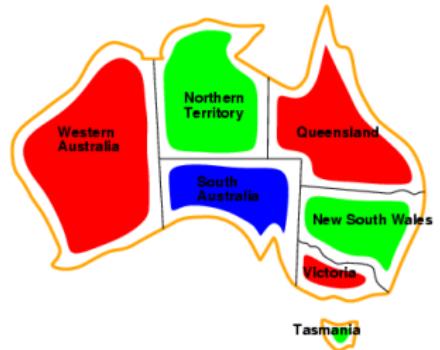
2	5	8	7	3	6	9	4	1
6	1	9	8	2	4	3	5	7
4	3	7	9	1	5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

~

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

CSP Example: Map-Coloring

- ▶ **Definition 1.6.** Given a map M , the **map coloring** problem is to assign colors to regions in a map so that no adjoining regions have the same color.
- ▶ **Example 1.7 (Map coloring in Australia).**



- ▶ **Variables:** WA, NT, Q, NSW, V, SA, T
- ▶ **Domains:** $D_i = \{\text{red}, \text{green}, \text{blue}\}$
- ▶ **Constraints:** adjacent regions must have different colors e.g., $\text{WA} \neq \text{NT}$ (if the language allows this), or $\langle \text{WA}, \text{NT} \rangle \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), (\text{green}, \text{red}), \dots\}$
- ▶ **Intuition:** Solutions map variables to domain values (assignments) satisfying all constraints,
- ▶ e.g., $\{\text{WA} = \text{red}, \text{NT} = \text{green}, \dots\}$

Bundesliga Constraints

- ▶ **Variables:** $v_{A\text{vs.}B}$ where A and B are teams, with **domains** $\{1, \dots, 34\}$: 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 runtime: End of this universe, and the next couple billion ones after it ...
- ▶ Directly enumerate all **permutations** of the numbers $1, \dots, 306$, test for each whether it's a legal Bundesliga schedule.
 - ▶ Estimated runtime: Maybe only the time span of a few thousand universes.
- ▶ View this as **variables/constraints** and use backtracking (this chapter)
 - ▶ Executed runtime: About 1 minute.
- ▶ How do they actually do it?: Modern computers and **CSP** methods: fractions of a second. 19th (20th/21st?) century: Combinatorics and manual work.
- ▶ Try it yourself: with an off-the shelf **CSP** solver, e.g. Minion [Min]

More Constraint Satisfaction Problems

Traveling Tournament Problem



Scheduling



Timetabling



Radio Frequency Assignment



Our Agenda for This Topic

- ▶ Our treatment of the topic “Constraint Satisfaction Problems” consists of Chapters 7 and 8. in [RN03]
- ▶ **This Chapter:** Basic definitions and concepts; naïve backtracking search.
 - ▶ Sets up the framework. Backtracking underlies many successful algorithms for solving constraint satisfaction problems (and, naturally, we start with the simplest version thereof).
- ▶ **Next Chapter:** Inference and decomposition methods.
 - ▶ Inference 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

- ▶ **Constraint networks** and **Assignments, Consistency, Solutions**: How are constraint satisfaction problems defined? What is a solution?
- ▶ Get ourselves on firm ground.

Our Agenda for This Chapter

- ▶ **Constraint networks** and **Assignments, Consistency, Solutions**: How are constraint satisfaction problems defined? What is a solution?
 - ▶ Get ourselves on firm ground.
- ▶ **Naïve Backtracking**: How does backtracking work? What are its main weaknesses?
 - ▶ Serves to understand the basic workings of this wide-spread algorithm, and to motivate its enhancements.

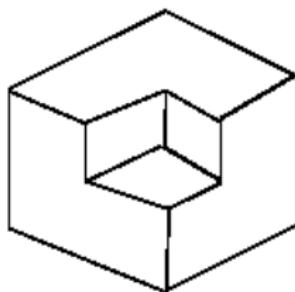
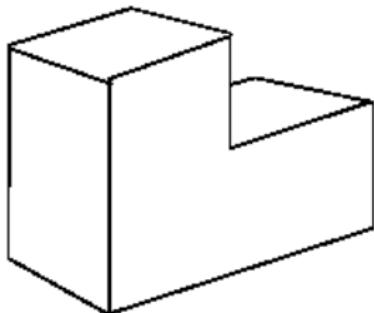
Our Agenda for This Chapter

- ▶ **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 search?
 - ▶ Simple methods for making backtracking aware of the structure of the problem, and thereby reduce search.

2 The Waltz Algorithm

The Waltz Algorithm

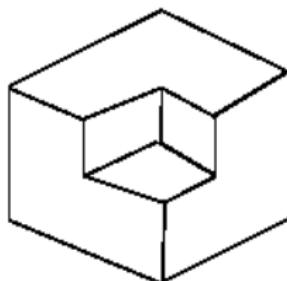
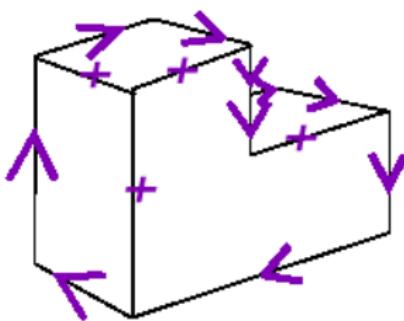
- ▶ One of the earliest examples of applied CSPs
- ▶ Motivation: interpret line drawings of polyhedra



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

Waltz Algorithm on Simple Scenes

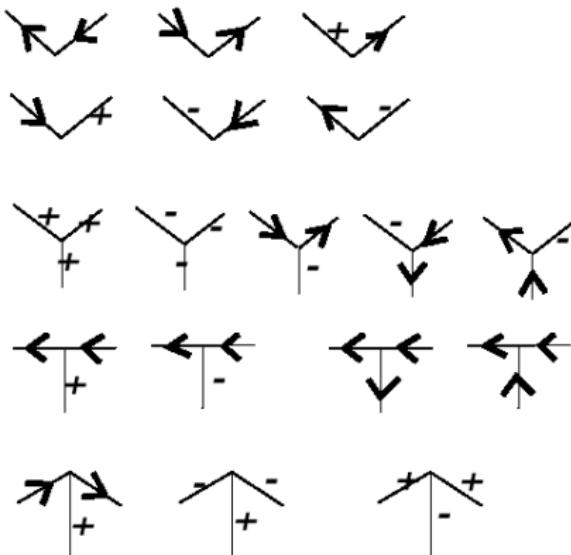
- ▶ **Assumptions:** All objects
 - ▶ have no shadows or cracks,
 - ▶ have only three-faced vertices,
 - ▶ are in “general position”, i.e. no junctions change with small movements of the eye.
 - ▶ **Observation 2.1.** Then each line on the images is one of the following:
 - ▶ a boundary line (edge of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



(label with "+")
(label with "-")

18 Legal Kinds of Junctions

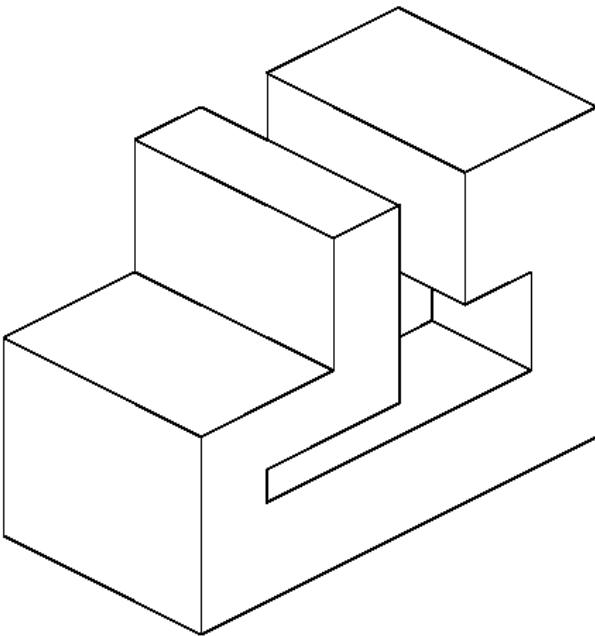
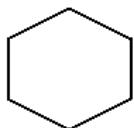
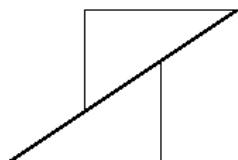
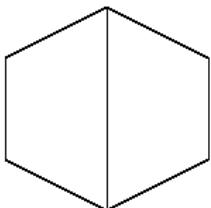
- ***Observation 2.2.*** There are only 18 “legal” kinds of junctions:



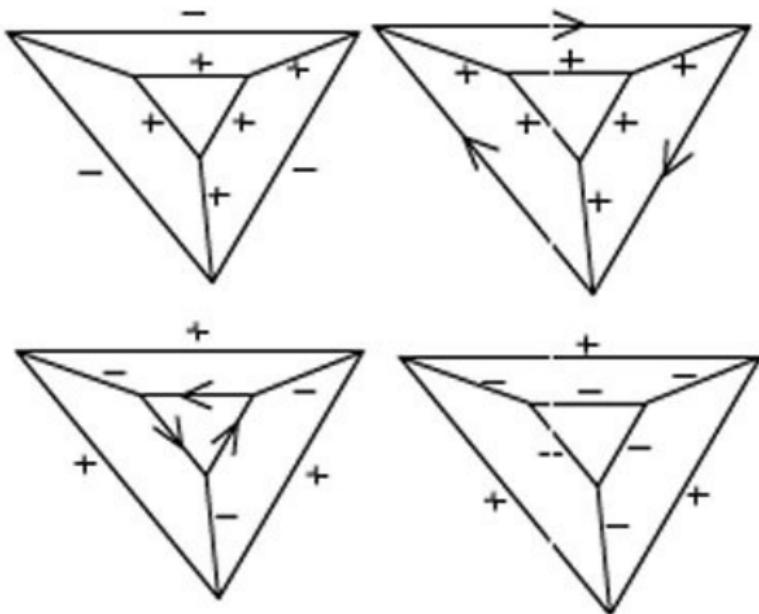
- **Idea:** given a representation of a diagram
- label each junction in one of these manners (lots of possible ways)
 - junctions must be labeled, so that lines are labeled consistently
- **Fun Fact:** CSP always works perfectly! (early success story for CSP [Wal75])

Waltz's Examples

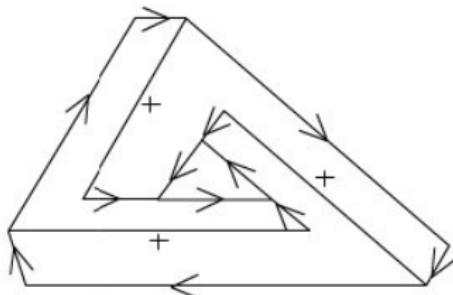
- In his dissertation 1972 [Wal75] David Waltz used the following examples



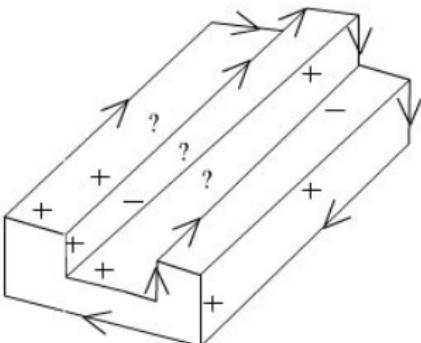
Waltz Algorithm (More Examples): Ambiguous Figures



Waltz Algorithm (More Examples): Impossible Figures



Consistent labelling for impossible figure



No consistent labelling possible

- ▶ **Definition 3.1.** We call a **CSP discrete**, iff all of the **variables** have **countable domains**; we have two
 - ▶ **finite domains** (size $d \sim \mathcal{O}(d^n)$ solutions)
 - ▶ e.g., Boolean CSPs (solvability $\hat{=}$ Boolean satisfiability \sim NP complete)
 - ▶ infinite domains (e.g. integers, strings, etc.)
 - ▶ e.g., job scheduling, variables are start/end days for each job
 - ▶ need a "constraint language", e.g., $StartJob_1 + 5 \leq StartJob_3$
 - ▶ linear constraints **decidable**, nonlinear ones **undecidable**
- ▶ **Definition 3.2.** We call a **CSP continuous**, iff one **domain** is **uncountable**.
- ▶ **Example 3.3.** start/end times for Hubble Telescope observations
- ▶ linear constraints solvable in poly time by linear programming methods
- ▶ there cannot be optimal algorithms for nonlinear constraint systems

- ▶ We classify the **constraints** by the number of **variables** they involve.
- ▶ **Definition 3.4.** **Unary constraints** involve a single variable, e.g., $SA \neq green$.
- ▶ **Definition 3.5.** **Binary constraints** involve pairs of variables, e.g., $SA \neq WA$.
- ▶ **Definition 3.6.** **Higher order constraints** involve $n = 3$ or more **variables**, e.g., cryptarithmetic column constraints.
The number n of **variables** is called the **order** of the **constraint**.
- ▶ **Definition 3.7.** **Preferences (soft constraints)** (e.g., red is better than green) are often representable by a cost for each variable assignment \leadsto **constrained optimization problems**.

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

- **Example 3.8 (Send More Money).** A student writes home:

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

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

- **Variables:** S, E, N, D, M, O, R, Y with domain $\{0, \dots, 9\}$
► **Constraints:**

1. all variables should have different values: $S \neq E, S \neq N, \dots$
2. first digits are non-zero: $S \neq 0, M \neq 0$.
3. the addition scheme should work out: i.e.

$$1000 \cdot S + 100 \cdot E + 10 \cdot N + D + 1000 \cdot M + 100 \cdot O + 10 \cdot R + E = \\ 10000 \cdot M + 1000 \cdot O + 100 \cdot N + 10 \cdot E + Y.$$

BTW: The solution is

$S \mapsto 9, E \mapsto 5, N \mapsto 6, D \mapsto 7, M \mapsto 1, O \mapsto 0, R \mapsto 8, Y \mapsto 2 \rightsquigarrow$ parents send 10652€

- **Definition 3.9.** Problems like the one in 3.8 are called **crypto arithmetic puzzles**.

Encoding Higher-Order Constraints as Binary ones

- ▶ Problem: The last constraint is of order 8. ($n = 8$ variables involved)
- ▶ **Observation 3.10.** We can write the addition scheme constraint column-wise using auxiliary variables, i.e. variables that do not “occur” in the original problem.

$$\begin{array}{rcl} D+E & = & Y+10 \cdot X_1 \\ X_1+N+R & = & E+10 \cdot X_2 \\ X_2+E+O & = & N+10 \cdot X_3 \\ X_3+S+M & = & O+10 \cdot M \end{array} \quad \begin{array}{r} S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

These constraints are of order ≤ 5 .

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

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

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

Constraint Graph

- ▶ **Definition 3.11.** A **binary CSP** is a **CSP** where each **constraint** is **binary**.
- ▶ **Observation 3.12.** A **binary CSP** forms a **graph** called the **constraint graph** whose **nodes** are **variables**, and whose **edges** represent the **constraints**.
- ▶ **Example 3.13.** 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!)

- ▶ **Example 3.14 (Assignment problems).** (e.g., who teaches what class)
- ▶ **Example 3.15 (Timetabling problems).** (e.g., which class is offered when and where?)
- ▶ **Example 3.16 (Hardware configuration).**
- ▶ **Example 3.17 (Spreadsheets).**
- ▶ **Example 3.18 (Transportation scheduling).**
- ▶ **Example 3.19 (Factory scheduling).**
- ▶ **Example 3.20 (Floorplanning).**
- ▶ **Note:** many real-world problems involve real-valued variables \leadsto continuous CSPs.

- ▶ **Definition 3.21.** A **constraint network** is a triple $\gamma := \langle V, D, C \rangle$, where
 - ▶ $V := \{X_1, \dots, X_n\}$ is a finite set of **variables**,
 - ▶ $D := \{D_v | v \in V\}$ the set of their **domains**, and
 - ▶ $C := \{C_{uv} | u, v \in V \text{ and } u \neq v\}$, where a (binary) **constraint** C_{uv} is a **relation** between D_u and D_v ($C_{uv} \subseteq D_u \times D_v$) and $C_{uv} = C_{vu}^{-1}$.
- ▶ $\langle V, C \rangle$ form a **graph**, we call it the **constraint graph** of γ .
- ▶ **Observation 3.22.** Binary CSPs can be formulated as **constraint networks**.
- ▶ **Remarks:** The mathematical formulation gives us a lot of leverage:
 - ▶ $C_{uv} \subseteq D_u \times D_v \hat{=} \text{possible assignments to variables } u \text{ and } v$
 - ▶ **relations** are most general formalization, generally we use symbolic formulations, e.g. “ $u = v$ ” for the **relation** $C_{uv} = \{(a, b) | a = b\}$ or “ $u \neq v$ ”.
 - ▶ We can express **unary** constraints C_v by restricting the **domain** of v : $D_v := C_v$.

Example: SuDoKu as a Constraint Network

- ▶ **Example 3.23 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as in 1.5.

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

- ▶ **Variables:**

Note that the ideas are still the same as in 1.5, but in constraint networks we have a language to formulate things precisely.

Example: SuDoKu as a Constraint Network

- **Example 3.23 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as in 1.5.

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

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

Note that the ideas are still the same as in 1.5, but in constraint networks we have a language to formulate things precisely.

Example: SuDoKu as a Constraint Network

- **Example 3.23 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as in 1.5.

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

- **Variables:** $V = \{v_{ij} | 1 \leq i, j \leq 9\}$: v_{ij} = cell row i column j .
- **Domains** For all $v \in V$: $D_v = D = \{1, \dots, 9\}$.
- **Unary** constraints:

Note that the ideas are still the same as in 1.5, but in constraint networks we have a language to formulate things precisely.

Example: SuDoKu as a Constraint Network

- **Example 3.23 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as in 1.5.

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

- **Variables:** $V = \{v_{ij} | 1 \leq i, j \leq 9\}$: v_{ij} = cell row i column j .
- **Domains** For all $v \in V$: $D_v = D = \{1, \dots, 9\}$.
- **Unary constraints:** $C_{v_{ij}} = \{d\}$ if cell i, j is pre-filled with d .
- **Binary constraint:**

Note that the ideas are still the same as in 1.5, but in constraint networks we have a language to formulate things precisely.

Example: SuDoKu as a Constraint Network

- **Example 3.23 (Formalize SuDoKu).** We use the added formality to encode SuDoKu as a constraint network, not just as a CSP as in 1.5.

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

- **Variables:** $V = \{v_{ij} | 1 \leq i, j \leq 9\}$: v_{ij} = cell row i column j .
- **Domains** For all $v \in V$: $D_v = D = \{1, \dots, 9\}$.
- **Unary constraints:** $C_{v_{ij}} = \{d\}$ if cell i, j is pre-filled with d .
- **Binary constraint:** $C_{v_{ij} v_{i'j'}} \hat{=} "v_{ij} \neq v_{i'j'}"$, i.e.
 $C_{v_{ij} v_{i'j'}} = \{(d, d') \in D \times D | d \neq d'\}$, for: $i = i'$ (same row), or $j = j'$ (same column), or
 $(\lceil \frac{i}{3} \rceil, \lceil \frac{j}{3} \rceil) = (\lceil \frac{i'}{3} \rceil, \lceil \frac{j'}{3} \rceil)$ (same block).

Note that the ideas are still the same as in 1.5, but in constraint networks we have a language to formulate things precisely.

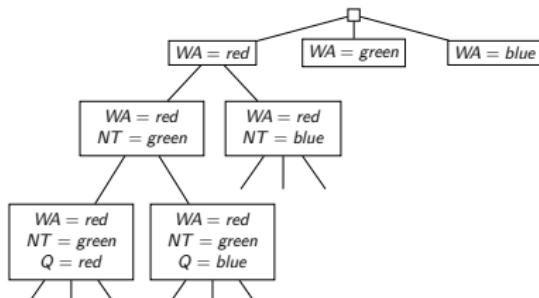
- ▶ **Definition 3.24.** Let $\langle V, D, C \rangle$ be a constraint network, then we call a partial function $a: V \rightarrow \bigcup_{u \in V} D_u$ a **partial assignment** if $a(v) \in D_v$ for all $v \in \text{dom}(V)$. If a is total, then we simply call a an **assignment**.
- ▶ **Definition 3.25.** A partial assignment a is called **inconsistent**, iff there are variables $u, v \in \text{dom}(a)$ and $C_{uv} \in C$, but $(a(u), a(v)) \notin C_{uv}$. a is called **consistent**, iff it is not inconsistent.
- ▶ **Definition 3.26.** We call $v \in D_u$ a **legal** value for a variable u , iff (u, v) is a consistent assignment.
- ▶ **Example 3.27.** The **empty assignment** \emptyset is (trivially) consistent
- ▶ **Definition 3.28.** Let f and g be partial assignments, then we say that f **extends** (or is an **extension of**) g , iff $\text{dom}(g) \subseteq \text{dom}(f)$ and $f|_{\text{dom}(g)} = g$.
- ▶ **Definition 3.29.** Let $\gamma := \langle V, D, C \rangle$ be a constraint network, then we call a consistent (total) assignment a **solution** γ and γ itself **solvable** or **satisfiable**.

- ▶ **Lemma 3.30.** Higher order constraints can be transformed into equi-satisfiable binary constraints using auxiliary variables.
- ▶ **Corollary 3.31.** Any CSP can be represented by a constraint network.
- ▶ In other words The notion of a constraint network is a refinement of that of a CSP.
- ▶ So we will stick to constraint networks in this course.
- ▶ **Observation 3.32.** We can view a constraint network as a search problem, if we take the states as the partial assignments, the actions as assignment extension, and the goal states as consistent assignments.
- ▶ Idea: We will explore that idea for algorithms that solve constraint networks.

4 CSP as Search

Standard search formulation (incremental)

- ▶ Let's start with the straightforward, dumb approach, then fix it
- ▶ States are defined by the values assigned so far
- ▶ Initial state: the empty assignment, \emptyset
- ▶ Successor function: assign a value to an unassigned variable that does not conflict with current assignment.
- ▶ \rightsquigarrow fail if no consistent assignments (not fixable!)
- ▶ Goal test: the current assignment is complete
- ▶ This is the same for all CSPs! ☺
- ▶ Every solution appears at depth n with n variables (\rightsquigarrow use depth-first search)
- ▶ Path is irrelevant, so can also use complete-state formulation
- ▶ $b = (n - \ell)d$ at depth ℓ , hence $n!d^n$ leaves!!!! ☺



- ▶ Assignments for different variables are independent!
 - ▶ e.g. first WA = red then NT = green vs. first NT = green then WA = red
 - ▶ ↵ we only need to consider assignments to a single variable at each node
 - ▶ ↵ $b=d$ and there are d^n leaves.
- ▶ **Definition 4.1.** Depth-first search for CSPs with single-variable assignments is called **backtracking search**.
- ▶ Backtracking search is the basic uninformed algorithm for CSPs
- ▶ Can solve n -queens for $n \approx 25$

Backtracking Search (Implementation)

- ▶ **Definition 4.2.** The generic backtracking search algorithm

```
procedure Backtracking–Search(csp) returns solution/failure
    return Recursive–Backtracking ( $\emptyset$ , csp)
```

```
procedure Recursive–Backtracking (assignment) returns soln/failure
    if assignment is complete then return assignment
    var := Select–Unassigned–Variable(Variables[csp], assignment, csp)
    foreach value in Order–Domain–Values(var, assignment, csp) do
        if value is consistent with assignment given Constraints[csp] then
            add  $\{ \text{var} = \text{value} \}$  to assignment
            result := Recursive–Backtracking(assignment,csp)
            if result  $\neq$  failure then return result
            remove  $\{ \text{var} = \text{value} \}$  from assignment
    return failure
```

Backtracking in Australia

- ▶ **Example 4.3.** We apply backtracking search for a map coloring problem:



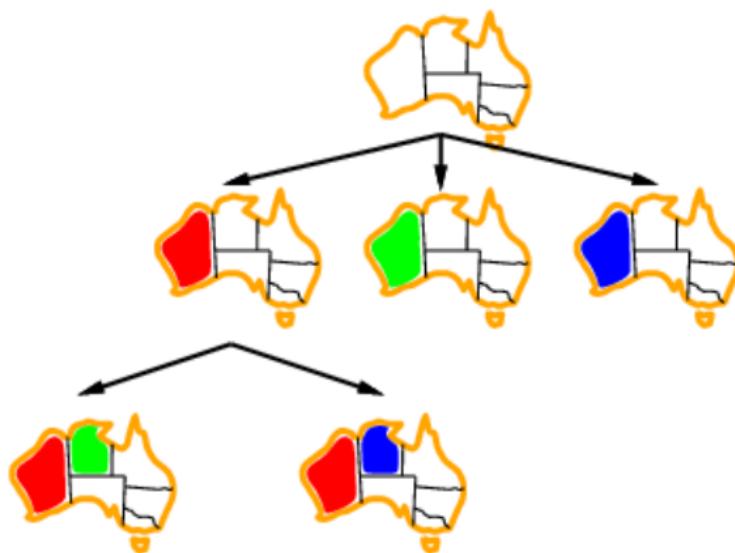
Backtracking in Australia

- ▶ **Example 4.3.** We apply backtracking search for a map coloring problem:



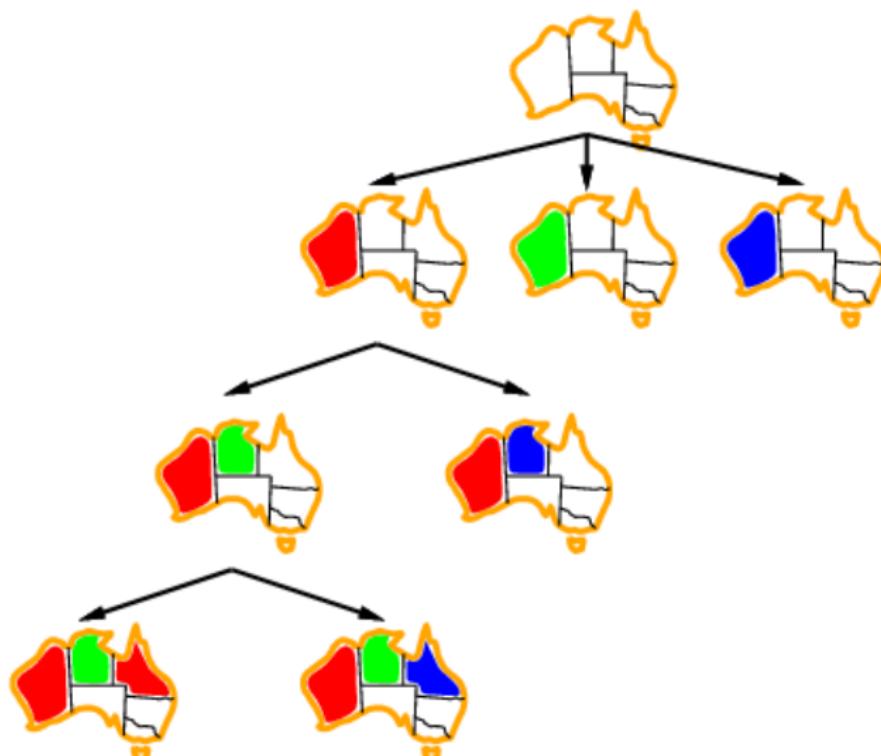
Backtracking in Australia

- ▶ **Example 4.3.** We apply backtracking search for a map coloring problem:



Backtracking in Australia

- ▶ **Example 4.3.** We apply backtracking search for a map coloring problem:

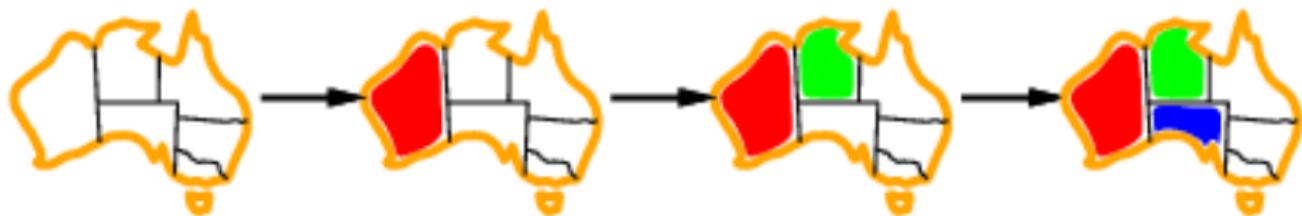


Improving backtracking efficiency

- ▶ General-purpose methods can give huge gains in speed for backtracking search.
- ▶ Answering the following questions well helps:
 1. Which variable should be assigned next?
 2. In what order should its values be tried?
 3. Can we detect inevitable failure early?
 4. Can we take advantage of problem structure?
- ▶ We use answers to these questions to give powerful heuristics.

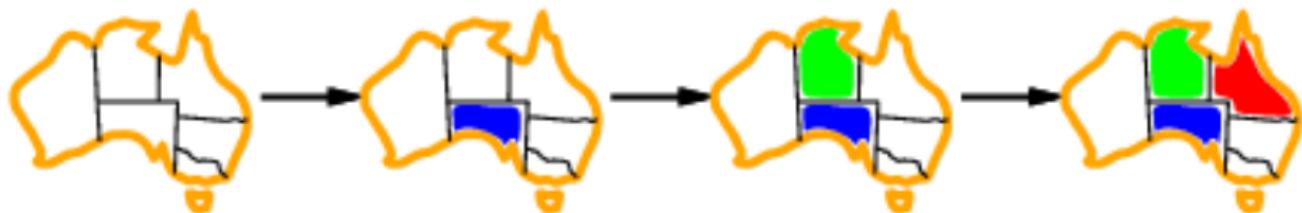
Heuristic: Minimum Remaining Values (Which Variable)

- ▶ **Definition 4.4.** The **minimum remaining values (MRV)** heuristic for backtracking search always chooses the **variable** with the fewest **legal** values, i.e. such that $\#\{d \in D_v \mid a \cup \{v \mapsto d\} \text{ is consistent}\}$ is minimal
- ▶ **Intuition:** By choosing a most constrained variable v first, we reduce the branching factor (number of sub-trees generated for v) and thus reduce the size of our search tree.
- ▶ **Extreme case:** If $\#\{d \in D_v \mid a \cup \{v \mapsto d\} \text{ is consistent}\} = 1$, then the value assignment to v is **forced** by our previous choices.
- ▶ **Example 4.5.** in step 3, there is only one remaining value for SA!



Degree Heuristic (Variable Order Tie Breaker)

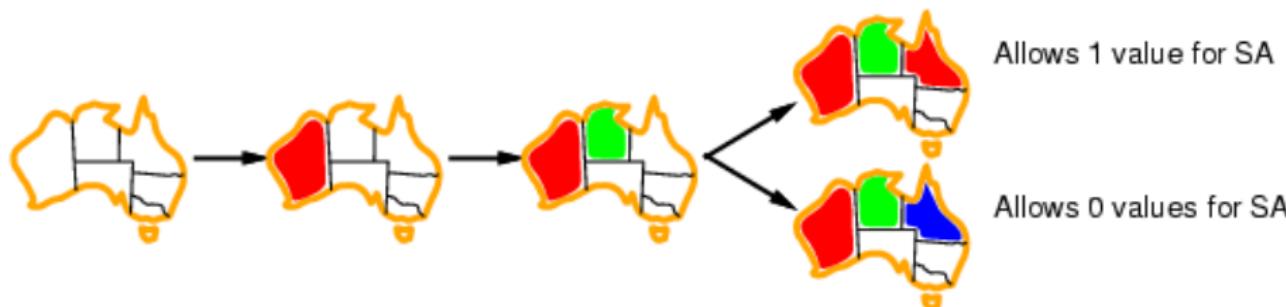
- ▶ Problem: Need a tie-breaker among MRV variables (there was no preference in step 1,2)
- ▶ **Definition 4.6.** The **degree heuristic** in backtracking search always chooses the variable with the most constraints on remaining variables, ie. always pick a v with $\#\{v \in (V \setminus \text{dom}(a)) \mid C_{uv} \in C\}$ 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 variables, pick a most constraining variable.
- ▶ **Example 4.7.**



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

Least Constraining Value Heuristic (Value Ordering)

- ▶ **Definition 4.8.** Given a variable, the **least constraining value** heuristic chooses the least constraining value: the one that rules out the fewest values in the remaining variables, i.e. for a given variable v pick the value $d \in D_v$ with $\#\{e \in (D_u \setminus \text{dom}(a)) \mid C_{uv} \in C \text{ and } (e, d) \notin C_{uv}\}$ minimal.
- ▶ By choosing the least constraining value first, we increase the chances to not rule out the solutions below the current node.
- ▶ **Example 4.9.**



- ▶ Combining these heuristics makes 1000 queens feasible

5 Conclusion & 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 **partial assignment** a maps some variables to values, a **total assignment** does so for all variables. 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** instantiates variables one-by-one, pruning inconsistent partial assignments.
 - ▶ **Variable orderings** in backtracking can dramatically reduce the size of the search tree. **Value orderings** have this potential (only) in solvable sub-trees.
- ▶ Up next: Inference and decomposition, for improved efficiency.

Chapter 10 Constraint Propagation

1 Introduction

► Constraint network γ :



- **Question:** An additional constraint we can add without losing any solutions?
- For example, $C_{WAQ} := "="$. 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 = obtaining an equivalent network with a “tighter description” and hence with a smaller number of consistent partial assignments.

Illustration: Decomposition

- ▶ **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.
- ▶ Extreme case: No inter-dependencies at all, as in our example here.

Our Agenda for This Chapter

- ▶ **Inference:** 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.

2 Inference

- ▶ **Definition 2.1.** **Inference** in constraint networks consists in deducing additional constraints (unary or binary), that follow from the already known constraints, i.e. that are satisfied in all solutions.
- ▶ **Example 2.2.** It's what you do all the time when playing SuDoKu:

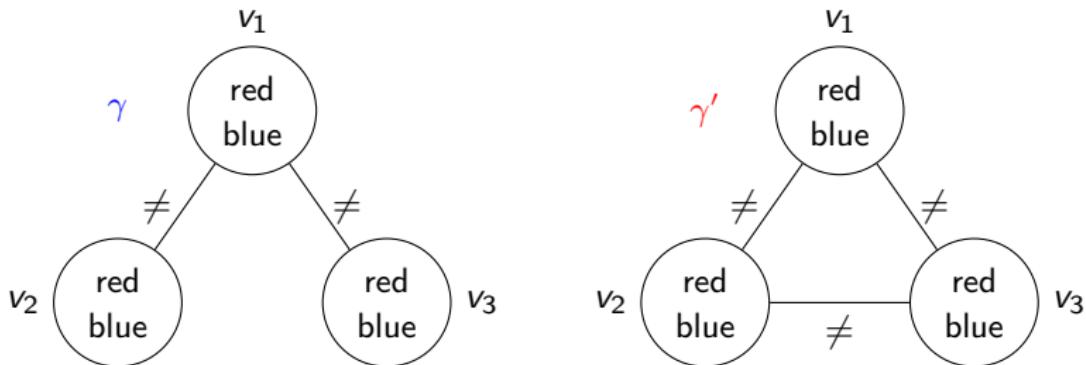
	5	8	7		6	9	4	1
		9	8		4	3	5	7
4		7	9		5	2	6	8
3	9	5	2	7	1	4	8	6
7	6	2	4	9	8	1	3	5
8	4	1	6	5	3	7	2	9
1	8	4	3	6	9	5	7	2
5	7	6	1	4	2	8	9	3
9	2	3	5	8	7	6	1	4

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

- ▶ **Definition 2.3.** Let $\gamma := \langle V, D, C \rangle$ and $\gamma' := \langle V, D', C' \rangle$ be constraint networks sharing the same set of variables. We say that γ and γ' are **equivalent**, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- ▶ **Example 2.4.**

Equivalent Constraint Networks

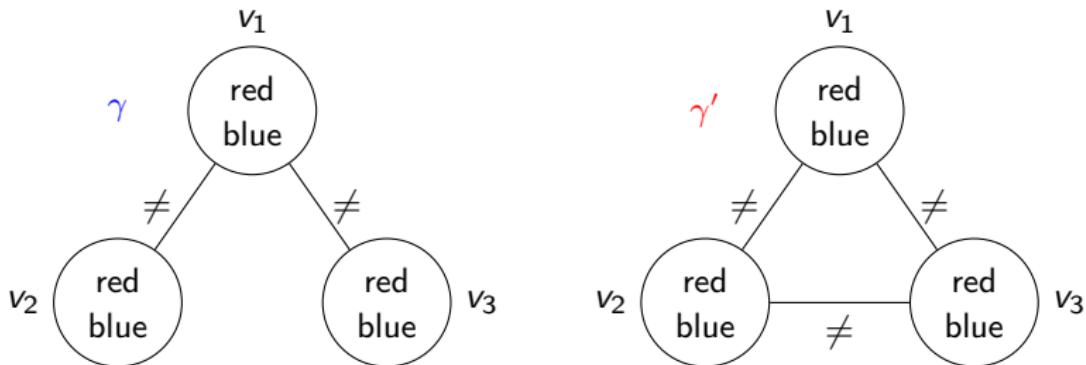
- ▶ **Definition 2.3.** Let $\gamma := \langle V, D, C \rangle$ and $\gamma' := \langle V, D', C' \rangle$ be constraint networks sharing the same set of variables. We say that γ and γ' are **equivalent**, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- ▶ **Example 2.4.**



Are these constraint networks equivalent?

Equivalent Constraint Networks

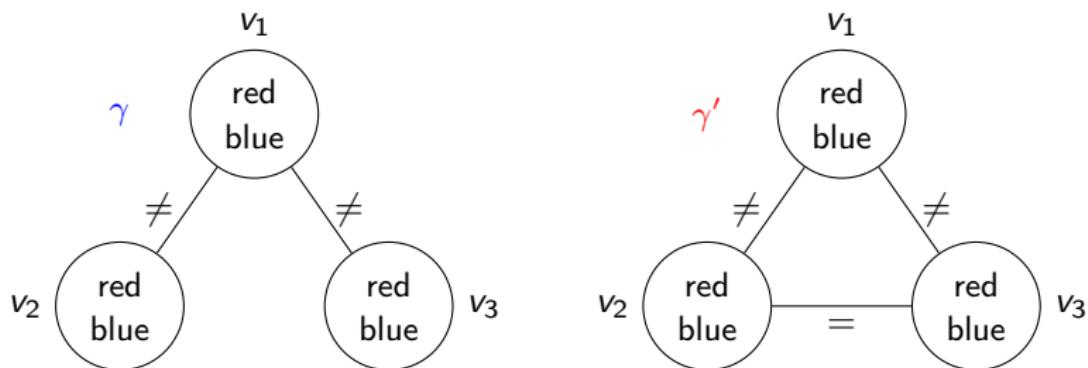
- ▶ **Definition 2.3.** Let $\gamma := \langle V, D, C \rangle$ and $\gamma' := \langle V, D', C' \rangle$ be constraint networks sharing the same set of variables. We say that γ and γ' are **equivalent**, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- ▶ **Example 2.4.**



Are these constraint networks equivalent? No.

Equivalent Constraint Networks

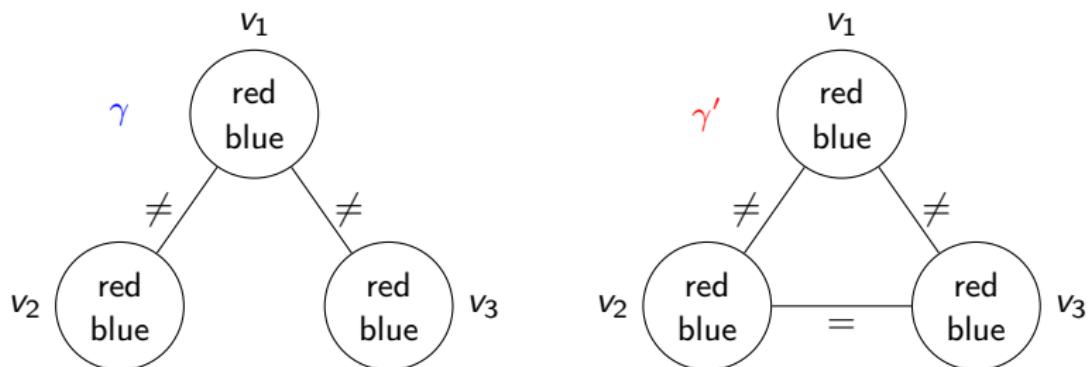
- ▶ **Definition 2.3.** Let $\gamma := \langle V, D, C \rangle$ and $\gamma' := \langle V, D', C' \rangle$ be constraint networks sharing the same set of variables. We say that γ and γ' are **equivalent**, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- ▶ **Example 2.4.**



Are these constraint networks equivalent?

Equivalent Constraint Networks

- ▶ **Definition 2.3.** Let $\gamma := \langle V, D, C \rangle$ and $\gamma' := \langle V, D', C' \rangle$ be constraint networks sharing the same set of variables. We say that γ and γ' are **equivalent**, (write $\gamma' \equiv \gamma$), if they have the same solutions.
- ▶ **Example 2.4.**



Are these constraint networks equivalent? Yes.

Tightness

► **Definition 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. We say that 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 $C'_{uv} \not\subseteq C_{uv}$ or $C'_{uv} \subseteq C_{uv}$.

γ' is **strictly tighter** than γ , (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is strict.

► **Example 2.6.**

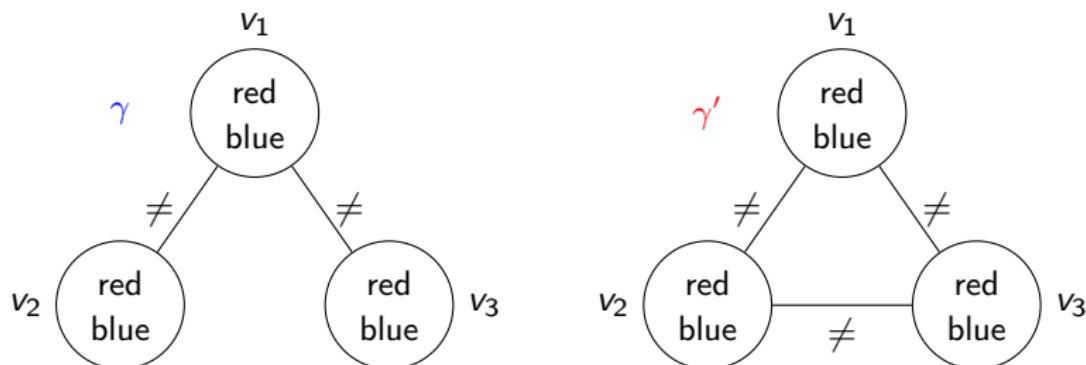
Tightness

► **Definition 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. We say that 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 $C'_{uv} \not\subseteq C_{uv}$ or $C'_{uv} \subseteq C_{uv}$.

γ' is **strictly tighter** than γ , (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is strict.

► **Example 2.6.**



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

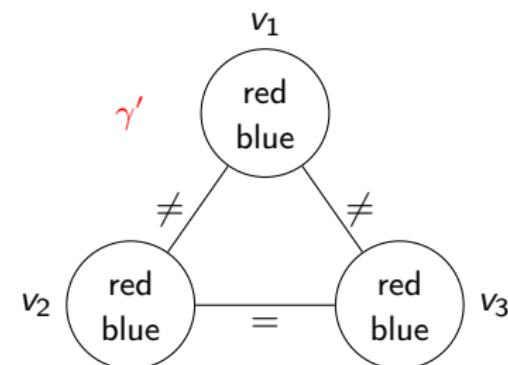
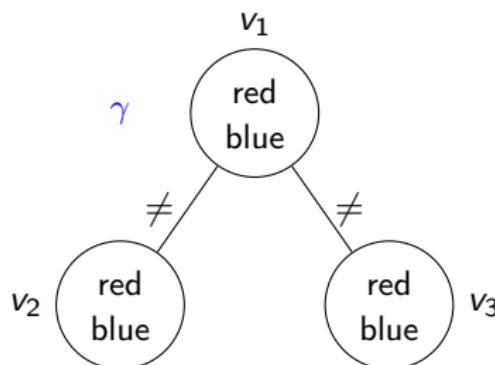
Tightness

► **Definition 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. We say that 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 $C'_{uv} \not\subseteq C_{uv}$ or $C'_{uv} \subseteq C_{uv}$.

γ' is **strictly tighter** than γ , (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is strict.

► **Example 2.6.**



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

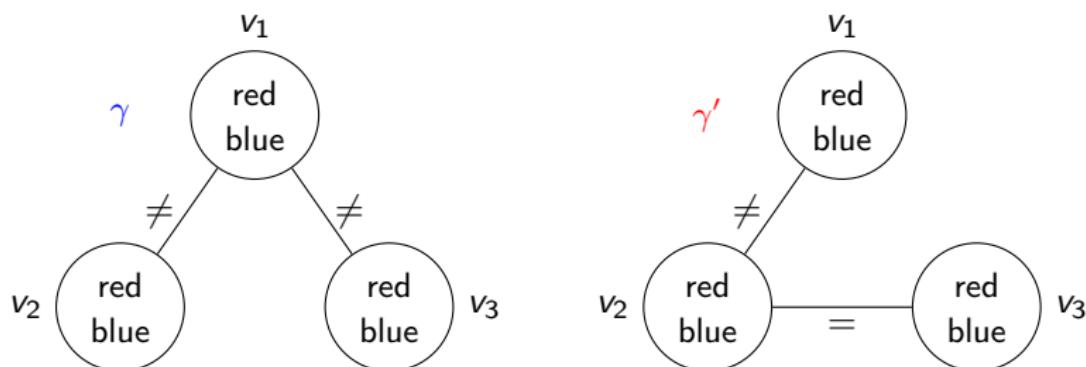
Tightness

► **Definition 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. We say that 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 $C'_{uv} \not\subseteq C_{uv}$ or $C'_{uv} \subseteq C_{uv}$.

γ' is **strictly tighter** than γ , (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is strict.

► **Example 2.6.**



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

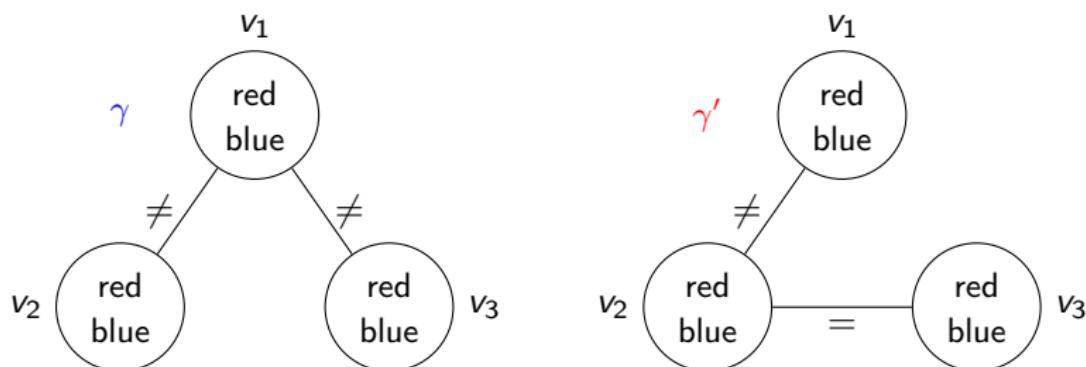
Tightness

► **Definition 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. We say that 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 $C'_{uv} \not\subseteq C_{uv}$ or $C'_{uv} \subseteq C_{uv}$.

γ' is **strictly tighter** than γ , (written $\gamma' \sqsubset \gamma$), if at least one of these inclusions is strict.

► **Example 2.6.**

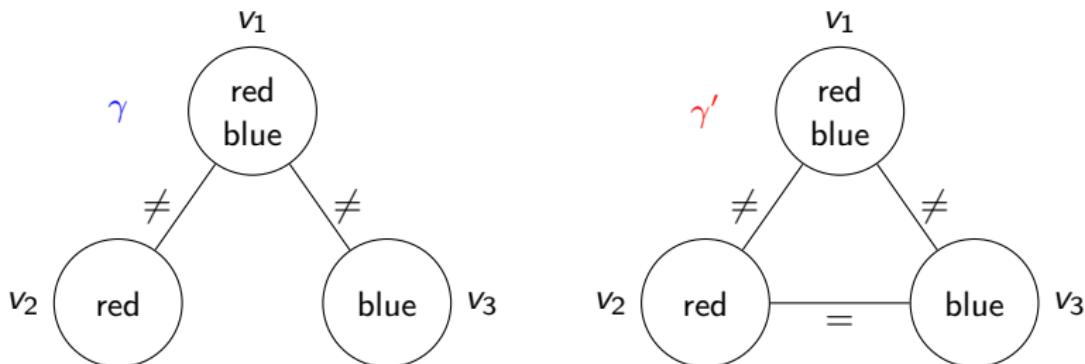


Here, we do not have $\gamma' \sqsubseteq \gamma$.

► Tightness: " γ' has the same constraints as γ , plus some".

Equivalence + Tightness = Inference

- ▶ **Theorem 2.7.** Let γ and γ' be constraint networks s.t. $\gamma' \equiv \gamma$ and $\gamma' \sqsubseteq \gamma$. Then γ' has the same solutions as, but fewer consistent partial assignments than, γ .
- ▶ $\sim \gamma'$ is a better encoding of the underlying problem.
- ▶ **Example 2.8.**



a cannot be extended to a solution (neither in γ nor in γ' because they're equivalent). a is consistent with γ , but not with γ' .

How to Use Inference in CSP Solvers?

- ▶ Simple: Inference as a pre-process:
 - ▶ When: Just once before search starts.
 - ▶ Effect: Little runtime overhead, little pruning power. Not considered here.
- ▶ More Advanced: Inference during search:
 - ▶ When: At every recursive call of backtracking.
 - ▶ Effect: Strong pruning power, may have large runtime overhead.
- ▶ Search vs. Inference: The more complex the inference, the *smaller* the number of search nodes, but the *larger* the runtime needed at each node.
- ▶ Idea: Encode partial assignment 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*.

Backtracking With Inference

- **Definition 2.9.** The general algorithm for **backtracking with inference** is

```
function BacktrackingWithInference( $\gamma, a$ ) returns a solution, or "inconsistent"
  if  $a$  is inconsistent then return "inconsistent"
  if  $a$  is a total assignment then return  $a$ 
   $\gamma' := a$  copy of  $\gamma$  /*  $\gamma' = (V, D', C')$  */
   $\gamma' := \text{Inference}(\gamma')$ 
  if exists  $v$  with  $D'_v = \emptyset$  then return "inconsistent"
  select some variable  $v$  for which  $a$  is not defined
  for each  $d \in$  copy of  $D'_v$  in some order do
     $a' := a \cup \{v = d\}$ ;  $D'_v := \{d\}$  /* makes  $a$  explicit as a constraint */
     $a'' := \text{BacktrackingWithInference}(\gamma', a')$ 
    if  $a'' \neq \text{"inconsistent"}$  then return  $a''$ 
  return "inconsistent"
```

- $\text{Inference}()$: Any procedure delivering a (tighter) equivalent network.
- $\text{Inference}()$ typically prunes domains; indicate unsolvability by $D'_v = \emptyset$.
- When backtracking out of a search branch, retract the inferred constraints: these were dependent on a , the search commitments so far.

3 Forward Checking

- ▶ **Definition 3.1.** **Forward checking** propagates information about illegal values:
Whenever a variable u is assigned by a , delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.

Forward Checking

- ▶ **Definition 3.1.** **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 3.2.** **Forward checking** in Australia



WA	NT	Q	NSW	V	SA	T
Red	Green	Blue	Red	Green	Blue	Red

Forward Checking

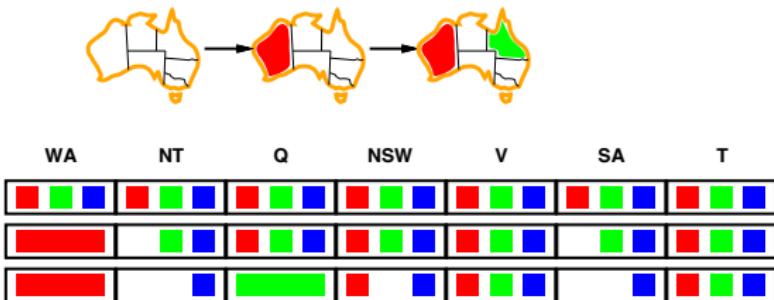
- ▶ **Definition 3.1.** **Forward checking** propagates information about illegal values:
Whenever a variable u is assigned by a , delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- ▶ **Example 3.2.** **Forward checking** in Australia



WA	NT	Q	NSW	V	SA	T
█ Red	█ Green	█ Blue	█ Green	█ Blue	█ Green	█ Blue
█ Red	█ Green	█ Blue	█ Green	█ Blue	█ Green	█ Blue

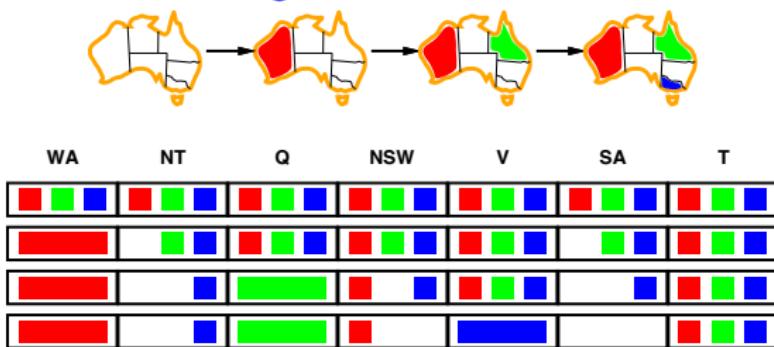
Forward Checking

- ▶ **Definition 3.1.** **Forward checking** propagates information about illegal values:
Whenever a variable u is assigned by a , delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- ▶ **Example 3.2.** **Forward checking** in Australia



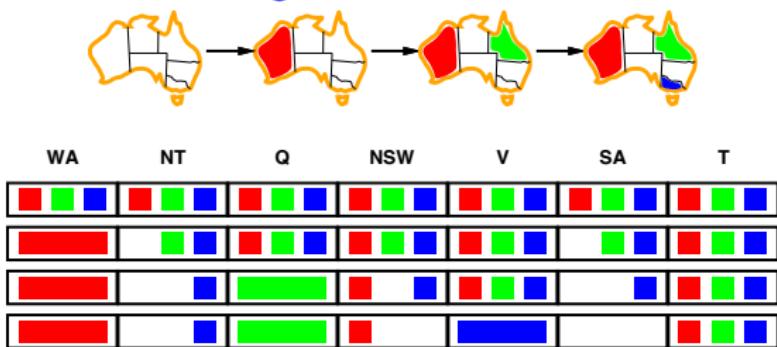
Forward Checking

- ▶ **Definition 3.1.** **Forward checking** propagates information about illegal values:
Whenever a variable u is assigned by a , delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- ▶ **Example 3.2.** **Forward checking** in Australia



Forward Checking

- ▶ **Definition 3.1.** **Forward checking** propagates information about illegal values:
Whenever a variable u is assigned by a , delete all values inconsistent with $a(u)$ from every D_v for all variables v connected with u by a constraint.
- ▶ **Example 3.2.** **Forward checking** in Australia



- ▶ Inference, Version 1: forward checking implemented

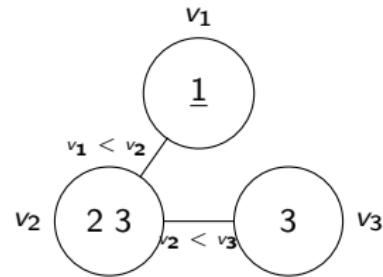
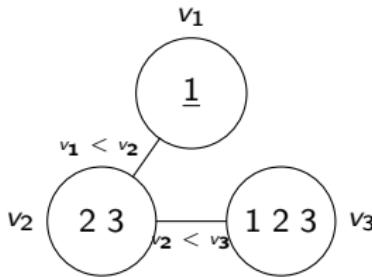
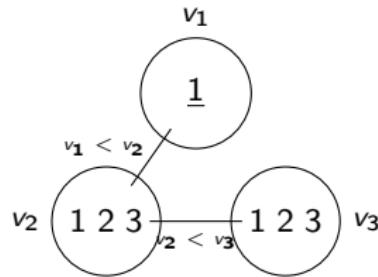
```
function ForwardChecking( $\gamma, a$ ) returns modified  $\gamma$ 
  for each  $v$  where  $a(v) = d'$  is defined do
    for each  $u$  where  $a(u)$  is undefined and  $C_{uv} \in C$  do
       $D_u := \{d \in D_u | (d, d') \in C_{uv}\}$ 
  return  $\gamma$ 
```

- ▶ Properties:
 - ▶ Forward checking is **sound**: *Its tightening of constraints does not rule out any solutions. In other words: it guarantees to deliver an equivalent network.*
 - ▶ Recall here that the partial assignment a is represented as unary constraints inside γ .
 - ▶ Please also excuse the slight mismatch with the call of "Inference(γ')" on slide 255.
 - ▶ Incremental computation: Instead of the first for-loop, use only the 2nd one every time a new assignment $a(v) = d'$ is added.
- ▶ Practice:
 - ▶ 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**).

4 Arc Consistency

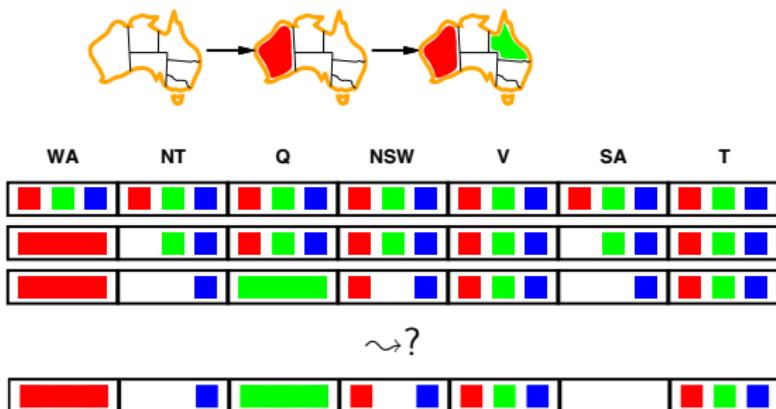
When Forward Checking is Not Good Enough I

► Example 4.1.



When Forward Checking is Not Good Enough II

► Example 4.2.



► Problem: Forward checking makes inferences only from assigned to unassigned variables.

- ▶ **Definition 4.3 (Arc Consistency).** Let $\gamma = \langle V, D, C \rangle$ be a constraint network.
 - (i) A variable $u \in V$ is **arc consistent** relative to another variable $v \in V$ if either $C_{uv} \not\subseteq C_v$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d, d') \in C_{uv}$.
 - (ii) The **constraint network γ** is **arc consistent** if every variable $u \in V$ is **arc consistent** relative to every other variable $v \in V$.
- ▶ **Intuition:** Arc consistency $\hat{=}$ 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.
- ▶ **Example 4.4 (Arc Consistency (previous slide)).**
 - ▶ **Question:** On top, middle, is v_3 **arc consistent** relative to v_2 ?

- ▶ **Definition 4.3 (Arc Consistency).** Let $\gamma = \langle V, D, C \rangle$ be a constraint network.
 - (i) A variable $u \in V$ is **arc consistent** relative to another variable $v \in V$ if either $C_{uv} \not\subseteq C_v$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d, d') \in C_{uv}$.
 - (ii) The **constraint network γ** is **arc consistent** if every variable $u \in V$ is **arc consistent** relative to every other variable $v \in V$.
- ▶ **Intuition:** Arc consistency $\hat{=}$ 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.
- ▶ **Example 4.4 (Arc Consistency (previous slide)).**
 - ▶ **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.
 - ▶ **Question:** And on the right?

- ▶ **Definition 4.3 (Arc Consistency).** Let $\gamma = \langle V, D, C \rangle$ be a constraint network.
 - (i) A variable $u \in V$ is **arc consistent** relative to another variable $v \in V$ if either $C_{uv} \not\subseteq C_v$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d, d') \in C_{uv}$.
 - (ii) The constraint network γ is **arc consistent** if every variable $u \in V$ is arc consistent relative to every other variable $v \in V$.
- ▶ **Intuition:** Arc consistency $\hat{=}$ 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.
- ▶ **Example 4.4 (Arc Consistency (previous slide)).**
 - ▶ **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.
 - ▶ **Question:** And on the right?
 - ▶ **Answer:** Yes. (But v_2 is not arc consistent relative to v_3)
 - ▶ **Note:** SA is not arc consistent relative to NT in 4.2, 3rd row.

Enforcing Arc Consistency: General Remarks

- ▶ Inference, version 2: “Enforcing Arc Consistency” = removing variable domain values until γ is arc consistent. (Up next)
- ▶ Note: Assuming such an inference method $AC(\gamma)$
- ▶ $AC(\gamma)$ is sound: guarantees to deliver an equivalent network.
- ▶ 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.
- ▶ $AC(\gamma)$ subsumes forward checking: $AC(\gamma) \sqsubseteq ForwardChecking(\gamma)$.
- ▶ Proof: sketch
 1. Forward checking removes d from D_u only if there is a constraint C_{uv} such that $D_v = \{d'\}$ (i.e. when v was assigned the value d'), and $(d, d') \notin C_{uv}$.
 2. Clearly, enforcing arc consistency of u relative to v removes d from D_u as well.



Enforcing Arc Consistency for One Pair of Variables

- ▶ **Definition 4.5 (Revise).** An algorithm enforcing arc consistency of u relative to v

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

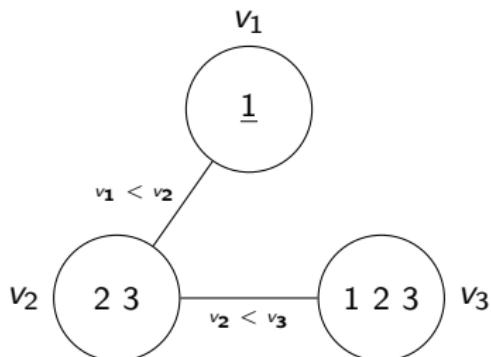
- ▶ **Lemma 4.6.** If k is maximal domain size in γ and the test " $(d, d') \in C_{uv}$?" has runtime $\mathcal{O}(1)$, then the runtime of $\text{Revise}(\gamma, u, v)$ is $\mathcal{O}(k^2)$

Enforcing Arc Consistency for One Pair of Variables

- ▶ **Definition 4.5 (Revise).** An algorithm enforcing arc consistency of u relative to v

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

- ▶ **Lemma 4.6.** If k is maximal domain size in γ and the test " $(d, d') \in C_{uv}$?" has runtime $\mathcal{O}(1)$, then the runtime of $\text{Revise}(\gamma, u, v)$ is $\mathcal{O}(k^2)$
- ▶ **Example 4.7.** $\text{Revise}(\gamma, v_3, v_2)$

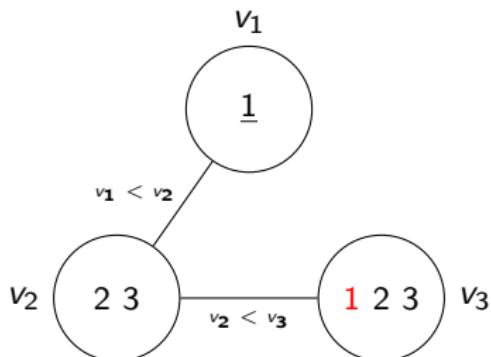


Enforcing Arc Consistency for One Pair of Variables

- ▶ **Definition 4.5 (Revise).** An algorithm enforcing arc consistency of u relative to v

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

- ▶ **Lemma 4.6.** If k is maximal domain size in γ and the test " $(d, d') \in C_{uv}$?" has runtime $\mathcal{O}(1)$, then the runtime of $\text{Revise}(\gamma, u, v)$ is $\mathcal{O}(k^2)$
- ▶ **Example 4.7.** $\text{Revise}(\gamma, v_3, v_2)$

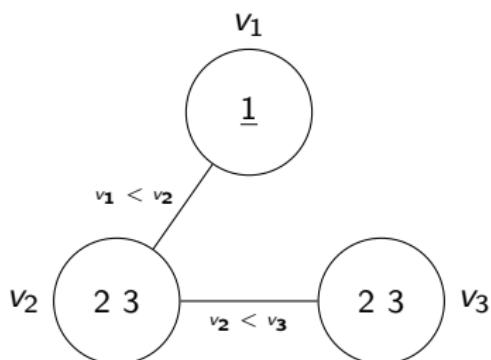


Enforcing Arc Consistency for One Pair of Variables

- ▶ **Definition 4.5 (Revise).** An algorithm enforcing arc consistency of u relative to v

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

- ▶ **Lemma 4.6.** If k is maximal domain size in γ and the test " $(d, d') \in C_{uv}$?" has runtime $\mathcal{O}(1)$, then the runtime of $\text{Revise}(\gamma, u, v)$ is $\mathcal{O}(k^2)$
- ▶ **Example 4.7.** $\text{Revise}(\gamma, v_3, v_2)$

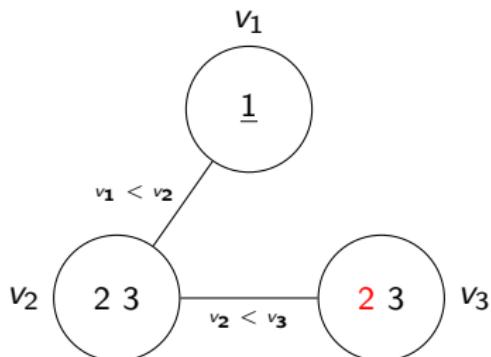


Enforcing Arc Consistency for One Pair of Variables

- ▶ **Definition 4.5 (Revise).** An algorithm enforcing arc consistency of u relative to v

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

- ▶ **Lemma 4.6.** If k is maximal domain size in γ and the test " $(d, d') \in C_{uv}$?" has runtime $\mathcal{O}(1)$, then the runtime of $\text{Revise}(\gamma, u, v)$ is $\mathcal{O}(k^2)$
- ▶ **Example 4.7.** $\text{Revise}(\gamma, v_3, v_2)$

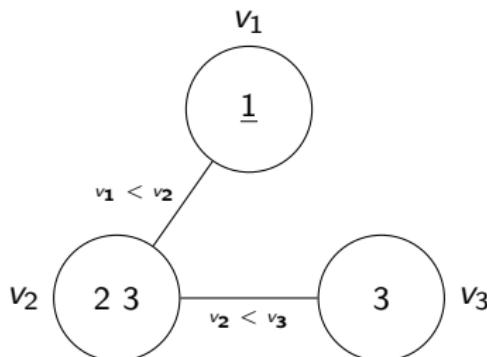


Enforcing Arc Consistency for One Pair of Variables

- ▶ **Definition 4.5 (Revise).** An algorithm enforcing arc consistency of u relative to v

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

- ▶ **Lemma 4.6.** If k is maximal domain size in γ and the test " $(d, d') \in C_{uv}$?" has runtime $\mathcal{O}(1)$, then the runtime of $\text{Revise}(\gamma, u, v)$ is $\mathcal{O}(k^2)$
- ▶ **Example 4.7.** $\text{Revise}(\gamma, v_3, v_2)$

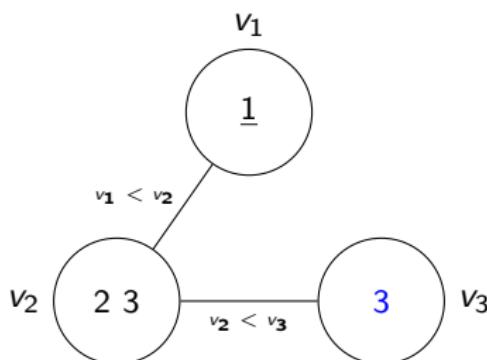


Enforcing Arc Consistency for One Pair of Variables

- ▶ **Definition 4.5 (Revise).** An algorithm enforcing arc consistency of u relative to v

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

- ▶ **Lemma 4.6.** If k is maximal domain size in γ and the test " $(d, d') \in C_{uv}$?" has runtime $\mathcal{O}(1)$, then the runtime of $\text{Revise}(\gamma, u, v)$ is $\mathcal{O}(k^2)$
- ▶ **Example 4.7.** $\text{Revise}(\gamma, v_3, v_2)$



AC-1: Enforcing Arc Consistency (Version 1)

- ▶ **Idea:** Apply Revise pairwise up to a fixed point.
- ▶ **Definition 4.8.** AC-1 enforces **arc consistency** in **constraint networks**:

```
function AC-1( $\gamma$ ) returns modified  $\gamma$ 
repeat
    changesMade := False
    for each constraint  $C_{uv}$  do
        Revise( $\gamma, u, v$ ) /* if  $D_u$  reduces, set changesMade := True */
        Revise( $\gamma, v, u$ ) /* if  $D_v$  reduces, set changesMade := True */
    until changesMade = False
    return  $\gamma$ 
```

- ▶ **Observation:** Obviously, this does indeed enforce **arc consistency** for γ .
- ▶ **Lemma 4.9.** If γ has n variables, m constraints, and maximal domain size k , then the runtime of $AC1(\gamma)$ is $\mathcal{O}(mk^2nk)$
- ▶ **Proof Sketch:** $\mathcal{O}(mk^2)$ for each inner loop, fixed point reached at the latest once all nk 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 their domains haven't changed since the last time.

AC-3: Enforcing Arc Consistency (Version 3)

- ▶ **Idea:** Remember the potentially inconsistent variable pairs.
- ▶ **Definition 4.10.** AC-3 optimizes AC-1 for enforcing arc consistency.

function $\text{AC-3}(\gamma)$ **returns** modified γ

$M := \emptyset$

for each constraint $C_{uv} \in C$ **do**

$M := M \cup \{(u,v), (v,u)\}$

while $M \neq \emptyset$ **do**

remove any element (u,v) from M

$\text{Revise}(\gamma, u, v)$

if D_u has changed **in** the call **to** Revise **then**

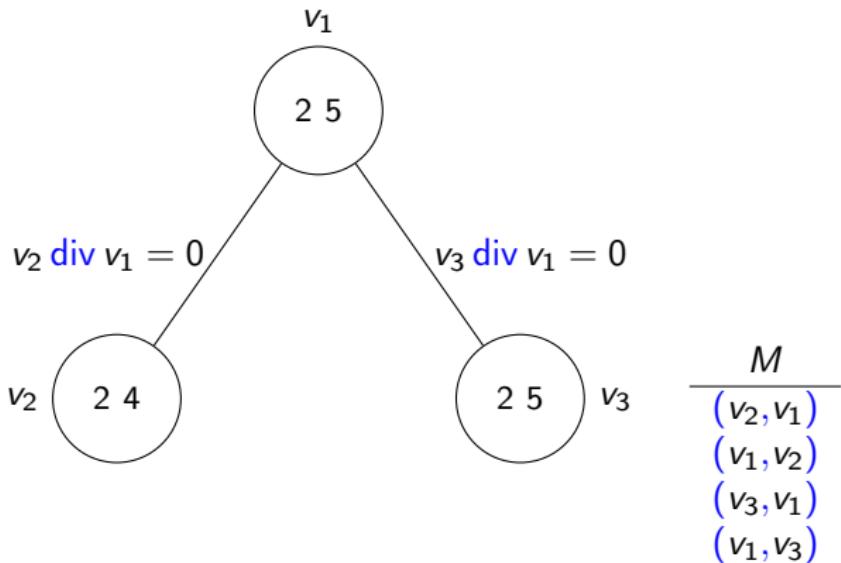
for each constraint $C_{wu} \in C$ where $w \neq v$ **do**

$M := M \cup \{(w,u)\}$

return γ

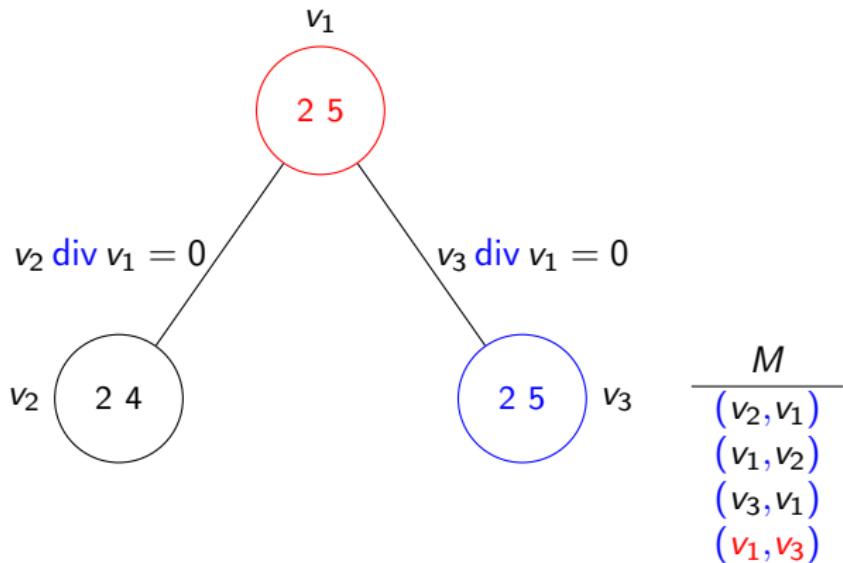
- ▶ **Question:** $\text{AC-3}(\gamma)$ enforces arc consistency because?
- ▶ **Answer:** At any time during the while-loop, if $(u,v) \notin M$ then u is arc consistent relative to v .
- ▶ **Question:** Why only "where $w \neq v$ "?
- ▶ **Answer:** If $w = v$ is the reason why D_u changed, then w is still arc consistent relative to u : the values just removed from D_u did not match any values from

- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x



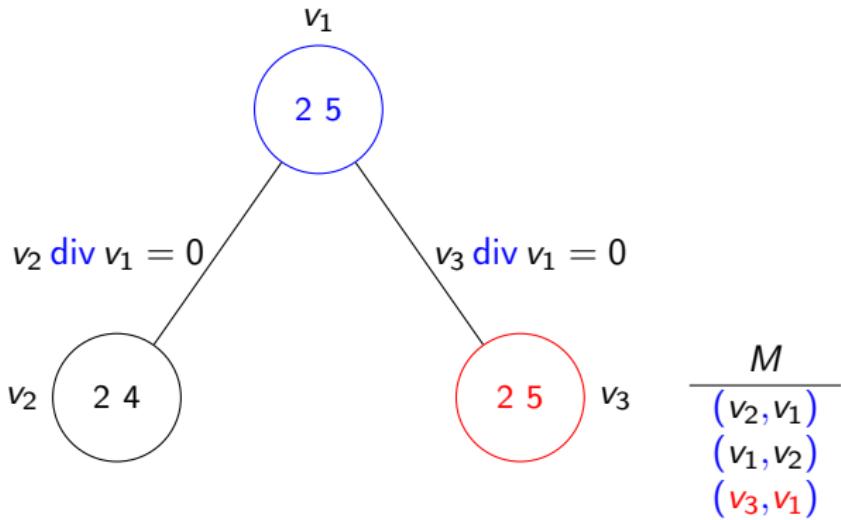
AC-3: Example

- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x

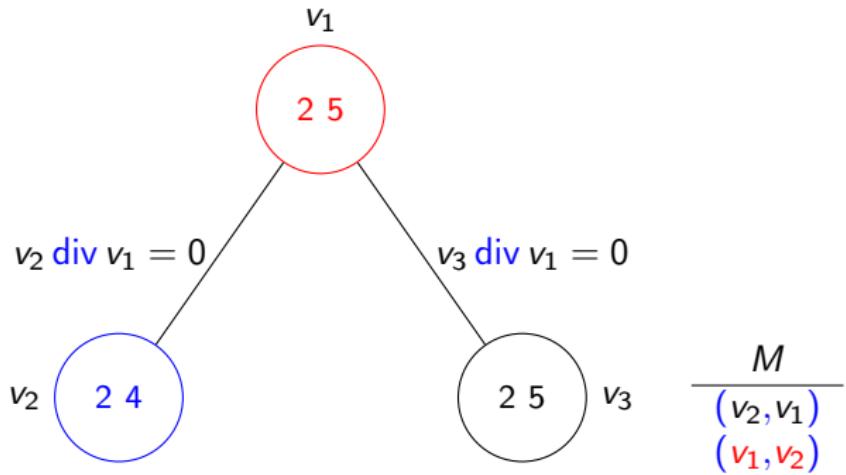


AC-3: Example

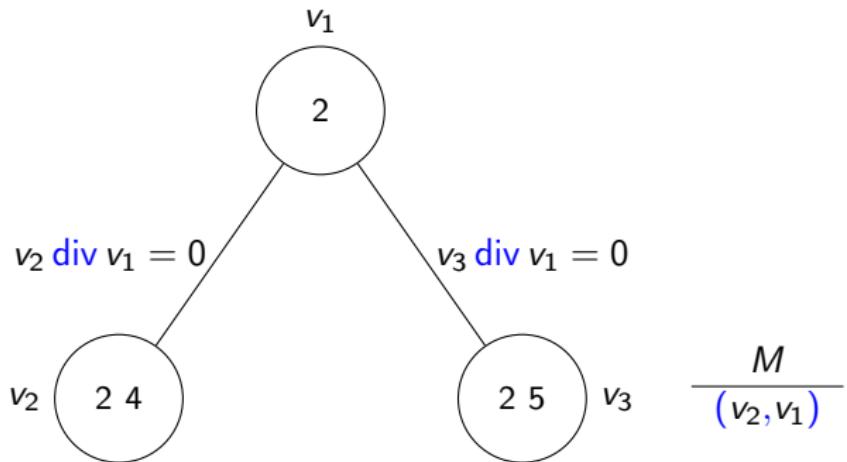
- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x



- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x

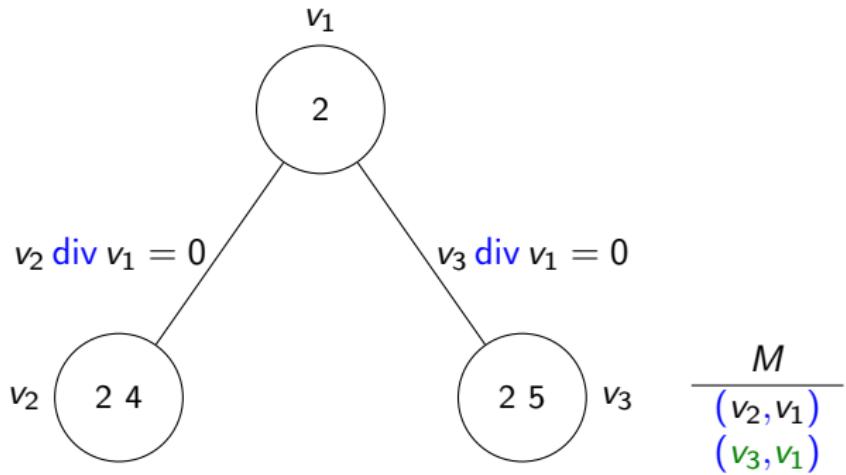


- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x



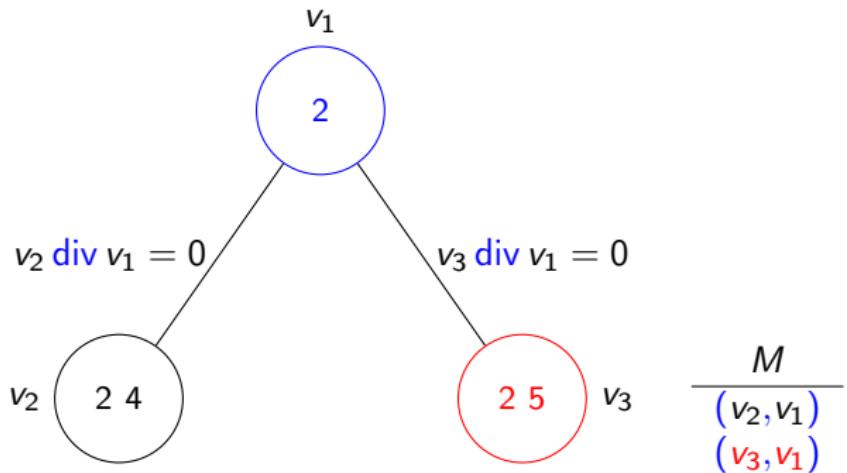
AC-3: Example

- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x

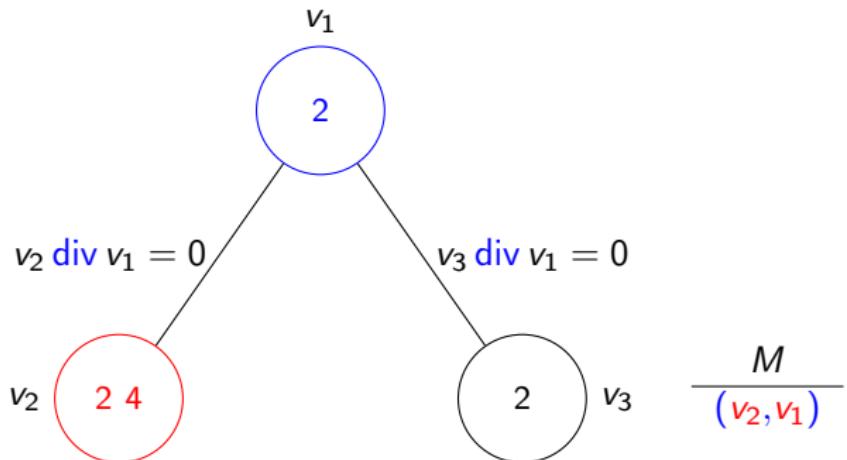


AC-3: Example

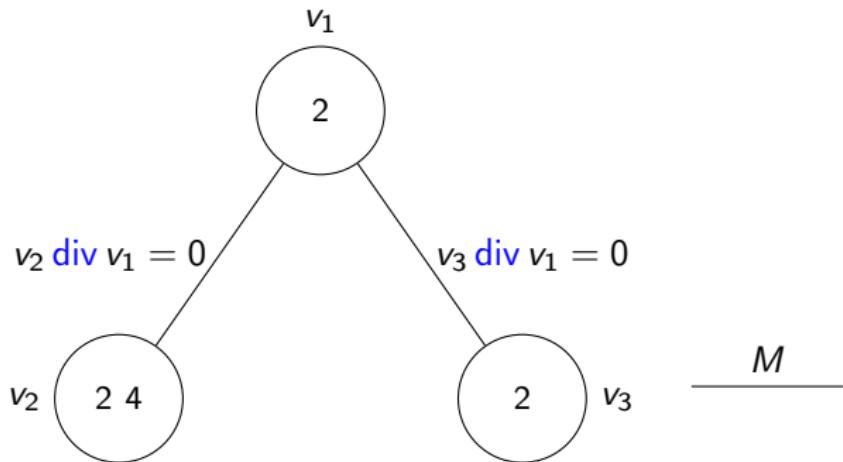
- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x



- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x



- **Example 4.11.** $y \text{ div } x = 0$: y modulo x is 0, i.e., y can be divided by x



- ▶ **Theorem 4.12 (Runtime of AC-3).** Let $\gamma = \langle V, D, C \rangle$ be a constraint network with m constraints, and maximal domain size k . Then $\text{AC-3}(\gamma)$ runs in time $\mathcal{O}(mk^3)$.
- ▶ **Proof:** by counting how often Revise is called.
 1. Each call to $\text{Revise}(\gamma, u, v)$ takes time $\mathcal{O}(k^2)$ so it suffices to prove that at most $\mathcal{O}(mk)$ of these calls are made.
 2. The number of calls to $\text{Revise}(\gamma, u, v)$ is the number of iterations of the while-loop, which is at most the number of insertions into M .
 3. Consider any constraint C_{uv} .
 4. Two variable pairs corresponding to C_{uv} are inserted in the for-loop. In the while loop, if a pair corresponding to C_{uv} is inserted into M , then
 5. beforehand the domain of either u or v was reduced, which happens at most $2k$ times.
 6. Thus we have $\mathcal{O}(k)$ insertions per constraint, and $\mathcal{O}(mk)$ insertions overall, as desired.

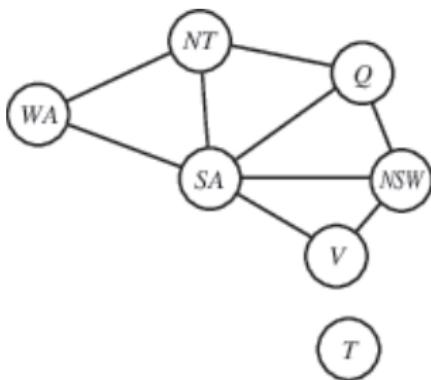


5 Decomposition: Constraint Graphs, and Three Simple Cases

- ▶ Say γ is a **constraint network** with n variables and maximal domain size k .
 - ▶ k^n total assignments must be tested in the worst case to solve γ
- ▶ **Inference:** One method to try to avoid/ameliorate this 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 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

- ▶ Tasmania and mainland are “independent subproblems”
- ▶ **Definition 5.1.** **Independent subproblems** are identified as **connected components** of constraint graphs.
- ▶ Suppose each subproblem has c variables out of n total
- ▶ Worst-case solution cost is $n \text{ div } c \cdot d^c$ (linear in n)
- ▶ E.g., $n = 80$, $d = 2$, $c = 20$
 - ▶ $2^{80} = 4$ billion years at 10 million nodes/sec
 - ▶ $42^{20} = 0.4$ seconds at 10 million nodes/sec



“Decomposition” 1.0: Disconnected Constraint Graphs

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

“Decomposition” 1.0: Disconnected Constraint Graphs

- ▶ **Theorem 5.2 (Disconnected Constraint Graphs).** Let $\gamma = \langle V, D, C \rangle$ be a constraint network. Let a_i be a solution to each connected component V_i of the constraint graph of γ . Then $a := \bigcup_i a_i$ is a solution to γ .
- ▶ Proof:
 1. a satisfies all C_{uv} where u and v are inside the same connected component.
 2. The latter is the case for all C_{uv} .
 3. If two parts of γ are not connected, then they are independent.

□

“Decomposition” 1.0: Disconnected Constraint Graphs

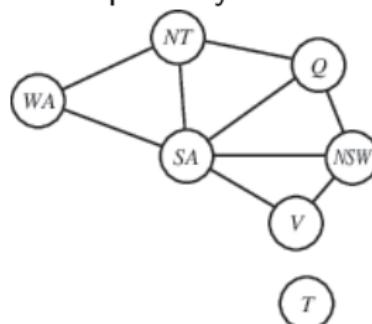
- **Theorem 5.2 (Disconnected Constraint Graphs).** Let $\gamma = \langle V, D, C \rangle$ be a constraint network. Let a_i be a solution to each connected component V_i of the constraint graph of γ . Then $a := \bigcup_i a_i$ is a solution to γ .

► Proof:

1. a satisfies all C_{uv} where u and v are inside the same connected component.
2. The latter is the case for all C_{uv} .
3. If two parts of γ are not connected, then they are independent.

□

► **Example 5.3.** Color Tasmania separately in Australia



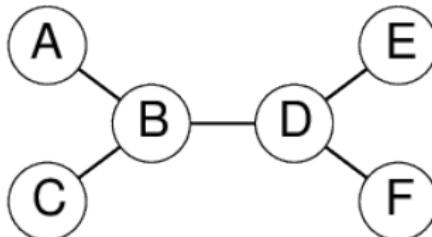
“Decomposition” 1.0: Disconnected Constraint Graphs

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

1. a satisfies all C_{uv} where u and v are inside the same connected component.
2. The latter is the case for all C_{uv} .
3. If two parts of γ are not connected, then they are independent.

□

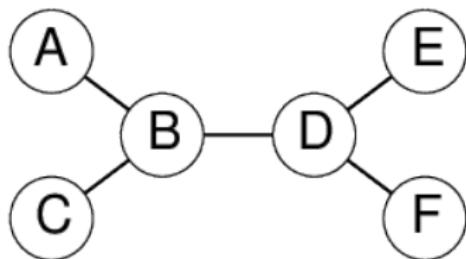
- ▶ **Example 5.3.** Color Tasmania separately in Australia
- ▶ **Example 5.4 (Doing the Numbers).**
 - ▶ γ with $n = 40$ variables, each domain size $k = 2$. Four separate connected components each of size 10.
 - ▶ Reduction of worst-case when using decomposition:
 - ▶ No decomposition: 2^{40} . With: $4 \cdot 2^{10}$. Gain: $2^{28} \approx 280.000.000$.



- ▶ **Theorem 5.5.** if the constraint graph has no loops, the CSP can be solved in $\mathcal{O}(nd^2)$ time
- ▶ Compare to general CSPs, where worst-case time is $\mathcal{O}(d^n)$
- ▶ This property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and the complexity of reasoning.

Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



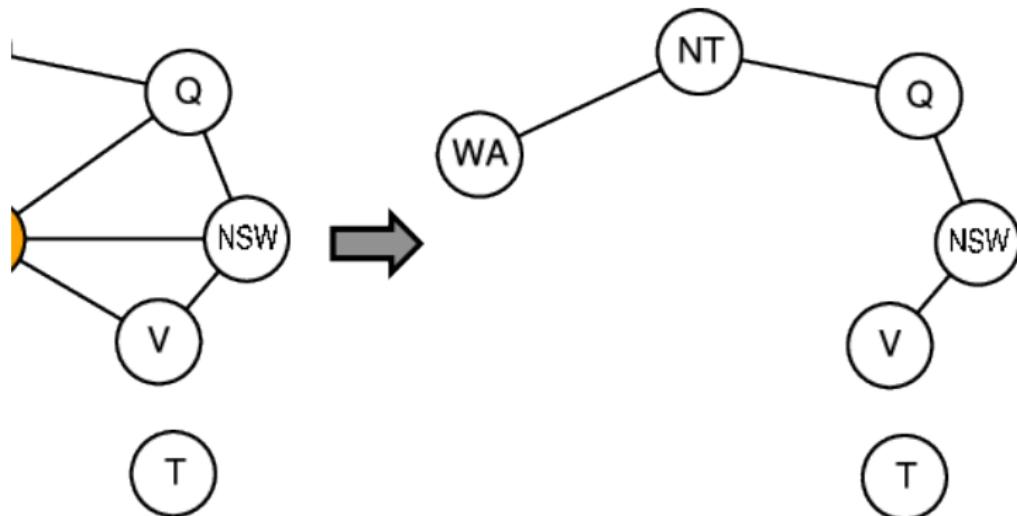
2. For j from n down to 2, apply

```
RemoveInconsistent(Parent($X_j$, $X_j$))
```

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

Nearly tree-structured CSPs

- ▶ **Definition 5.6. Conditioning:** instantiate a variable, prune its neighbors' domains
- ▶ **Example 5.7.**



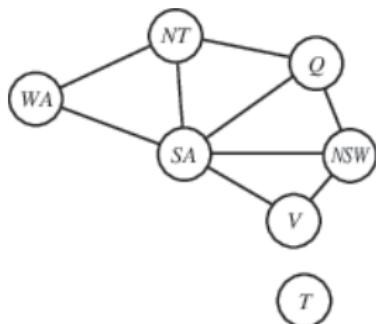
- ▶ **Definition 5.8. Cutset conditioning:** instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- ▶ Cutset size $c \sim$ runtime $\mathcal{O}(d^c \cdot (n - c)(d^2))$, very fast for small c

“Decomposition” 2.0: Acyclic Constraint Graphs

- ▶ **Theorem 5.9 (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 inconsistent, in time $\mathcal{O}(nk^2)$.
- ▶ Proof Sketch: See the algorithm on the next slide □
- ▶ Constraint networks with acyclic constraint graphs can be solved in (low-order) polynomial time.

“Decomposition” 2.0: Acyclic Constraint Graphs

- ▶ **Theorem 5.9 (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 inconsistent, in time $\mathcal{O}(nk^2)$.
- ▶ Proof Sketch: See the algorithm on the next slide □
- ▶ Constraint networks with acyclic constraint graphs can be solved in (low-order) polynomial time.
- ▶ **Example 5.10.** Australia is not acyclic. (But see next section)



- ▶ **Theorem 5.9 (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 inconsistent, in time $\mathcal{O}(nk^2)$.
- ▶ Proof Sketch: See the algorithm on the next slide □
- ▶ Constraint networks with acyclic constraint graphs can be solved in (low-order) polynomial time.
- ▶ **Example 5.10.** Australia is not acyclic. (But see next section)
- ▶ **Example 5.11 (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: 402^2 . Gain: 2^{32} .

Acyclic Constraint Graphs: How To

► **Definition 5.12.** Algorithm AcyclicCG(γ):

1. Obtain a directed tree from γ 's **constraint graph**, picking an arbitrary variable v as the root, and directing arcs outwards.¹

¹We assume here that γ 's **constraint graph** is **connected**. If it is not, do this and the following for each **component** separately.

► **Definition 5.12.** Algorithm AcyclicCG(γ):

1. Obtain a directed tree from γ 's **constraint graph**, picking an arbitrary variable v as the root, and directing arcs outwards.¹
2. Order the variables topologically, i.e., such that each vertex is ordered before its children; denote that order by v_1, \dots, v_n .

¹We assume here that γ 's **constraint graph** is **connected**. If it is not, do this and the following for each **component** separately.

Acyclic Constraint Graphs: How To

► **Definition 5.12.** Algorithm AcyclicCG(γ):

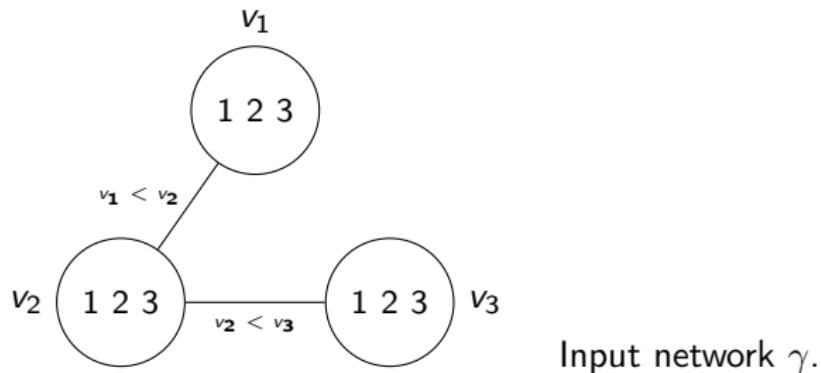
1. Obtain a directed tree from γ 's constraint graph, picking an arbitrary variable v as the root, and directing arcs outwards.¹
2. Order the variables topologically, i.e., such that each vertex is ordered before its children; denote that order by v_1, \dots, v_n .
3. **for** $i := n, n - 1, \dots, 2$ **do**:
 - 3.1 Revise($\gamma, v_{\text{parent}(i)}, v_i$).
 - 3.2 **if** $D_{v_{\text{parent}(i)}} = \emptyset$ **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, \dots, v_n .

► **Lemma 5.13.** This algorithm will find a solution without ever having to backtrack!

¹We assume here that γ 's constraint graph is connected. If it is not, do this and the following for each component separately.

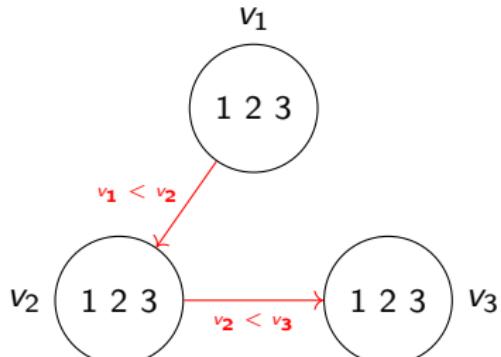
AcyclicCG(γ): Example

- ▶ **Example 5.14 (AcyclicCG() execution).**



AcyclicCG(γ): Example

- ▶ Example 5.14 (AcyclicCG() execution).

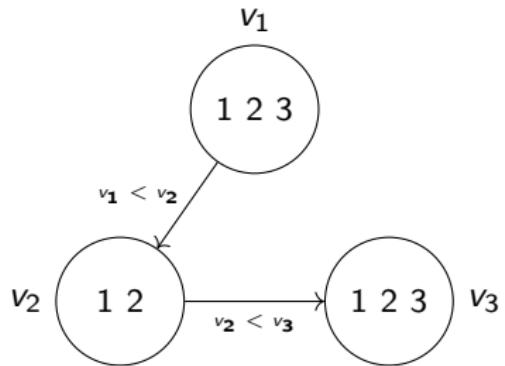


Step 2: Order v_1, v_2, v_3 .

Step 1: Directed tree for root v_1 .

AcyclicCG(γ): Example

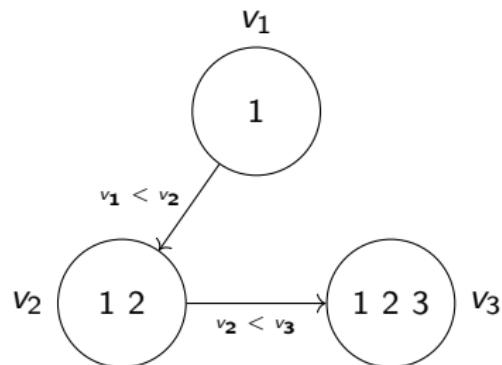
- ▶ Example 5.14 (AcyclicCG() execution).



Step 3: After $\text{Revise}(\gamma, v_2, v_3)$.

AcyclicCG(γ): Example

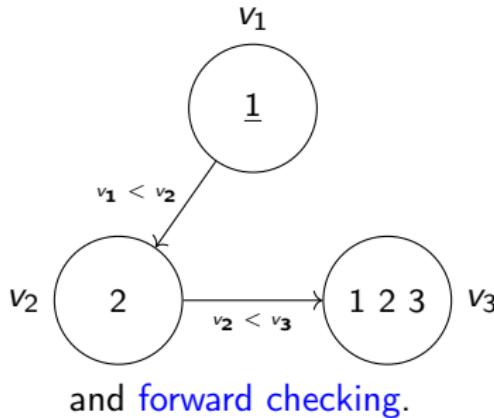
- ▶ Example 5.14 (AcyclicCG() execution).



Step 3: After $\text{Revise}(\gamma, v_1, v_2)$.

AcyclicCG(γ): Example

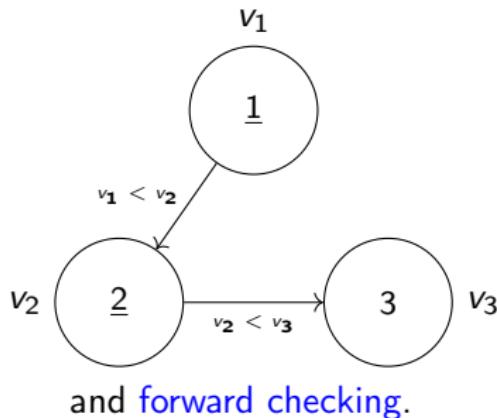
- ▶ Example 5.14 (AcyclicCG() execution).



Step 4: After $a(v_1) := 1$

AcyclicCG(γ): Example

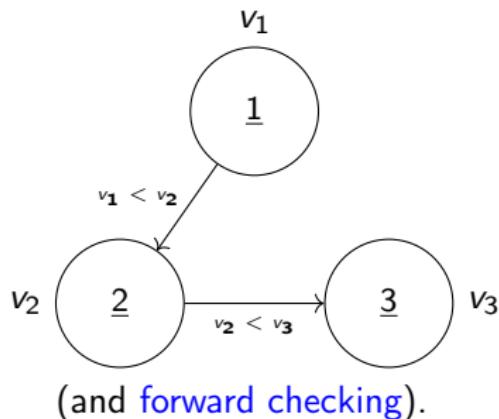
- ▶ Example 5.14 (AcyclicCG() execution).



Step 4: After $a(v_2) := 2$

AcyclicCG(γ): Example

- ▶ Example 5.14 (AcyclicCG() execution).

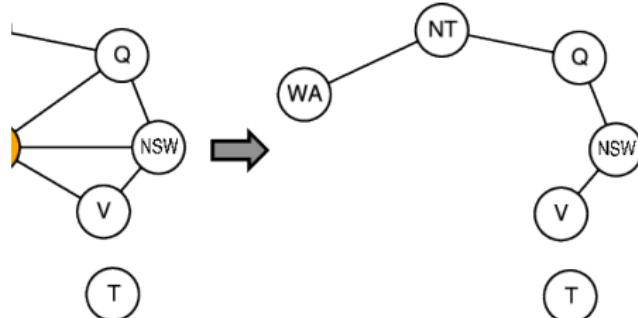


Step 4: After $a(v_3) := 3$

6 Cutset Conditioning

“Almost” Acyclic Constraint Graphs

► Example 6.1 (Coloring Australia).



► Cutset Conditioning: Idea:

1. Recursive call of backtracking on a s.t. the sub-graph of the **constraint graph** induced by $\{v \in V \mid a(v) \text{ is undefined}\}$ is **acyclic**.
 - Then we can solve the remaining sub-problem with **AcyclicCG()**.
2. Choose the variable order so that removing the first d variables renders the **constraint graph acyclic**.
 - Then with (1) we won't have to search deeper than $d \dots !$

“Decomposition” 3.0: Cutset Conditioning

- ▶ **Definition 6.2 (Cutset).** Let $\gamma = \langle V, D, C \rangle$ be a constraint network, and $V_0 \subseteq V$. Then V_0 is a **cutset** for γ if the sub-graph of γ 's constraint graph induced by $V \setminus V_0$ is acyclic. V_0 is called **optimal** if its size is minimal among all cutsets for γ .
- ▶ **Definition 6.3.** The **cutset conditioning** algorithm, computes an **optimal cutset**, from γ and an existing cutset V_0 .

```
function CutsetConditioning( $\gamma, V_0, a$ ) returns a solution, or “inconsistent”
     $\gamma' :=$  a copy of  $\gamma$ ;  $\gamma' :=$  ForwardChecking( $\gamma', a$ )
    if ex.  $v$  with  $D'_v = \emptyset$  then return “inconsistent”
    if ex.  $v \in V_0$  s.t.  $a(v)$  is undefined then select such  $v$ 
    else  $a' :=$  AcyclicCG( $\gamma'$ ); if  $a' \neq$  “inconsistent” then return  $a \cup a'$  else return “inconsistent”
    for each  $d \in$  copy of  $D'_v$  in some order do
         $a' := a \cup \{v = d\}$ ;  $D'_v := \{d\}$ ;
         $a'' :=$  CutsetConditioning( $\gamma', V_0, a'$ )
        if  $a'' \neq$  “inconsistent” then return  $a''$  else return “inconsistent”
```

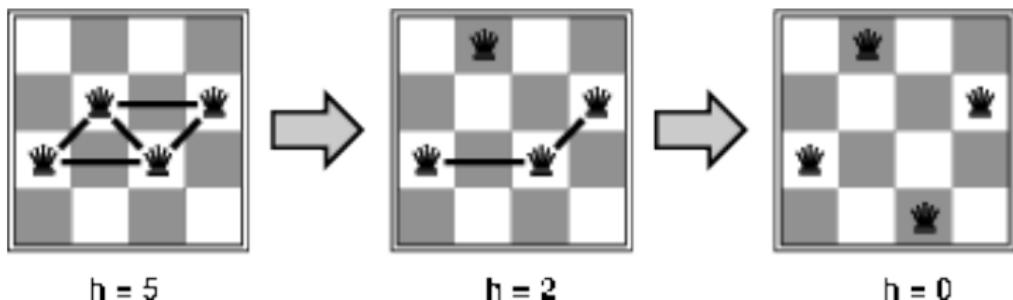
- ▶ Forward checking is required so that “ $a \cup$ AcyclicCG(γ')” is consistent in γ .
- ▶ **Observation 6.4.** Runtime is exponential only in $\#(V_0)$, not in $\#(V)!$
- ▶ **Remark 6.5.** Finding optimal cutsets is NP-hard, but approximations exist.

7 Constraint Propagation with Local Search

- ▶ Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned
- ▶ To apply to [CSPs](#): allow states with unsatisfied constraints operators *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., hillclimb with $h(n)$:=total number of violated constraints

Example: 4-Queens

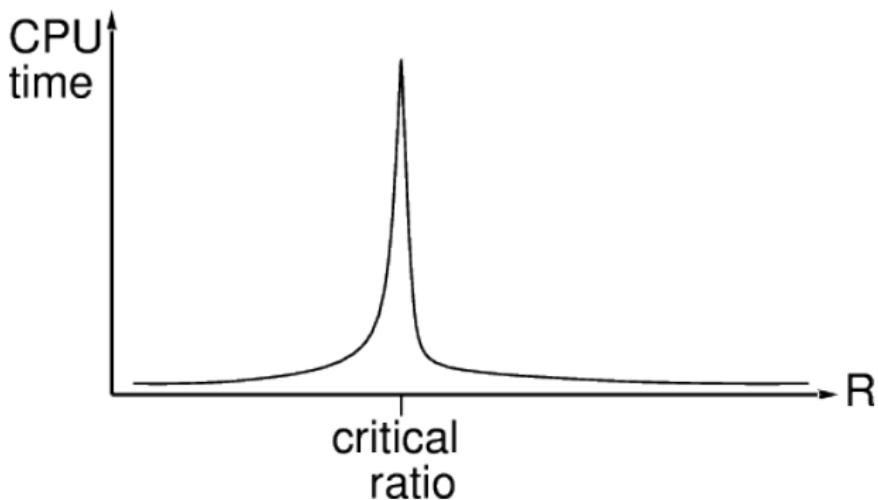
- ▶ States: 4 queens in 4 columns ($4^4 = 256$ states)
- ▶ Operators: move queen in column
- ▶ Goal test: no attacks
- ▶ Evaluation: $h(n) = \text{number of attacks}$



Performance of min-conflicts

- ▶ Given 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

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



8 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. This reduces the amount of search needed; that benefit must be traded off against the runtime 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 decomposition 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

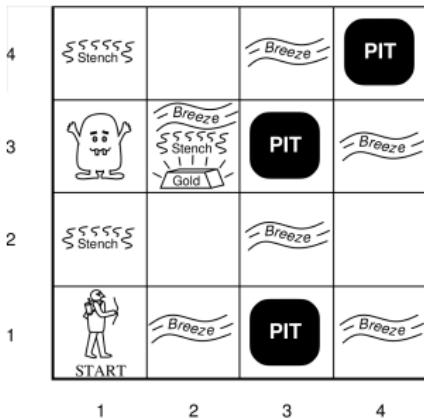
- ▶ **Path consistency, k -consistency:** Generalizes arc consistency to size- k subsets of variables. Path consistency $\hat{=}$ 3-consistency.
- ▶ **Tree decomposition:** Instead of instantiating variables until the leaf nodes are trees, distribute the variables and constraints over sub-CSPs whose connections form a tree.
- ▶ **Backjumping:** Like backtracking, but with ability to back up *across several levels* (to a previous assignment identified to be responsible for failure).
- ▶ **No-Good Learning:** Inferring additional constraints based on information gathered during backtracking.
- ▶ **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 polynomial time.
- ▶ **Global Constraints:** Constraints over many/all variables, with associated specialized inference methods.
- ▶ **Constraint Optimization Problems (COP):** Utility function over solutions, need an optimal one.

Part III Knowledge and Inference

Chapter 11 Propositional Reasoning, Part I: Principles

1 Introduction

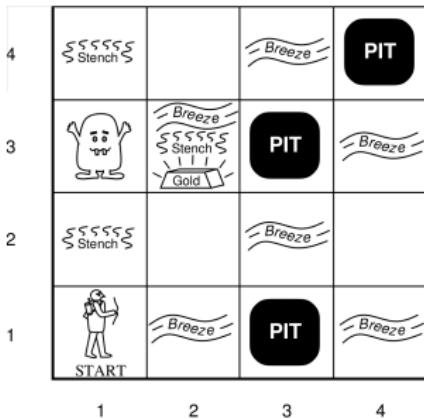
The Wumpus World



Definition 1.1. The **Wumpus world** is a simple game where an **agent** explores a cave with 16 **cells** that can contain **pits**, **gold**, and the **Wumpus** with the goal of getting back out alive with the **gold**.

- ▶ **Definition 1.2 (Actions).** The **agent** can perform the following **actions**: **goForward**, **turnRight** (by 90°), **turnLeft** (by 90°), **shoot** arrow in direction you're facing (you got exactly one arrow), **grab** an object in current cell, **leave** cave if you're in cell [1, 1].
- ▶ **Definition 1.3 (Initial and Terminal States).** Initially, the **agent** is in cell [1, 1] facing east. If the **agent** falls down a **pit** or meets live **Wumpus** it dies.

The Wumpus World



Definition 1.1. The **Wumpus world** is a simple game where an **agent** explores a cave with 16 **cells** that can contain **pits**, **gold**, and the **Wumpus** with the goal of getting back out alive with the **gold**.

- ▶ **Definition 1.4 (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**).

Reasoning in the Wumpus World

► Example 1.5 (Reasoning in the Wumpus World).

A: agent, V: visited, OK: safe, P: pit, W: Wumpus, B: breeze, S: stench, G: gold.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	2,1	3,1	4,1

(1) Initial state

Reasoning in the Wumpus World

► Example 1.5 (Reasoning in the Wumpus World).

A: agent, V: visited, OK: safe, P: pit, W: Wumpus, B: breeze, S: stench, G: gold.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A OK	2,1 OK	3,1	4,1

(1) Initial state

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1 V OK	2,1 A OK	3,1 B P?	4,1

(2) One step to right

Reasoning in the Wumpus World

► Example 1.5 (Reasoning in the Wumpus World).

A: agent, V: visited, OK: safe, P: pit, W: Wumpus, B: breeze, S: stench, G: gold.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A OK	2,1 OK	3,1 OK	4,1 OK

(1) Initial state

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

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1 V OK	2,1 A B OK	3,1 P?	4,1 OK

(2) One step to right

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1 OK

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

Reasoning in the Wumpus World

► Example 1.5 (Reasoning in the Wumpus World).

A: agent, V: visited, OK: safe, P: pit, W: Wumpus, B: breeze, S: stench, G: gold.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A OK	2,1 OK	3,1 OK	4,1 OK

(1) Initial state

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1 V OK	2,1 A OK	3,1 B OK	4,1 P?

(2) One step to right

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2 OK	4,2 OK
1,1 V OK	2,1 B OK	3,1 P! OK	4,1 OK

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

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

Reasoning in the Wumpus World

► Example 1.5 (Reasoning in the Wumpus World).

A: agent, V: visited, OK: safe, P: pit, W: Wumpus, B: breeze, S: stench, G: gold.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A OK	2,1 OK	3,1 OK	4,1 OK

(1) Initial state

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1 V OK	2,1 A OK	3,1 B OK	4,1 P?

(2) One step to right

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2 OK	4,2 OK
1,1 V OK	2,1 B V OK	3,1 P! OK	4,1 OK

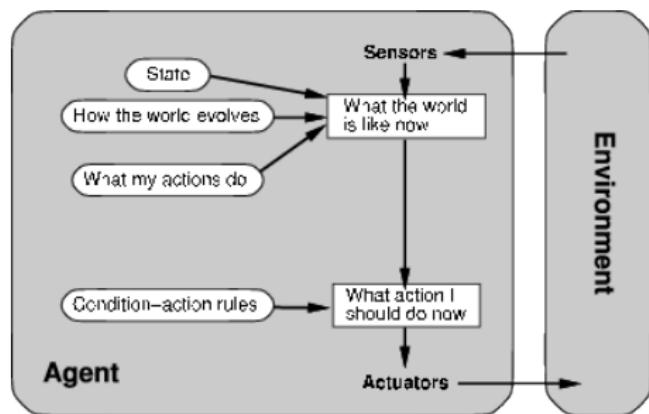
(3) Back, and up to [1,2]

- *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 [1,2], so the breeze in [2,1] can only come from [3,1].

Agents that Think Rationally

► Think Before You Act!:

A model based agent can think about the world and his actions



► Think Before You Act!:

A **model based agent** can think about the world and his 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*

Agents that Think Rationally

► Think Before You Act!:

A model based agent can think about the world and his 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*

► Idea: “Thinking” = Reasoning about knowledge represented using logic.

- ▶ **Definition 1.6.** **Syntax:** What are legal statements (**formulas**) A in the logic?
- ▶ **Example 1.7.** “W” and “ $W \Rightarrow S$ ”. ($W \hat{=} \text{Wumpus is here}$, $S \hat{=} \text{it stinks}$)

- ▶ **Definition 1.6.** **Syntax:** What are legal statements (**formulas**) A in the logic?
- ▶ **Example 1.7.** “W” and “ $W \Rightarrow S$ ”. ($W \hat{=} \text{Wumpus is here}$, $S \hat{=} \text{it stinks}$)
- ▶ **Definition 1.8.** **Semantics:** Which formulas A are true under which **assignment** φ , written $\varphi \models A$?
- ▶ **Example 1.9.** If $\varphi := \{W \mapsto T, S \mapsto F\}$, then $\varphi \models W$ but $\varphi \not\models (W \Rightarrow S)$.
- ▶ **Intuition:** Knowledge about the state of the world is described by **formulae**, **interpretations** evaluate them in the current world (they should turn out true!)

- ▶ **Definition 1.10.** **Entailment:** Which B are entailed by A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? E.g., $(P \wedge P \Rightarrow Q) \models Q$.
- ▶ **Intuition:** Entailment $\hat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information

- ▶ **Definition 1.10.** **Entailment:** Which B are entailed by A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? E.g., $(P \wedge P \Rightarrow Q) \models Q$.
- ▶ **Intuition:** Entailment $\hat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information
- ▶ **Definition 1.11. Deduction:** Which statements B can be derived from A using a set \mathcal{C} of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- ▶ **Example 1.12.** If \mathcal{C} contains $\frac{A \quad A \Rightarrow B}{B}$ then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$

- ▶ **Definition 1.10.** **Entailment:** Which B are entailed by A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? E.g., $(P \wedge P \Rightarrow Q) \models Q$.
- ▶ **Intuition:** Entailment $\hat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information
- ▶ **Definition 1.11. Deduction:** Which statements B can be derived from A using a set \mathcal{C} of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- ▶ **Example 1.12.** If \mathcal{C} contains
$$\frac{A \quad A \Rightarrow B}{B}$$
 then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$
- ▶ **Intuition:** Deduction $\hat{=}$ process in an actual computer trying to reason about entailment. E.g. a mechanical process attempting to determine Wumpus position.

- ▶ **Definition 1.10.** **Entailment:** Which B are entailed by A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? E.g., $(P \wedge P \Rightarrow Q) \models Q$.
- ▶ **Intuition:** Entailment $\hat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information
- ▶ **Definition 1.11. Deduction:** Which statements B can be derived from A using a set \mathcal{C} of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- ▶ **Example 1.12.** If \mathcal{C} contains
$$\frac{A \quad A \Rightarrow B}{B}$$
 then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$
- ▶ **Intuition:** Deduction $\hat{=}$ process in an actual computer trying to reason about entailment. E.g. a mechanical process attempting to determine Wumpus position.
- ▶ **Definition 1.13. Soundness:** whenever $A \vdash_{\mathcal{C}} B$, we also have $A \models B$.
- ▶ **Definition 1.14. Completeness:** whenever $A \models B$, we also have $A \vdash_{\mathcal{C}} B$.

- ▶ **Idea:** Any problem that can be formulated as reasoning about logic. \leadsto use off-the-shelf reasoning tool.
- ▶ Very successful using propositional logic and modern **SAT solvers!**
(**Propositional satisfiability testing;**)

- ▶ Propositional logic = canonical form of knowledge + reasoning.
 - ▶ Syntax: Atomic **propositions** that can be either true or false, connected by “**and, or, 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 CSP problems. (see)
- ▶ gives an example for verification.

Our Agenda for This Topic

- ▶ This document: Basic definitions and concepts; tableaux, resolution.
- ▶ Sets up the framework. Resolution is the quintessential reasoning procedure underlying most successful **SAT solvers**.
- ▶ : The Davis-Putnam procedure and clause learning; practical problem structure.
- ▶ State-of-the-art algorithms for reasoning about propositional logic, and an important observation about how they behave.

Our Agenda for This Chapter

- ▶ **Propositional Logic:** What's the syntax and semantics? How can we capture deduction?
 - ▶ We study this logic formally.
- ▶ **Tableaux, Resolution:** How can we make deduction mechanizable? What are its properties?
 - ▶ Formally introduces the most basic machine-oriented reasoning methods.
- ▶ **Killing a Wumpus:** How can we use all this to figure out where the Wumpus is?
 - ▶ Coming back to our introductory example.

2 Propositional Logic (Syntax/Semantics)

Propositional Logic (Syntax)

- ▶ **Definition 2.1 (Syntax).** The **formulae** of **propositional logic** (write PL^0) are made up from

- ▶ **propositional variables:** $\mathcal{V}_o := \{P, Q, R, P^1, P^2, \dots\}$ (countably infinite)
- ▶ constants/constructors called **connectives**: $\Sigma_o := \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \dots\}$

We define the set $wff_o(\mathcal{V}_o)$ of **well-formed propositional formulae (wffs)** as

- ▶ propositional variables,
- ▶ the logical constants T and F ,
- ▶ negations $\neg A$,
- ▶ conjunctions $A \wedge B$ (A and B are called **conjuncts**),
- ▶ disjunctions $A \vee B$ (A and B are called **disjuncts**),
- ▶ implications $A \Rightarrow B$, and
- ▶ equivalences (or **biimplications**). $A \Leftrightarrow B$,

where $A, B \in wff_o(\mathcal{V}_o)$ themselves.

- ▶ **Example 2.2.** $(P \wedge Q), (P \vee Q), (\neg P \vee Q) \Leftrightarrow (P \Rightarrow Q) \in wff_o(\mathcal{V}_o)$

- ▶ **Definition 2.3.** Propositional formulae without **connectives** are called **atomic** (or an **atoms**) and **complex** otherwise.

Alternative Notations for Connectives

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

- ▶ **Definition 2.4.** A **model** $\mathcal{M} := \langle \mathcal{D}_o, \mathcal{I} \rangle$ for propositional logic consists of
 - ▶ the **universe** $\mathcal{D}_o = \{\text{T}, \text{F}\}$
 - ▶ the **interpretation** \mathcal{I} that assigns values to essential **connectives**.
 - ▶ $\mathcal{I}(\neg) : \mathcal{D}_o \rightarrow \mathcal{D}_o; \text{T} \mapsto \text{F}, \text{F} \mapsto \text{T}$
 - ▶ $\mathcal{I}(\wedge) : \mathcal{D}_o \times \mathcal{D}_o \rightarrow \mathcal{D}_o; \langle \alpha, \beta \rangle \mapsto \text{T}$, iff $\alpha = \beta = \text{T}$
- ▶ We call a constructor a **logical constant**, iff its value is fixed by the interpretation
- ▶ Treat the other **connectives** as abbreviations, e.g. $\text{A} \vee \text{B} \hat{=} \neg(\neg \text{A} \wedge \neg \text{B})$ and $\text{A} \Rightarrow \text{B} \hat{=} \neg \text{A} \vee \text{B}$, and $\text{T} \hat{=} \text{P} \vee \neg \text{P}$ (only need to treat \neg, \wedge directly)

- ▶ Problem: The interpretation function only assigns meaning to connectives.
- ▶ Definition 2.5. A variable assignment $\varphi: \mathcal{V}_o \rightarrow \mathcal{D}_o$ assigns values to propositional variables.
- ▶ Definition 2.6. The value function $\mathcal{I}_\varphi: wff_o(\mathcal{V}_o) \rightarrow \mathcal{D}_o$ assigns values to PL⁰ formulae. It is recursively defined,
 - ▶ $\mathcal{I}_\varphi(P) = \varphi(P)$ (base case)
 - ▶ $\mathcal{I}_\varphi(\neg A) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(A))$.
 - ▶ $\mathcal{I}_\varphi(A \wedge B) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(A), \mathcal{I}_\varphi(B))$.
- ▶ Note that $\mathcal{I}_\varphi(A \vee B) = \mathcal{I}_\varphi(\neg(\neg A \wedge \neg B))$ is only determined by $\mathcal{I}_\varphi(A)$ and $\mathcal{I}_\varphi(B)$, so we think of the defined connectives as logical constants as well.

Computing Semantics

- Example 2.7. Let $\varphi := [T/P_1], [F/P_2], [T/P_3], [F/P_4]$ then

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

- What a mess!

Semantic Properties of Propositional Formulae

- ▶ **Definition 2.8.** Let $\mathcal{M} := \langle \mathcal{U}, \mathcal{I} \rangle$ be our model, then we call A
 - ▶ true under φ (φ satisfies A) in \mathcal{M} , iff $\mathcal{I}_\varphi(A) = T$ (write $\mathcal{M} \models^\varphi A$)
 - ▶ false under φ (φ falsifies A) in \mathcal{M} , iff $\mathcal{I}_\varphi(A) = F$ (write $\mathcal{M} \not\models^\varphi A$)
 - ▶ satisfiable in \mathcal{M} , iff $\mathcal{I}_\varphi(A) = T$ for some assignment φ
 - ▶ valid in \mathcal{M} , iff $\mathcal{M} \models^\varphi A$ for all assignments φ
 - ▶ falsifiable in \mathcal{M} , iff $\mathcal{I}_\varphi(A) = F$ for some assignments φ
 - ▶ unsatisfiable in \mathcal{M} , iff $\mathcal{I}_\varphi(A) = F$ for all assignments φ
- ▶ **Example 2.9.** $x \vee x$ is satisfiable and falsifiable.
- ▶ **Example 2.10.** $x \vee \neg x$ is valid and $x \wedge \neg x$ is unsatisfiable.
- ▶ Alternative Notation: Write $\mathcal{I}_\varphi(\text{[]})$ for $\mathcal{I}_\varphi(A)$, if $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$. (and $\llbracket A \rrbracket$, if A is ground, and $\llbracket \mathcal{A} \rrbracket$, if \mathcal{M} is clear)
- ▶ **Definition 2.11 (Entailment).** (aka. logical consequence) We say that A entails B ($A \models B$), iff $\mathcal{I}_\varphi(B) = T$ for all φ with $\mathcal{I}_\varphi(A) = T$ (i.e. all assignments that make A true also make B true)

A better mouse-trap: Truth Tables

- ▶ Truth tables to visualize truth functions:

\neg	\wedge		\vee	
	T	F	T	F
T	F		T	F
F	T		F	F

- ▶ If we are interested in values for all assignments (e.g. of $z \wedge x \vee \neg(z \wedge y)$)

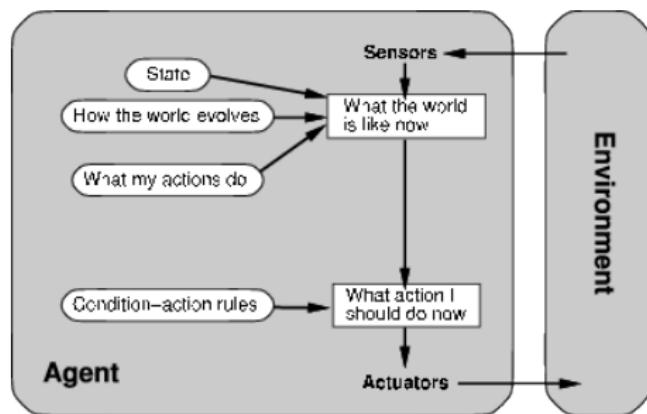
assignments			intermediate results			full
x	y	z	$e_1 := z \wedge y$	$e_2 := \neg e_1$	$e_3 := z \wedge x$	$e_3 \vee e_2$
F	F	F	F	T	F	T
F	F	T	F	T	F	T
F	T	F	F	T	F	T
F	T	T	T	F	F	F
T	F	F	F	T	F	T
T	F	T	F	T	T	T
T	T	F	F	T	F	T
T	T	T	T	F	T	T

3 Formal Systems (Syntax and Semantics in General)

Agents that Think Rationally

► Think Before You Act!:

A model based agent can think about the world and his actions



► Think Before You Act!:

A **model based agent** can think about the world and his 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*

Agents that Think Rationally

► Think Before You Act!:

A model based agent can think about the world and his 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*

► Idea: “Thinking” = Reasoning about knowledge represented using logic.

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	T	F	\downarrow	T	F
T	F	T	T	F	F
F	T	T	F	F	T

$\neg a$	$a \uparrow a$	$a \downarrow a$
ab	$a \uparrow b \uparrow a \uparrow b \uparrow$	$a \downarrow a \downarrow b \downarrow b$
ab	$a \uparrow a \uparrow b \uparrow b \uparrow$	$a \downarrow b \downarrow a \downarrow b$

- **Observation:** The set $wff_o(\mathcal{V}_o)$ of **well-formed propositional formulae** is a formal language over the **alphabet** given by \mathcal{V}_o , the connectives, and brackets.
- **Recall:** We are mostly interested in
 - **satisfiability** – i.e. whether $\mathcal{M} \models^\varphi A$, and
 - **entailment** – i.e. whether $A \models B$.
- **Observation:** In particular, the inductive/compositional nature of $wff_o(\mathcal{V}_o)$ and $\mathcal{I}_\varphi: wff_o(\mathcal{V}_o) \rightarrow \mathcal{D}_o$ are secondary.
- **Idea:** Concentrate on language, models (\mathcal{M}, φ) , and satisfiability.

- ▶ **Definition 3.1.** A **logical system** is a triple $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$, where \mathcal{L} is a **formal language**, \mathcal{K} is a **set** and $\models \subseteq \mathcal{K} \times \mathcal{L}$. Members of \mathcal{L} are called **formulas** of \mathcal{S} , members of \mathcal{K} **models** for \mathcal{S} , and \models the **satisfaction relation**.
- ▶ **Example 3.2 (Propositional Logic).**
 $\langle \text{wff}_o(\mathcal{V}_o), \mathcal{K}, \models \rangle$ is a **logical system**, if we define $\mathcal{K} := \mathcal{V}_o \rightarrow \mathcal{D}_o$ (the set of variable assignments) and $\varphi \models A : \equiv \mathcal{I}_\varphi(A) = T$.
- ▶ **Definition 3.3.** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a **logical system**, $\mathcal{M} \in \mathcal{K}$ be a **model** and $A \in \mathcal{L}$ a **formula**, then we say that A is
 - ▶ **satisfied** by \mathcal{M} , iff $\mathcal{M} \models A$.
 - ▶ **falsified** by \mathcal{M} , iff $\mathcal{M} \not\models A$.
 - ▶ **satisfiable** in \mathcal{K} , iff $\mathcal{M} \models A$ for some model $\mathcal{M} \in \mathcal{K}$.
 - ▶ **valid** in \mathcal{K} (write $\models \mathcal{M}$), iff $\mathcal{M} \models A$ for all models $\mathcal{M} \in \mathcal{K}$.
 - ▶ **falsifiable** in \mathcal{K} , iff $\mathcal{M} \not\models A$ for some $\mathcal{M} \in \mathcal{K}$.
 - ▶ **unsatisfiable** in \mathcal{K} , iff $\mathcal{M} \not\models A$ for all $\mathcal{M} \in \mathcal{K}$.

Derivation Relations and Inference Rules

- ▶ **Definition 3.4.** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a relation $\vdash \subseteq \mathcal{P}(\mathcal{L}) \times \mathcal{L}$ a **derivation relation** for \mathcal{S} , if it
 - ▶ is **proof reflexive**, i.e. $\mathcal{H} \vdash A$, if $A \in \mathcal{H}$;
 - ▶ is **proof transitive**, i.e. if $\mathcal{H} \vdash A$ and $\mathcal{H} \cup \{A\}' \vdash B$, then $\mathcal{H} \cup \mathcal{H}' \vdash B$;
 - ▶ **monotonic** (or **admits weakening**), i.e. $\mathcal{H} \vdash A$ and $\mathcal{H} \subseteq \mathcal{H}'$ imply $\mathcal{H}' \vdash A$.
- ▶ **Definition 3.5.** We call $\langle \mathcal{L}, \mathcal{K}, \models, \vdash \rangle$ a **formal system**, iff $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ is a logical system, and \vdash a derivation relation for \mathcal{S} .
- ▶ **Definition 3.6.** Let \mathcal{L} be the formal language of a logical system, then an **inference rule** over \mathcal{L} is a decidable $n + 1$ -ary relation on \mathcal{L} . Inference rules as traditionally written as

$$\frac{A_1 \quad \dots \quad A_n}{C} \mathcal{N}$$

where A_1, \dots, A_n and C are formula schemata for \mathcal{L} and \mathcal{N} is a name.
The A_i are called **assumption**, and C is called **conclusion**.

- ▶ **Definition 3.7.** An inference rule without assumptions is called an **axiom**.
- ▶ **Definition 3.8.** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a set \mathcal{C} of inference rules over \mathcal{L} a **calculus** for \mathcal{S} .

► **Definition 3.9.** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system and \mathcal{C} a calculus for \mathcal{S} , then a \mathcal{C} -derivation of a formula $C \in \mathcal{L}$ from a set $\mathcal{H} \subseteq \mathcal{L}$ of hypotheses (write $\mathcal{H} \vdash_{\mathcal{C}} C$) is a sequence A_1, \dots, A_m of \mathcal{L} -formulae, such that

- $A_m = C$, (derivation culminates in C)
- for all $1 \leq i \leq m$, either $A_i \in \mathcal{H}$, or (hypothesis)
- there is an inference rule $\frac{A_{l_1} \cdots A_{l_k}}{A_i}$ in \mathcal{C} with $l_j < i$ for all $j \leq k$. (rule application)

We can also see a derivation as a derivation tree, where the A_{l_j} are the children of the node A_k .

► **Example 3.10.**

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.

$$\frac{\frac{P \Rightarrow Q \Rightarrow P}{K} \quad P}{Q \Rightarrow P} MP$$

- ▶ **Observation 3.11.** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system and \mathcal{C} a calculus for \mathcal{S} , then the \mathcal{C} -derivation relation $\vdash_{\mathcal{C}}$ defined in 3.9 is a derivation relation in the sense of the definition above.
- ▶ Therefore we will sometimes also call $\langle \mathcal{L}, \mathcal{K}, \models, \mathcal{C} \rangle$ a **formal system**, iff $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ is a logical system, and \mathcal{C} a calculus for \mathcal{S} .
- ▶ **Definition 3.12.** Let \mathcal{C} be a calculus, then a \mathcal{C} -derivation $\emptyset \vdash_{\mathcal{C}} A$ is called a **proof** of A and if one exists (write $\vdash_{\mathcal{C}} A$) then A is called a \mathcal{C} -**theorem**.
- ▶ **Definition 3.13.** An inference rule \mathcal{I} is called **admissible** in \mathcal{C} , if the extension of \mathcal{C} by \mathcal{I} does not yield new theorems.
- ▶ **Definition 3.14.** An inference rule $\frac{A_1 \quad \dots \quad A_n}{C}$ is called **derivable** in \mathcal{C} , if there is a \mathcal{C} -derivation $\{A_1, \dots, A_n\} \vdash_{\mathcal{C}} C$.
- ▶ **Observation 3.15.** Derivable inference rules are admissible, but not the other way around.

A Simple Formal System: Prop. Logic with Hilbert-Calculus

- ▶ Formulae: built from propositional variables: $P, Q, R \dots$ and implication: \Rightarrow
- ▶ Semantics: $\mathcal{I}_\varphi(P) = \varphi(P)$ and $\mathcal{I}_\varphi(A \Rightarrow B) = T$, iff $\mathcal{I}_\varphi(A) = F$ or $\mathcal{I}_\varphi(B) = T$.
- ▶ **Definition 3.16.** The **Hilbert calculus** \mathcal{H}^0 consists of the inference rules:

$$\frac{}{P \Rightarrow Q \Rightarrow P} K$$

$$\frac{}{P \Rightarrow Q \Rightarrow R \Rightarrow P \Rightarrow Q \Rightarrow P \Rightarrow R} S$$

$$\frac{A \Rightarrow B \quad A}{B} MP$$

$$\frac{A}{[B/X]A} Subst$$

- ▶ **Example 3.17.** A \mathcal{H}^0 theorem $C \Rightarrow C$ and proof

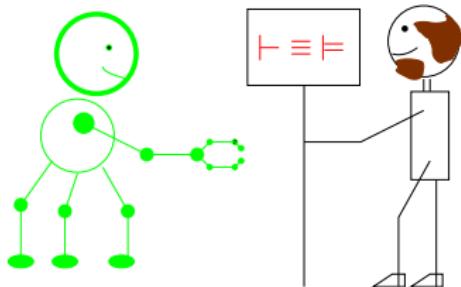
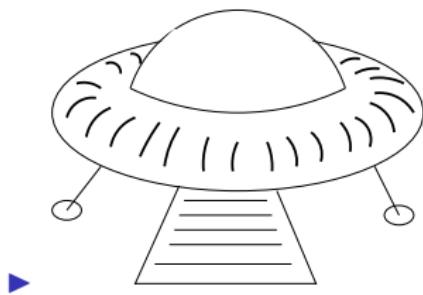
Proof: We show that $\emptyset \vdash_{\mathcal{H}^0} C \Rightarrow C$

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

□

Soundness and Completeness

- ▶ **Definition 3.18.** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a **logical system**, then we call a **calculus** \mathcal{C} for \mathcal{S}
 - ▶ **sound** (or **correct**), iff $\mathcal{H} \models \mathbb{A}$, whenever $\mathcal{H} \vdash_{\mathcal{C}} \mathbb{A}$, and
 - ▶ **complete**, iff $\mathcal{H} \vdash_{\mathcal{C}} \mathbb{A}$, whenever $\mathcal{H} \models \mathbb{A}$.
- ▶ **Goal:** Find **calculi** \mathcal{C} , such that $\vdash_{\mathcal{C}} \mathbb{A}$ iff $\models \mathbb{A}$ (**provability and validity coincide**)
 - ▶ To TRUTH through PROOF (CALCULEMUS [Leibniz ~1680])



The miracle of logics

- Purely formal derivations are true in the real world!

World of Logics

$\forall x (\text{human } x \rightarrow \text{mortal } x)$

\wedge

human Socrates

mortal Socrates

\Downarrow

Real World



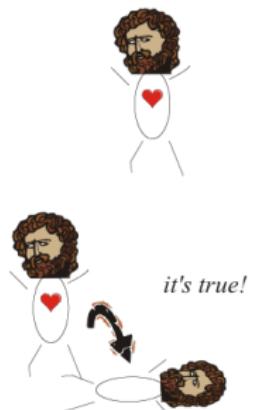
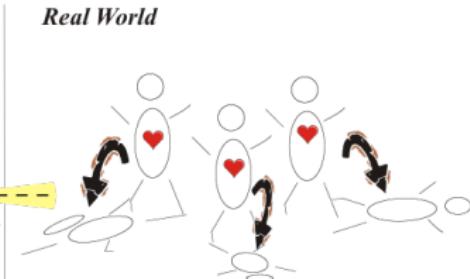
it's true!



it's true!



it must be true --
it's proven!



4 Propositional Natural Deduction Calculus

- Idea: \mathcal{ND}^0 tries to mimic human theorem proving behavior (non-minimal)
- **Definition 4.1.** The propositional natural deduction calculus \mathcal{ND}^0 has rules for the introduction and elimination of connectives

Introduction	Elimination	Axiom
$\frac{A \quad B}{A \wedge B} \wedge I$	$\frac{A \wedge B}{A} \wedge E_I \quad \frac{A \wedge B}{B} \wedge E_r$	
$\frac{[A]^1}{\equiv}$		
$\frac{B}{A \Rightarrow B} \Rightarrow I^1$	$\frac{A \Rightarrow B \quad A}{B} \Rightarrow E$	$\overline{A \vee \neg A} \text{ TND}$

- TND is used only in classical logic (otherwise constructive/intuitionistic)

► Example 4.2 (Inference with Local Hypotheses).

$$\frac{\frac{[A \wedge B]^1}{B} \wedge E_r \quad \frac{[A \wedge B]^1}{A} \wedge E_l}{\frac{B \wedge A}{A \wedge B \Rightarrow B \wedge A}} \wedge I$$
$$\frac{[A]^1 \quad [B]^2}{\frac{A}{B \Rightarrow A}} \Rightarrow I^2$$
$$\frac{A \quad B \Rightarrow A}{A \Rightarrow B \Rightarrow A} \Rightarrow I^1$$

A Deduction Theorem for ND^0

- ▶ **Theorem 4.3.** $\mathcal{H}, A \vdash_{\text{ND}^0} B$, iff $\mathcal{H} \vdash_{\text{ND}^0} A \Rightarrow B$.
- ▶ **Proof:** We show the two directions separately
 1. If $\mathcal{H}, A \vdash_{\text{ND}^0} B$, then $\mathcal{H} \vdash_{\text{ND}^0} A \Rightarrow B$ by $\Rightarrow I$, and
 2. If $\mathcal{H} \vdash_{\text{ND}^0} A \Rightarrow B$, then $\mathcal{H}, A \vdash_{\text{ND}^0} A \Rightarrow B$ by weakening and $\mathcal{H}, A \vdash_{\text{ND}^0} B$ by $\Rightarrow E$.

□

More Rules for Natural Deduction

- **Definition 4.4.** \mathcal{ND}^0 has the following additional rules for the remaining connectives.

$$\frac{\begin{array}{c} A \\ \hline A \vee B \end{array} \textcolor{blue}{\vee I_l} \quad \begin{array}{c} B \\ \hline A \vee B \end{array} \textcolor{blue}{\vee I_r}}{A \vee B}$$
$$\frac{\begin{array}{c} A \vee B \\ \vdots \\ C \end{array} \quad \begin{array}{c} C \\ \hline C \end{array}}{C} \textcolor{blue}{\vee E^1}$$

$$\frac{\begin{array}{c} [A]^1 \quad [A]^1 \\ \vdots \quad \vdots \\ C \quad \neg C \end{array} \textcolor{blue}{\neg I^1}}{\neg A} \quad \frac{\neg \neg A}{A} \textcolor{blue}{\neg E}$$

$$\frac{\begin{array}{c} \neg A \quad A \\ \hline F \end{array} \textcolor{blue}{FI}}{A} \textcolor{blue}{FE}$$

Natural Deduction in Sequent Calculus Formulation

- ▶ Idea: Represent hypotheses explicitly. (lift calculus to judgments)
- ▶ Definition 4.5. A **judgment** is a meta-statement about the provability of propositions.
- ▶ Definition 4.6. A **sequent** is a **judgment** of the form $\mathcal{H} \vdash A$ about the provability of the formula A from the set \mathcal{H} of hypotheses. We write $\vdash A$ for $\emptyset \vdash A$.
- ▶ Idea: Reformulate ND rules so that they act on sequents.
- ▶ Example 4.7. We give the **sequent**-style version of 4.2:

$$\begin{array}{c} \frac{\text{Ax}}{A \wedge B \vdash A \wedge B} \wedge E_r \quad \frac{\text{Ax}}{A \wedge B \vdash A \wedge B} \wedge E_l \\ \hline \frac{A \wedge B \vdash B}{A \wedge B \vdash B \wedge A} \wedge I \\ \hline \frac{}{\vdash A \wedge B \Rightarrow B \wedge A} \Rightarrow I \\ \\ \frac{\text{Ax}}{A, B \vdash A} \Rightarrow I \\ \hline \frac{A \vdash B \Rightarrow A}{\vdash A \Rightarrow B \Rightarrow A} \Rightarrow I \end{array}$$

- ▶ Note: Even though the antecedent of a sequent is written like a sequence, it is actually a set. In particular, we can permute and duplicate members at will.

Sequent-Style Rules for Natural Deduction

- **Definition 4.8.** The following inference rules make up the **propositional sequent-style natural deduction calculus** \mathcal{ND}^0 :

$$\frac{}{\Gamma, A \vdash A} \text{Ax}$$

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B} \text{weaken}$$

$$\frac{}{\Gamma \vdash A \vee \neg A} \text{TND}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge I$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge E_l$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge E_r$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee l_l$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee l_r$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \vee E$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow I$$

$$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow E$$

$$\frac{\Gamma, A \vdash F}{\Gamma \vdash \neg A} \neg I$$

$$\frac{\Gamma \vdash \neg \neg A}{\Gamma \vdash A} \neg E$$

$$\frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash F} FI$$

$$\frac{\Gamma \vdash F}{\Gamma \vdash A} FE$$

Linearized Notation for (Sequent-Style) ND Proofs

- ▶ Linearized notation for sequent-style ND proofs

1. $\mathcal{H}_1 \vdash A_1 (\mathcal{J}_1)$
2. $\mathcal{H}_2 \vdash A_2 (\mathcal{J}_2)$
3. $\mathcal{H}_3 \vdash A_3 (\mathcal{R}1, 2)$

corresponds to

$$\frac{\mathcal{H}_1 \vdash A_1 \quad \mathcal{H}_2 \vdash A_2}{\mathcal{H}_3 \vdash A_3} \mathcal{R}$$

- ▶ **Example 4.9.** We show a linearized version of 4.7

#	hyp	\vdash	formula	NDjust	#	hyp	\vdash	formula	NDjust
1.	1	\vdash	$A \wedge B$	Ax	1.	1	\vdash	A	Ax
2.	1	\vdash	B	$\wedge E_r 1$	2.	2	\vdash	B	Ax
3.	1	\vdash	A	$\wedge E_l 1$	3.	1, 2	\vdash	A	$\text{weaken } 1, 2$
4.	1	\vdash	$B \wedge A$	$\wedge I 2, 3$	4.	1	\vdash	$B \Rightarrow A$	$\Rightarrow I 3$
5.		\vdash	$A \wedge B \Rightarrow B \wedge A$	$\Rightarrow I 4$	5.		\vdash	$A \Rightarrow B \Rightarrow A$	$\Rightarrow I 4$

5 Machine-Oriented Calculi for Propositional Logic

- ▶ Recall: Our knowledge of the cave entails a definite Wumpus position! (slide 284)
- ▶ Problem: That was human reasoning, can we build an agent function that does this?
- ▶ Answer: As for constraint networks, we use inference, here resolution/tableau

- ▶ **Theorem 5.1 (Unsatisfiability Theorem).** $\mathcal{H} \models A$ iff $\mathcal{H} \cup \{\neg A\}$ is unsatisfiable.
- ▶ Proof:
 1. We prove both directions separately
 - 1.1. “ \Rightarrow ”: Say $\mathcal{H} \models A$: For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \models A$ and thus $\varphi \not\models (\neg A)$. □
 - 1.2. “ \Leftarrow ”: Say $\mathcal{H} \cup \{\neg A\}$ is unsatisfiable.: For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \not\models (\neg A)$ and thus $\varphi \models A$. □
 - ▶ **Observation 5.2.** Entailment can be tested via satisfiability.

Test Calculi: A Paradigm for Automating Inference

- ▶ **Definition 5.3.** Given a formal system $\langle \mathcal{L}, \mathcal{K}, \models, \mathcal{C} \rangle$, the task of **theorem proving** consists in determining whether a **conjecture** $A \in \mathcal{L}$ is a \mathcal{C} -theorem.
- ▶ **Definition 5.4.** The task of **automated theorem proving (ATP)** consists of developing programs that given a set $\mathcal{H} \subseteq \mathcal{L}$ and a **conjecture** A determine whether $\mathcal{H} \vdash_{\mathcal{C}} A$.
- ▶ Idea: ATP induces a **search problem** $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, where the **states** \mathcal{S} are sets of formulae in \mathcal{L} , the **actions** \mathcal{A} are the **inference rules** from \mathcal{C} , the **initial state** $\mathcal{I} = \{\mathcal{H}\}$, and the **goal states** are those with $A \in \mathcal{S}$.
- ▶ Problem: ATP as a **search problem** does not admit good **heuristics**, since these need to take the **conjecture** A into account.
- ▶ Idea: Turn the search around – using the unsatisfiability theorem (from above).
- ▶ **Definition 5.5.** A **test calculus** that for a given **conjecture** \mathcal{A} and hypotheses \mathcal{H} tries to derive an obviously unsatisfiable formula \perp .
- ▶ Intuition: Instead of showing $\emptyset \vdash T$, show $\neg T \vdash \perp$.
- ▶ Observation: A **test calculus** \mathcal{C} induces a **search problem** where the initial state is $\mathcal{H} \cup \{\neg A\}$ $S \in \mathcal{S}$ is a goal state iff $\perp \in S$.

- ▶ There are two quintessential normal forms for propositional formulae: (**there are others as well**)
- ▶ **Definition 5.6.** A formula is in **conjunctive normal form (CNF)** if it consists of a conjunction of disjunctions of literals: $\bigwedge_{i=1}^n \bigvee_{j=1}^{m_i} l_{i,j}$
- ▶ **Definition 5.7.** A formula is in **disjunctive normal form (DNF)** if it consists of a disjunction of conjunctions of literals: $\bigvee_{i=1}^n \bigwedge_{j=1}^{m_i} l_{i,j}$
- ▶ **Observation 5.8.** Every formula has equivalent formulas in CNF and DNF.

5.1 Calculi for Automated Theorem Proving: Analytical Tableaux

5.1.1 Analytical Tableaux

- ▶ **Definition 5.9.** We call a formula **atomic**, or an **atom**, iff it does not contain **connectives**. We call a formula **complex**, iff it is not atomic.
- ▶ **Definition 5.10.** We call a pair A^α a **labeled formula**, if $\alpha \in \{T, F\}$. A labeled atom A^α is called a (**positive** if $\alpha = T$, else **negative**) **literal**.
- ▶ **Intuition:** To satisfy a formula, we make it “true”. To satisfy a labeled formula A^α , it must have the truth value α .
- ▶ **Definition 5.11.** For a literal A^α , we call the literal A^β with $\alpha \neq \beta$ the **opposite literal** (or **partner literal**).
- ▶ **Definition 5.12.** Let Φ be a set of formulae, then we use $\Phi^\alpha := \{A^\alpha \mid A \in \Phi\}$.

- ▶ Note: **Literals** are often defined without recurring to **labeled formulae**:
- ▶ **Definition 5.13.** A **literal** is an **atoms** A (**positive literal**) or negated **atoms** $\neg A$ (**negative literal**). A and $\neg A$ are **opposite literals**
- ▶ Note: This notion of **literal** is equivalent to the **labeled formula**-notion of **literal**, but does not generalize as well to logics with more truth values.

Test Calculi: Tableaux and Model Generation

- ▶ Idea: A tableau calculus is a test calculus that
 - ▶ analyzes a labeled formula in a tree to determine satisfiability,
 - ▶ its branches correspond to valuations (\sim models).
 - ▶ Example 5.14. Tableau calculi try to construct models for labeled formulae:

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

- ▶ **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)
 - ▶ **Satisfiable**, iff there are open branches (correspond to models)

Analytical Tableaux (Formal Treatment of \mathcal{T}_0)

- Idea: A test calculus where
 - A labeled formula is analyzed in a tree to determine satisfiability,
 - branches correspond to valuations (models)
- Definition 5.15. The propositional tableau calculus \mathcal{T}_0 has two inference rules per connective (one for each possible label)

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

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

- Definition 5.16. We call any tree ($|$ introduces branches) produced by the ProbTabCalc inference rules from a set Φ of labeled formulae a **tableau** for Φ .
- Definition 5.17. Call a tableau **saturated**, iff no rule applies, and a branch **closed**, iff it ends in \perp , else **open**.
- Idea: Open branches in saturated tableaux yield models.
- Definition 5.18 (\mathcal{T}_0 -Theorem/Derivability). A is a \mathcal{T}_0 -theorem ($\vdash_{\mathcal{T}_0} A$), iff there is a closed tableau with A^F at the root.

$\Phi \subseteq wff_o(\mathcal{V}_o)$ derivation relation A in \mathcal{T}_0 ($\Phi \vdash_{\mathcal{T}_0} A$), iff there is a closed tableau starting with Φ and ending with A .

A Valid Real-World Example

► **Example 5.19.** If Mary loves Bill and John loves Mary, then John loves Mary

$$\begin{aligned} & (\text{loves(mary, bill)} \wedge \text{loves(john, mary)} \Rightarrow \text{loves(john, mary)})^F \\ & \neg(\neg(\text{loves(mary, bill)} \wedge \text{loves(john, mary)}) \wedge \neg\text{loves(john, mary)})^F \\ & (\neg(\text{loves(mary, bill)} \wedge \text{loves(john, mary)}) \wedge \neg\text{loves(john, mary)})^T \\ & \quad \neg(\text{loves(mary, bill)} \wedge \text{loves(john, mary)})^T \\ & \quad \neg(\text{loves(mary, bill)} \wedge \text{loves(john, mary)})^F \\ & \quad (\text{loves(mary, bill)} \wedge \text{loves(john, mary)})^T \\ & \quad \neg\text{loves(john, mary)}^T \\ & \quad \text{loves(mary, bill)}^T \\ & \quad \text{loves(john, mary)}^T \\ & \quad \text{loves(john, mary)}^F \\ & \quad \perp \end{aligned}$$

This is a closed tableau, so the

$\text{loves(mary, bill)} \wedge \text{loves(john, mary)} \Rightarrow \text{loves(john, mary)}$ is a \mathcal{T}_0 -theorem.

As we will see, \mathcal{T}_0 is sound and complete, so

$$\text{loves(mary, bill)} \wedge \text{loves(john, mary)} \Rightarrow \text{loves(john, mary)}$$

is valid.

Deriving Entailment in \mathcal{T}_0

- **Example 5.20.** *Mary loves Bill* and *John loves Mary* together entail that *John loves Mary*

$$\begin{array}{c} \text{loves(mary, bill)}^T \\ \text{loves(john, mary)}^T \\ \text{loves(john, mary)}^F \\ \perp \end{array}$$

This is a closed tableau, so the

$\{\text{loves(mary, bill)}, \text{loves(john, mary)}\} \vdash_{\mathcal{T}_0} \text{loves(john, mary)}$, again, as \mathcal{T}_0 is sound and complete we have

$$\{\text{loves(mary, bill)}, \text{loves(john, mary)}\} \models \text{loves(john, mary)}$$

A Falsifiable Real-World Example

- **Example 5.21.** * *If Mary loves Bill or John loves Mary, then John loves Mary*
Try proving the implication (this fails)

$$\begin{aligned} & (\text{loves(mary, bill)} \vee \text{loves(john, mary)}) \Rightarrow \text{loves(john, mary))}^F \\ & \neg(\neg(\text{loves(mary, bill)} \vee \text{loves(john, mary)}) \wedge \neg \text{loves(john, mary)})^F \\ & (\neg(\text{loves(mary, bill)} \vee \text{loves(john, mary)}) \wedge \neg \text{loves(john, mary)})^T \\ & \quad \neg \text{loves(john, mary)}^T \\ & \quad \text{loves(john, mary)}^F \\ & \neg \neg(\text{loves(mary, bill)} \vee \text{loves(john, mary)})^T \\ & \neg(\text{loves(mary, bill)} \vee \text{loves(john, mary)})^F \\ & (\text{loves(mary, bill)} \vee \text{loves(john, mary)})^T \\ & \text{loves(mary, bill)}^T \quad \text{loves(john, mary)}^T \\ & \qquad \qquad \bot \end{aligned}$$

Indeed we can make $\mathcal{I}_\varphi(\text{loves(mary, bill)}) = T$ but $\mathcal{I}_\varphi(\text{loves(john, mary)}) = F$.

- **Example 5.22.** Does *Mary loves Bill or John loves Mary* entail that *John loves Mary*?

$$\begin{array}{c} (\text{loves}(\text{mary}, \text{bill}) \vee \text{loves}(\text{john}, \text{mary}))^T \\ \text{loves}(\text{john}, \text{mary})^F \\ \text{loves}(\text{mary}, \text{bill})^T \quad \mid \quad \text{loves}(\text{john}, \text{mary})^T \\ \perp \end{array}$$

This saturated tableau has an open branch that shows that the interpretation with $\mathcal{I}_\varphi(\text{loves}(\text{mary}, \text{bill})) = T$ but $\mathcal{I}_\varphi(\text{loves}(\text{john}, \text{mary})) = F$ falsifies the derivability/entailment conjecture.

5.1.2 Practical Enhancements for Tableaux

Propositional Identities

- ▶ **Definition 5.23.** Let T and F be new logical constants with $\mathcal{I}(T) = T$ and $\mathcal{I}(F) = F$ for all assignments φ .
- ▶ We have the following identities:

Name	for \wedge	for \vee
Idenpotence	$\varphi \wedge \varphi = \varphi$	$\varphi \vee \varphi = \varphi$
Identity	$\varphi \wedge T = \varphi$	$\varphi \vee F = \varphi$
Absorption I	$\varphi \wedge F = F$	$\varphi \vee T = T$
Commutativity	$\varphi \wedge \psi = \psi \wedge \varphi$	$\varphi \vee \psi = \psi \vee \varphi$
Associativity	$\varphi \wedge \psi \wedge \theta = \varphi \wedge \psi \wedge \theta$	$\varphi \vee \psi \vee \theta = \varphi \vee \psi \vee \theta$
Distributivity	$\varphi \wedge (\psi \vee \theta) = \varphi \wedge \psi \vee \varphi \wedge \theta$	$\varphi \vee \psi \wedge \theta = (\varphi \vee \psi) \wedge (\varphi \vee \theta)$
Absorption II	$\varphi \wedge (\varphi \vee \theta) = \varphi$	$\varphi \vee \varphi \wedge \theta = \varphi$
De Morgan's Laws	$\neg(\varphi \wedge \psi) = \neg\varphi \vee \neg\psi$	$\neg(\varphi \vee \psi) = \neg\varphi \wedge \neg\psi$
Double negation		$\neg\neg\varphi = \varphi$
Definitions	$\varphi \Rightarrow \psi = \neg\varphi \vee \psi$	$\varphi \Leftrightarrow \psi = (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$

Derived Rules of Inference

- ▶ **Definition 5.24.** Let \mathcal{C} be a calculus, a rule of inference $\frac{A_1 \cdots A_n}{C}$ is called a **derived rule** in \mathcal{C} , iff there is a \mathcal{C} -proof of $A_1, \dots, A_n \vdash_{\mathcal{C}} C$.
- ▶ **Definition 5.25.** We have the following derived rules of inference

$$\begin{array}{c} \begin{array}{ccc} \dfrac{(A \Rightarrow B)^T}{A^F \quad | \quad B^T} & \dfrac{(A \Rightarrow B)^F}{A^T \quad | \quad B^F} & \dfrac{A^T}{(A \Rightarrow B)^T} \\[10pt] \dfrac{(A \vee B)^T}{A^T \quad | \quad B^T} & \dfrac{(A \vee B)^F}{A^F \quad | \quad B^F} & \dfrac{A \Leftrightarrow B^T}{A^T \quad | \quad A^F} \end{array} & \begin{array}{c} A^T \\ (\neg A \vee B)^T \\ (\neg(\neg A \wedge \neg B))^T \\ (\neg(\neg A \wedge \neg B))^F \\ \neg A^F \quad | \quad \neg B^F \\ \neg A^T \quad | \quad B^T \\ A^F \quad | \quad \perp \end{array} \end{array}$$

Tableaux with derived Rules (example)

Example 5.26.

$$\begin{array}{c} (\text{loves(mary, bill)} \wedge \text{loves(john, mary)} \Rightarrow \text{loves(john, mary)})^F \\ (\text{loves(mary, bill)} \wedge \text{loves(john, mary)})^T \\ \quad \text{loves(john, mary)}^F \\ \quad \text{loves(mary, bill)}^T \\ \quad \text{loves(john, mary)}^T \\ \perp \end{array}$$

5.1.3 Soundness and Termination of Tableaux

Soundness (Tableau)

- ▶ Idea: A test calculus is sound, iff it preserves satisfiability and the goal formulae are unsatisfiable.
- ▶ **Definition 5.27.** A labeled formula A^α is valid under φ , iff $\mathcal{L}_\varphi(A) = \alpha$.
- ▶ **Definition 5.28.** A tableau \mathcal{T} is satisfiable, iff there is a satisfiable branch \mathcal{P} in \mathcal{T} , i.e. if the set of formulae in \mathcal{P} is satisfiable.
- ▶ **Lemma 5.29.** Tableau rules transform satisfiable tableaux into satisfiable ones.
- ▶ **Theorem 5.30 (Soundness).** A set Φ of propositional formulae is valid, if there is a closed tableau \mathcal{T} for Φ^F .
- ▶ **Proof:** by contradiction: Suppose Φ is not valid.
 1. then the initial tableau is satisfiable (Φ^F satisfiable)
 2. so \mathcal{T} is satisfiable, by our Lemma. (by definition)
 3. there is a satisfiable branch (\mathcal{T} closed)
 4. but all branches are closed



- ▶ **Lemma 5.31.** The tableau procedure terminates, i.e. after a finite set of rule applications, it reaches a tableau, so that applying the tableau rules will only add labeled formulae that are already present on the branch.
- ▶ **Definition 5.32.** Let us call a labeled formulae A^α **worked off** in a tableau \mathcal{T} , if a tableau rule has already been applied to it.
- ▶ **Proof:**
 1. It is easy to see that applying rules to worked off formulae will only add formulae that are already present in its branch.
 2. Let $\mu(\mathcal{T})$ be the number of **connectives** in labeled formulae in \mathcal{T} that are not worked off.
 3. 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)
 4. At some point the tableau only contains worked off formulae and literals.
 5. Since there are only finitely many literals in \mathcal{T} , so we can only apply the tableau cut rule a finite number of times.



5.2 Resolution for Propositional Logic

- ▶ **Definition 5.33 (Resolution Calculus).** The **resolution calculus** operates on clause sets via a single inference rule:

$$\frac{P^T \vee A \quad P^F \vee B}{A \vee B}$$

This rule allows to add the clause below the line to a clause set which contains the two clauses above.

- ▶ **Definition 5.34 (Resolution Refutation).** Let S be a **clause set**, then we call a \mathcal{R} derivation $D: S \not\vdash_{\mathcal{R}} \square$ **resolution refutation**.
- ▶ **Definition 5.35 (Resolution Proof).** We call a **resolution refutation** of $CNF^0(A^F)$ a **resolution proof** for $A \in wff_o(\mathcal{V}_o)$.

Clause Normal Form Transformation (A calculus)

- ▶ **Definition 5.36.** A **clause** is a disjunction of literals. We will use \square for the “empty” disjunction (no disjuncts) and call it the **empty clause**.
- ▶ **Definition 5.37.** We will often write a **clause set** $\{C_1, \dots, C_n\}$ as $C_1; \dots; C_n$, use $S; T$ for the union of the clause sets S and T , and $S; C$ for the extension by a clause C .
- ▶ **Definition 5.38 (Transformation into Clause Normal Form).** The **CNF transformation calculus** CNF^0 consists of the following four inference rules on sets of labeled formulae.

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

- ▶ **Definition 5.39.** We write $CNF^0(A^\alpha)$ for the set of all clauses derivable from A^α via the rules above.

Derived Rules of Inference

- **Definition 5.40.** Let \mathcal{C} be a calculus, a rule of inference $\frac{A_1 \quad \dots \quad A_n}{C}$ is called a **derived inference rule** in \mathcal{C} , iff there is a \mathcal{C} -proof $A_1, \dots, A_n \vdash_{\mathcal{C}} C$.

$$\frac{\begin{array}{c} C \vee (A \Rightarrow B)^T \\ \hline C \vee (\neg A B)^T \end{array}}{C \vee \neg A^T \vee B^T} \rightsquigarrow \frac{C \vee (A \Rightarrow B)^T}{C \vee A^F \vee B^T}$$

- **Example 5.41.**

- **Others:**

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

Example: Proving Axiom S

► Example 5.42. Clause Normal Form transformation

$$\begin{array}{c} \frac{(P \Rightarrow Q \Rightarrow R \Rightarrow P \Rightarrow Q \Rightarrow P \Rightarrow R)^F}{(P \Rightarrow Q \Rightarrow R)^T; (P \Rightarrow Q \Rightarrow P \Rightarrow R)^F} \\ \frac{}{P^F \vee (Q \Rightarrow R)^T; (P \Rightarrow Q)^T; (P \Rightarrow R)^F} \\ \frac{}{P^F \vee Q^F \vee R^T; P^F \vee Q^T; P^T; R^F} \end{array}$$

Result $\{P^F \vee Q^F \vee R^T, P^F \vee Q^T, P^T, R^F\}$

► Example 5.43. Resolution Proof

1	$P^F \vee Q^F \vee R^T$	initial
2	$P^F \vee Q^T$	initial
3	P^T	initial
4	R^F	initial
5	$P^F \vee Q^F$	resolve 1.3 with 4.1
6	Q^F	resolve 5.1 with 3.1
7	P^F	resolve 2.2 with 6.1
8	\square	resolve 7.1 with 3.1

6 Killing a Wumpus with Propositional Inference

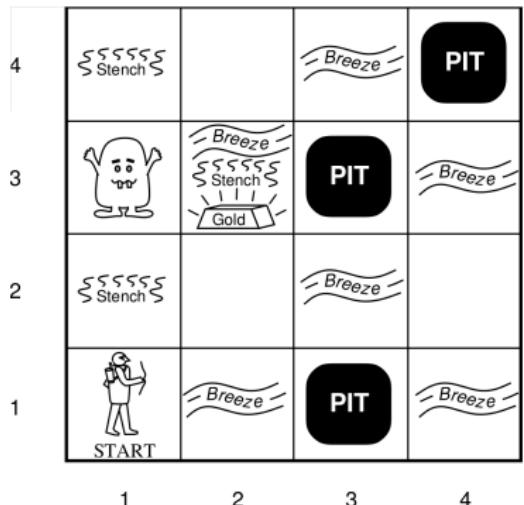
Applying Propositional Inference: Where is the Wumpus?

► Example 6.1 (Finding the Wumpus). The situation

4	Stench		Breeze	PIT
3	Wumpus	Breeze Stench Gold	PIT	Breeze
2	Stench		Breeze	
1	START	Breeze	PIT	Breeze
	1	2	3	4

Applying Propositional Inference: Where is the Wumpus?

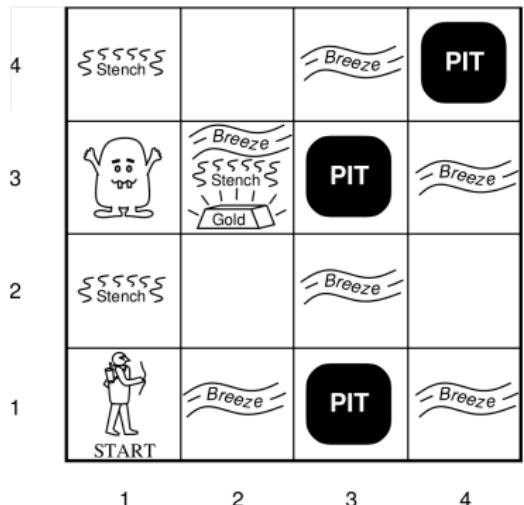
- **Example 6.1 (Finding the Wumpus).** The situation and what the agent knows



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

Applying Propositional Inference: Where is the Wumpus?

- **Example 6.1 (Finding the Wumpus).** The situation and what the agent knows

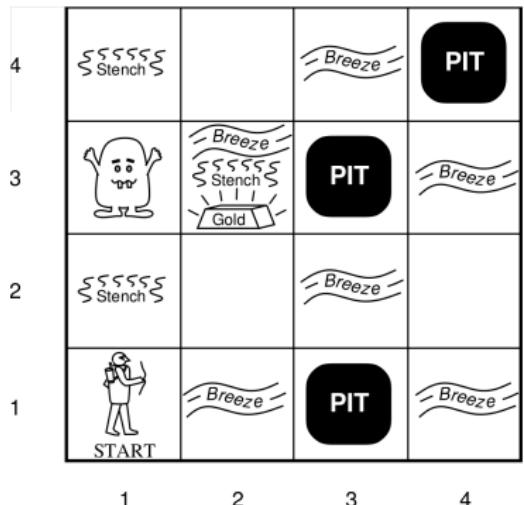


1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

- What should the agent do next and why?

Applying Propositional Inference: Where is the Wumpus?

- **Example 6.1 (Finding the Wumpus).** The situation and what the agent knows

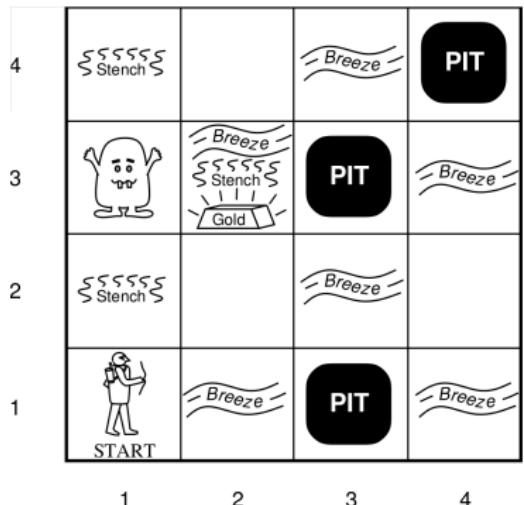


1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

- What should the agent do next and why?
► One possibility: Convince yourself that the Wumpus is in [1, 3] and shoot it.

Applying Propositional Inference: Where is the Wumpus?

- **Example 6.1 (Finding the Wumpus).** The situation and what the agent knows



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

- What should the agent do next and why?
- One possibility: Convince yourself that the Wumpus is in [1, 3] and shoot it.
- What is the general mechanism here? (for the agent function)

Where is the Wumpus? Our Knowledge

- **Idea:** We formalize the knowledge about the Wumpus world in PL^0 and use a test calculus to check for entailment.
- **Simplification:** We worry only about the Wumpus and stench:
 $S_{i,j} \hat{=} \text{stench in } [i,j]$, $W_{i,j} \hat{=} \text{Wumpus in } [i,j]$.
- 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

$$R_1 := \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$$

$$R_2 := \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}$$

$$R_3 := \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}$$

$$R_4 := S_{1,2} \Rightarrow W_{1,3} \vee W_{2,2} \vee W_{1,1}$$

:

- To show: $R_1, R_2, R_3, R_4 \models W_{1,3}$ (we will use resolution)

And Now Using Resolution Conventions

- ▶ We obtain the clause set Δ composed of the following clauses:
 - ▶ Propositions whose value we know: $S_{1,1}^F, W_{1,1}^F, S_{2,1}^F, W_{2,1}^F, S_{1,2}^T, W_{1,2}^F$
 - ▶ Knowledge about the Wumpus and smell:

from clauses

R_1	$S_{1,1}^T \vee W_{1,1}^F, S_{1,1}^T \vee W_{1,2}^F, S_{1,1}^T \vee W_{2,1}^F$
R_2	$S_{2,1}^T \vee W_{1,1}^F, S_{2,1}^T \vee W_{2,1}^F, S_{2,1}^T \vee W_{2,2}^F, S_{2,1}^T \vee W_{3,1}^F$
R_3	$S_{1,2}^T \vee W_{1,1}^F, S_{1,2}^T \vee W_{1,2}^F, S_{1,2}^T \vee W_{2,2}^F, S_{1,2}^T \vee W_{1,3}^F$
R_4	$S_{1,2}^F \vee W_{1,3}^T \vee W_{2,2}^T \vee W_{1,1}^T$

- ▶ Negated goal formula: $W_{1,3}^F$

Resolution Proof Killing the Wumpus!

- ▶ **Example 6.2 (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 neighbor cell of (1, 2).
 - ▶ Parents: $W_{1,3}^F$ and $S_{1,2}^F \vee W_{1,3}^T \vee W_{2,2}^T \vee W_{1,1}^T$.
 - ▶ Resolvent: $S_{1,2}^F \vee W_{2,2}^T \vee W_{1,1}^T$.
 - ▶ There's a stench in (1, 2), so it must be another neighbor.
 - ▶ Parents: $S_{1,2}^T$ and $S_{1,2}^F \vee W_{2,2}^T \vee W_{1,1}^T$.
 - ▶ Resolvent: $W_{2,2}^T \vee W_{1,1}^T$.
 - ▶ We've been to (1, 1), and there's no Wumpus there, so it can't be (1, 1).
 - ▶ Parents: $W_{1,1}^F$ and $W_{2,2}^T \vee W_{1,1}^T$.
 - ▶ Resolvent: $W_{2,2}^T$.
 - ▶ There is no stench in (2, 1) so it can't be (2, 2) either, in contradiction.
 - ▶ Parents: $S_{2,1}^F$ and $S_{2,1}^T \vee W_{2,2}^F$.
 - ▶ Resolvent: $W_{2,2}^F$.
 - ▶ Parents: $W_{2,2}^F$ and $W_{2,2}^T$.
 - ▶ Resolvent: \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 $W_{1,3}^F$ come from?

- ▶ Question: Where did the $W_{1,3}^F$ come from?
- ▶ **Observation 6.3.** We need a general mechanism for making conjectures.
- ▶ Idea: Interpret the Wumpus world as a search problem $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ where
 - ▶ the states \mathcal{S} are given by the cells (and agent orientation) and
 - ▶ the actions \mathcal{A} by the possible actions of the agent.
- Use tree search as the main agent function and a test calculus for testing all dangers (pits), opportunities (gold) and the Wumpus.
- ▶ **Example 6.4 (Back to the Wumpus).** In 6.1, 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.

7 Conclusion

- ▶ Sometimes, it pays off to think before acting.
- ▶ In AI, “thinking” is implemented in terms of **reasoning** in order to **deduce** new knowledge from a **knowledge base** represented in a suitable **logic**.
- ▶ Logic prescribes a **syntax** for formulas, as well as a **semantics** prescribing which **interpretations** satisfy them. A **entails** B if all interpretations that satisfy A also satisfy B. **Deduction** is the process of deriving new entailed formulas.
- ▶ **Propositional logic** formulas are built from **atomic propositions**, with the connectives **and**, **or**, **not**.
- ▶ 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 $\text{KB} \models A$ by showing that $\text{KB} \cup \{\neg A\}$ is unsatisfiable.

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 ...

- ▶ 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 objects. (cf. and)
- ▶ ...and a whole zoo of logics much more powerful still.
- ▶ Note: In applications, propositional CNF encodings are generated by computer programs. This mitigates (but does not remove!) the inconveniences of propositional modeling.

Chapter 12 Propositional Reasoning: SAT Solvers

1 Introduction

- ▶ : Basic definitions and concepts; machine-oriented calculi
- ▶ Sets up the framework. Resolution is the quintessential reasoning procedure underlying most successful **SAT solvers**.
- ▶ Here: The **Davis Putnam procedure** and **clause learning**.
- ▶ State-of-the-art algorithms for reasoning about propositional logic, and an important observation about how they behave.

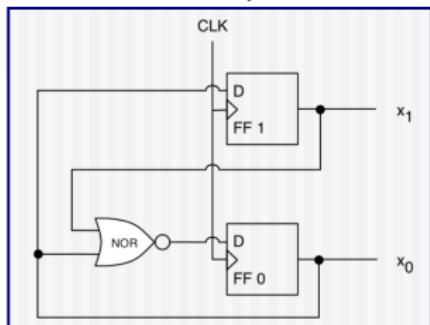
SAT: The Propositional Satisfiability Problem

- ▶ **Definition 1.1.** The **SAT problem (SAT)**: Given a propositional formula A, decide whether or not A is satisfiable. We denote the class of all SAT problems with **SAT**
- ▶ The **SAT problem** was the first problem proved to be **NP-complete!**
- ▶ A is commonly assumed to be in CNF. This is without loss of generality, because any A can be transformed into a satisfiability-equivalent CNF formula (cf.) in polynomial time.
- ▶ Active research area, annual **SAT** conference, lots of tools etc. available:
<http://www.satlive.org/>
- ▶ **Definition 1.2.** Tools addressing **SAT** are commonly referred to as **SAT solvers**.
- ▶ **Recall:** To decide whether $\text{KB} \models A$, decide satisfiability of $\theta := \text{KB} \cup \{\neg A\}$: θ is unsatisfiable iff $\text{KB} \models A$.
- ▶ **Consequence:** Deduction can be performed using **SAT solvers**.

- ▶ **Recall:** Constraint network $\gamma = \langle V, D, C \rangle$ has variables $v \in V$ with finite domains $D_v \in D$, and binary constraints $C_{uv} \in C$ which are relations over u, v specifying the permissible combined assignments to u and v . One extension is to allow constraints of higher arity.
- ▶ **Observation 1.3 (SAT: A kind of CSP).** SAT can be viewed as a CSP problem in which all variable domains are Boolean, and the constraints have unbounded arity.
- ▶ **Encoding CSP as SAT:** Given any constraint network γ , we can in low-order polynomial time construct a CNF formula $A(\gamma)$ that is satisfiable iff γ is solvable.
- ▶ **Problem** How to construct $A(\gamma)$? See the exercises.
- ▶ **Upshot:** Anything we can do with CSP, we can (in principle) do with SAT.

Example Application: Hardware Verification

► Example 1.4 (Hardware Verification).



- Counter, repeatedly from $c = 0$ to $c = 2$.
- 2 bits x_1 and x_0 ; $c = 2 * x_1 + x_0$.
- ("FF" Flip-Flop, "D" Data IN, "CLK" 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 set Δ of clauses.

- Clauses: $y_1^F \vee x_0^T$, $y_1^T \vee x_0^F$, $y_0^T \vee x_1^T \vee x_0^T$, $y_0^F \vee x_1^F$, $y_0^F \vee x_0^F$, $x_1^F \vee x_0^F$, y_1^T , y_0^T .

► Step 3: Call a SAT solver (up next).

Our Agenda for This Chapter

- ▶ The Davis-Putnam (Logemann-Loveland) Procedure: How to systematically test satisfiability?
 - ▶ The quintessential SAT solving procedure, DPLL.
- ▶ DPLL is (A Restricted Form of) Resolution: How does this relate to what we did in the last chapter?
 - ▶ Mathematical understanding of DPLL.
- ▶ Why Did Unit Propagation Yield a Conflict?: How can we analyze which mistakes were made in “dead” search branches?
 - ▶ Knowledge is power, see next.
- ▶ Clause Learning: How can we learn from our mistakes?
 - ▶ One of the key concepts, perhaps *the* key concept, underlying the success of SAT.
- ▶ Phase Transitions – Where the Really Hard Problems Are: Are *all* formulas “hard” to solve?
 - ▶ The answer is “no”. And in some cases we can figure out exactly when they are/aren’t hard to solve.

2 The Davis-Putnam (Logemann-Loveland) Procedure

The DPLL Procedure

- **Definition 2.1.** The **Davis Putnam procedure (DPLL)** is a SAT solver called on a clause set Δ and the empty variable assignment φ . It interleaves **unit propagation (UP)** and **splitting**:

```
function DPLL( $\Delta, I$ ) returns a partial interpretation  $I$ , or “unsatisfiable”
  /* Unit Propagation (UP) Rule: */
   $\Delta' :=$  a copy of  $\Delta$ ;  $I' := I$ 
  while  $\Delta'$  contains a unit clause  $C = I$  do
    extend  $I'$  with the respective truth value for the proposition underlying  $I$ 
    simplify  $\Delta'$  /* remove false literals and true clauses */
  /* Termination Test: */
  if  $\square \in \Delta'$  then return “unsatisfiable”
  if  $\Delta' = \{\}$  then return  $I'$ 
  /* Splitting Rule: */
  select some proposition  $P$  for which  $I'$  is not defined
   $I'' := I'$  extended with one truth value for  $P$ ;  $\Delta'' :=$  a copy of  $\Delta'$ ; simplify  $\Delta''$ 
  if  $I''' := \text{DPLL}(\Delta'', I'') \neq \text{“unsatisfiable”}$  then return  $I'''$ 
   $I'' := I'$  extended with the other truth value for  $P$ ;  $\Delta'' := \Delta'$ ; simplify  $\Delta''$ 
  return DPLL( $\Delta'', I''$ )
```

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

DPLL: Example (Vanilla1)

► **Example 2.2 (UP and Splitting).** Let $\Delta := P^T \vee Q^T \vee R^F; P^F \vee Q^F; R^T; P^T \vee Q^F$

1. UP Rule: $R \mapsto T$

$$P^T \vee Q^T; P^F \vee Q^F; P^T \vee Q^F$$

2. Splitting Rule:

2a. $P \mapsto F$
 $Q^T; Q^F$

2b. $P \mapsto T$
 Q^F

3a. UP Rule: $Q \mapsto T$
□
returning “unsatisfiable”

3b. UP Rule: $Q \mapsto F$
clause set empty
returning “ $R \mapsto T, P \mapsto T, Q \mapsto F$ ”

DPLL: Example (Vanilla2)

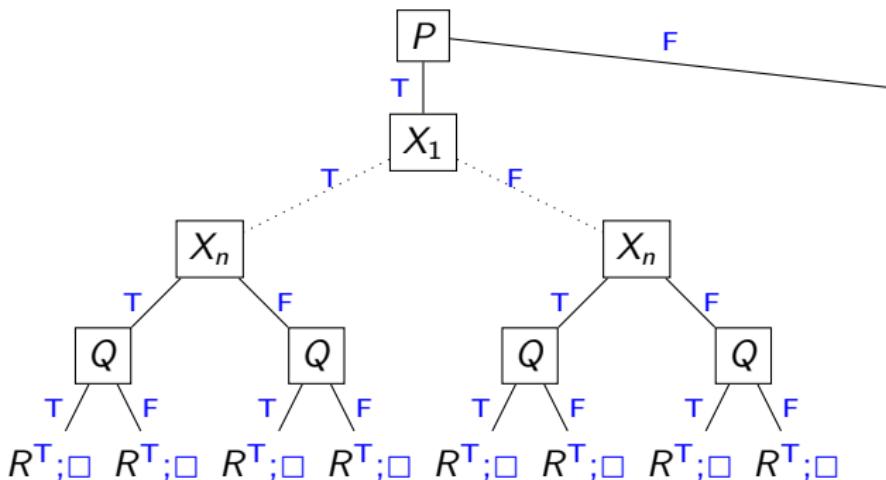
- ▶ **Observation:** Sometimes UP is all we need.
- ▶ **Example 2.3.** Let $\Delta := Q^F \vee P^F; P^T \vee Q^F \vee R^F \vee S^F; Q^T \vee S^F; R^T \vee S^F; S^T$
 1. UP Rule: $S \mapsto T$
 $Q^F \vee P^F; P^T \vee Q^F \vee R^F; Q^T; R^T$
 2. UP Rule: $Q \mapsto T$
 $P^F; P^T \vee R^F; R^T$
 3. UP Rule: $R \mapsto T$
 $P^F; P^T$
 4. UP Rule: $P \mapsto T$
□

DPLL: Example (Redundance1)

- **Example 2.4.** We introduce some nasty redundancy 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 $\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$



- ▶ **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.

- ▶ **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.)

- ▶ **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?

- ▶ **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 $\hat{=}$ backtracking with inference, where inference $\hat{=}$ unit propagation.
 - ▶ Unit propagation is sound: It does not reduce the set of solutions.
 - ▶ Runtime is exponential in worst-case, good variable/value selection strategies required.

3 DPLL $\hat{=}$ (A Restricted Form of) Resolution

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

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

- ▶ **Definition 3.1 (Unit Resolution).** Unit resolution (UR) is the calculus consisting of the following inference rule:

$$\frac{C \vee P^\alpha \quad P^\beta \quad \alpha \neq \beta}{C} \text{UR}$$

- ▶ Unit propagation $\hat{=}$ resolution restricted to cases where one parent is unit clause.
- ▶ **Observation 3.2 (Soundness).** UR is sound. (since resolution is)
- ▶ **Observation 3.3 (Completeness).** UR is not refutation complete.
- ▶ **Example 3.4.** $P^T \vee Q^T; P^T \vee Q^F; P^F \vee Q^T; P^F \vee Q^F$ 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.

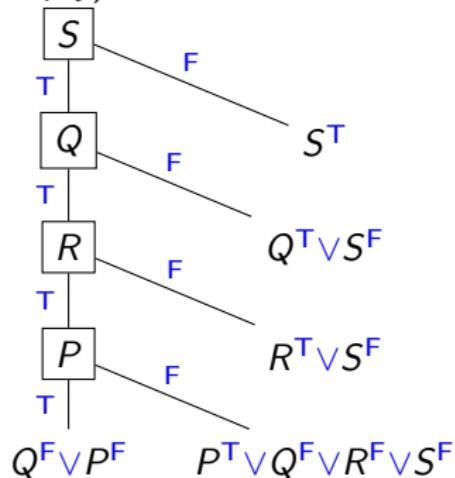
- ▶ **Definition 3.5.** We define the **number of decisions** of a DPLL run as the total number of times a truth value was set by either **unit propagation** or **splitting**.
- ▶ **Theorem 3.6.** If DPLL returns “unsatisfiable” on Δ , then $\Delta \vdash_{\mathcal{R}} \square$ with a **resolution proof** 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 $X = F$ the respective clause C must contain X^T ; and for $X = T$ 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, \dots, 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.



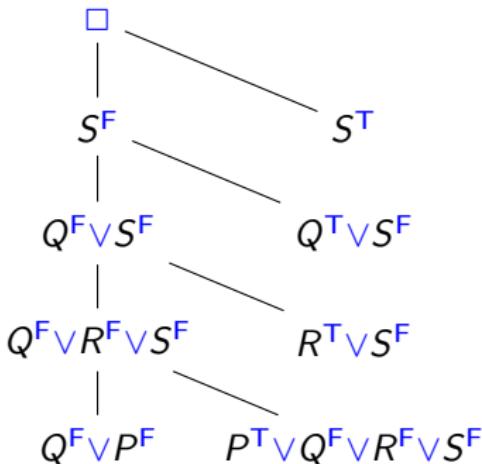
DPLL vs. Resolution: Example (Vanilla2)

- **Example 3.7.** $\Delta := Q^F \vee P^F; P^T \vee Q^F \vee R^F \vee S^F; Q^T \vee S^F; R^T \vee S^F; S^T$

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.

DPLL vs. Resolution: Discussion

- ▶ So What?: The theorem we just proved helps to understand DPLL: DPLL is an effective practical method for conducting resolution proofs.
- ▶ In fact: DPLL \cong tree resolution.
- ▶ **Definition 3.8.** In a **tree resolution**, each derived clause C is used only once (at its parent).
- ▶ Problem: The same C must be derived anew every time it is used!
- ▶ This is a fundamental weakness: There are inputs Δ whose shortest **tree resolution** proof is exponentially longer than their shortest (general) resolution proof.
- ▶ Intuitively: DPLL makes the same mistakes over and over again.
- ▶ Idea: DPLL should learn from its mistakes on one search branch, and apply the learned knowledge to other branches.
- ▶ To the rescue: clause learning (up next)

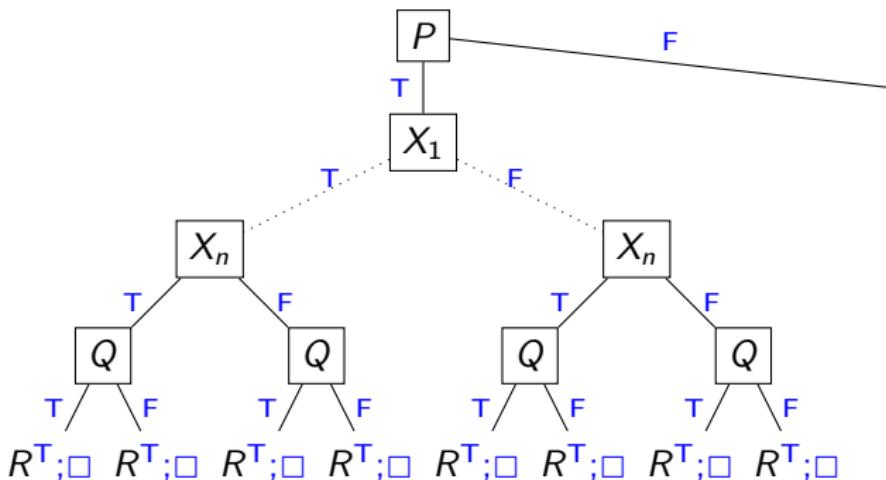
4 Why Did Unit Propagation Yield a Conflict?

DPLL: Example (Redundance1)

- **Example 4.1.** We introduce some nasty redundancy 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 $\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$



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 4.2.** 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 4.3 (Implication Graph).** Let Δ be a set of clauses, 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** \square_C for every clause C that became empty on β . Wherever a clause $(l_1 \vee \dots \vee l_k \vee l') \in \Delta$ became unit with **implied literal** l' , G_β^{impl} includes the edges (\bar{l}_i, l') .
Where $C = l_1 \vee \dots \vee l_k \in \Delta$ became empty, G_β^{impl} includes the edges (\bar{l}_i, \square_C) .
- ▶ **Question:** How do we know that \bar{l}_i are vertices in G_β^{impl} ?
- ▶ **Answer:** Because $l_1 \vee \dots \vee l_k \vee l'$ became unit/empty.
- ▶ **Observation 4.4.** 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 4.5.** Let $\Delta := P^T \vee Q^T \vee R^F; P^F \vee Q^F; R^T; P^T \vee Q^F$ We look at the left branch of the derivation from 2.2

Implication graph:

1. UP Rule: $R \mapsto T$

Implied literal R^T .

$P^T \vee Q^T; P^F \vee Q^F; P^T \vee Q^F$

2. Splitting Rule:

- 2a. $P \mapsto F$

Choice literal P^F .

$Q^T; Q^F$

- 3a. UP Rule: $Q \mapsto T$

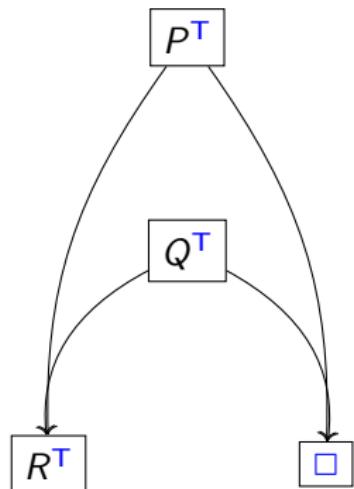
Implied literal Q^T

edges (R^T, Q^T) and (P^F, Q^T) .

□

Conflict vertex $\square_{(P^T \vee Q^F)}$

edges $(P^F, \square_{(P^T \vee Q^F)})$ and $(Q^T, \square_{(P^T \vee Q^F)})$.



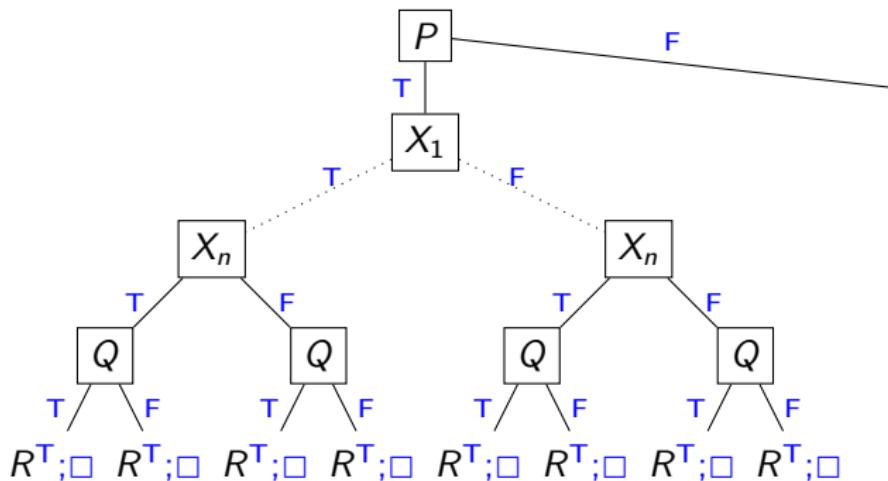
Implication Graphs: Example (Redundance1)

► **Example 4.6.** Continuing from 4.5:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$

$$\text{DPLL on } \Delta; \Theta \text{ with } \Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$$

Choice literals: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied literal: R^T .



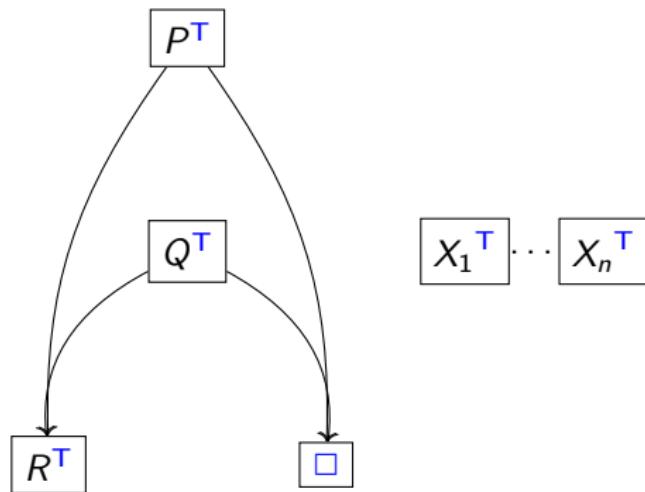
Implication Graphs: Example (Redundance1)

► **Example 4.6.** Continuing from 4.5:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$

$$\text{DPLL on } \Delta; \Theta \text{ with } \Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$$

Choice literals: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied literal: R^T .



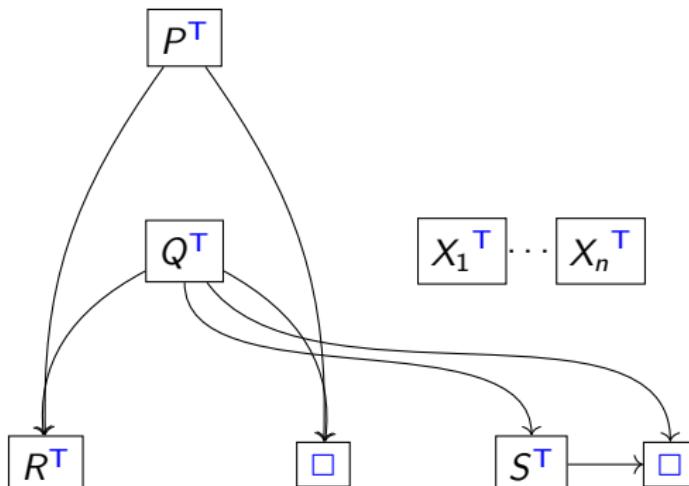
Implication Graphs: Example (Redundance2)

► **Example 4.7.** Continuing from 4.6:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$
$$\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$$

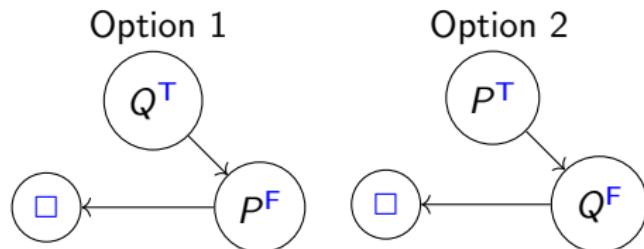
DPLL on $\Delta; \Theta; \Phi$ with $\Phi := Q^F \vee S^T; Q^F \vee S^F$

choice literals: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. 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 4.8.** $\Delta = P^F \vee Q^F; Q^T; P^T$



- ▶ A conflict graph captures “what went wrong” in a failed node.
- ▶ **Definition 4.9 (Conflict Graph).** Let Δ be a set of clauses, and let G_β^{impl} be the implication graph for some search branch β of DPLL on Δ . A sub-graph C of G_β^{impl} is a **conflict graph** if:
 - (i) C contains exactly one conflict vertex \square_C .
 - (ii) If I' is a vertex in C , then all parents of I' , i.e. vertices \bar{l}_i with a $/$ edge (\bar{l}_i, I') , are vertices in C as well.
 - (iii) All vertices in C have a path to \square_C .
- ▶ Conflict graph $\hat{=}$ Starting at a conflict vertex, backchain through the implication graph until reaching choice literals.

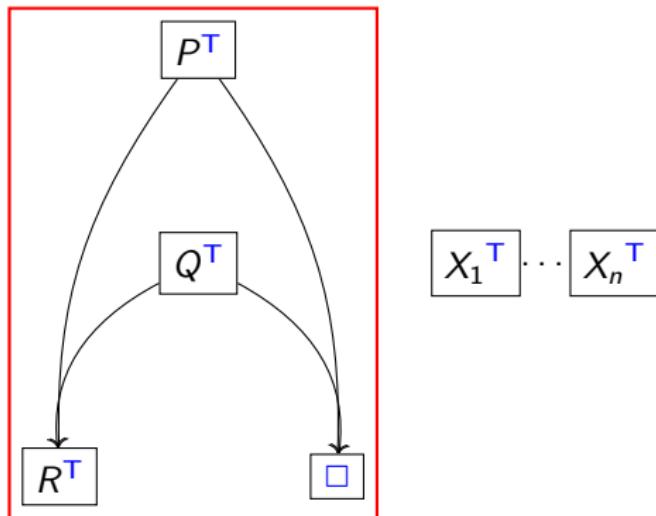
Conflict-Graphs: Example (Redundance1)

- **Example 4.10.** Continuing from 4.6:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$

DPLL on $\Delta; \Theta$ with $\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$

Choice literals: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied literals: R^T .



Conflict Graphs: Example (Redundance2)

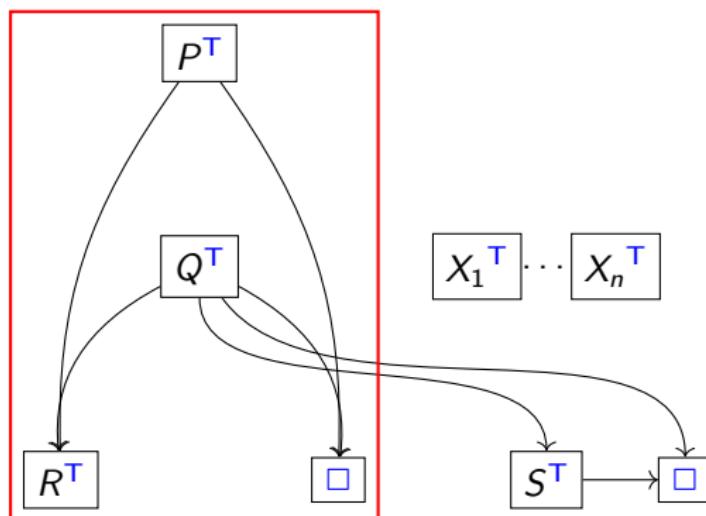
► **Example 4.11.** Continuing from 4.7 and 4.10:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$

$$\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$$

DPLL on $\Delta; \Theta; \Phi$ with $\Phi := Q^F \vee S^T; Q^F \vee S^F$

Choice literals: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied literals: R^T .



Conflict Graphs: Example (Redundance2)

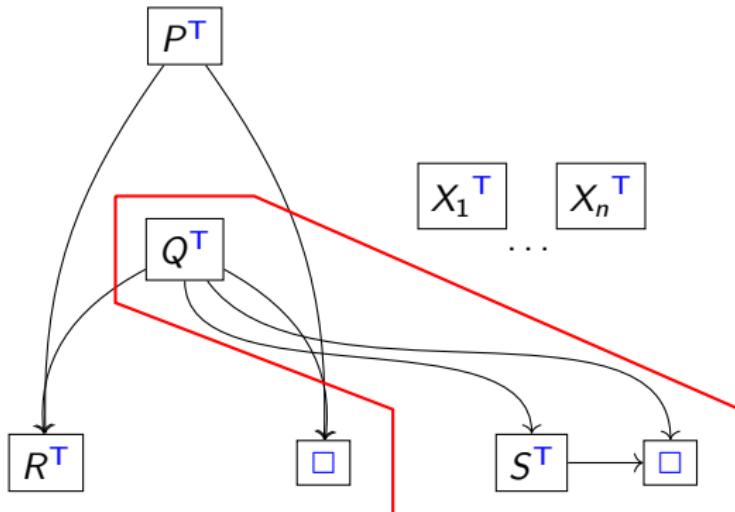
► **Example 4.11.** Continuing from 4.7 and 4.10:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$

$$\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$$

DPLL on $\Delta; \Theta; \Phi$ with $\Phi := Q^F \vee S^T; Q^F \vee S^F$

Choice literals: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied literals: R^T .



5 Clause Learning

- ▶ Observation: Conflict graphs encode entailment relation.
- ▶ **Definition 5.1.** Let Δ be a set of clauses, 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 $c := \bigvee_{l \in L} l$ the learned clause for C .
- ▶ Assertion Let Δ , C , and c as in 5.1, then $\Delta \models c$.
- ▶ 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 5.2.** Clause learning is the process of adding learned clauses to DPLL clause sets at specific points. (details coming up)

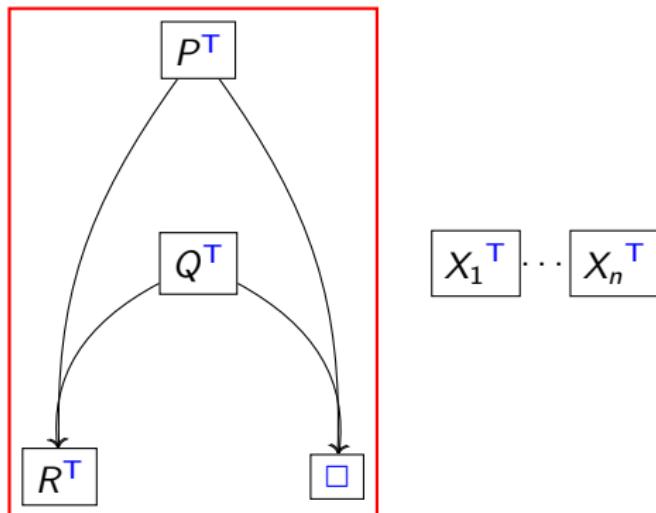
Clause Learning: Example (Redundance1)

► **Example 5.3.** Continuing from 4.10:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$

$$\text{DPLL on } \Delta; \Theta \text{ with } \Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$$

Choice literals: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied literals: R^T .



Learned clause: $P^F \vee Q^F$

- ▶ 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 I' . e.g. the choice $I' = Q$.
- ▶ 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 I' : otherwise, the conflict would have occurred earlier on.
So $C = \bar{l}_1 \vee \dots \vee \bar{l}_k \vee \bar{l}'$ where l_1, \dots, l_k are earlier choices.
- ▶ Example 5.4. $l_1 = P$, $C = P^F \vee Q^F$, $I' = Q$.
- ▶ Observation: Given the earlier choices l_1, \dots, l_k , after we learned the new clause $C = \bar{l}_1 \vee \dots \vee \bar{l}_k \vee \bar{l}'$, the value of \bar{l}' is now set by UP!
- ▶ So we can continue:
 3. We set the opposite choice \bar{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)

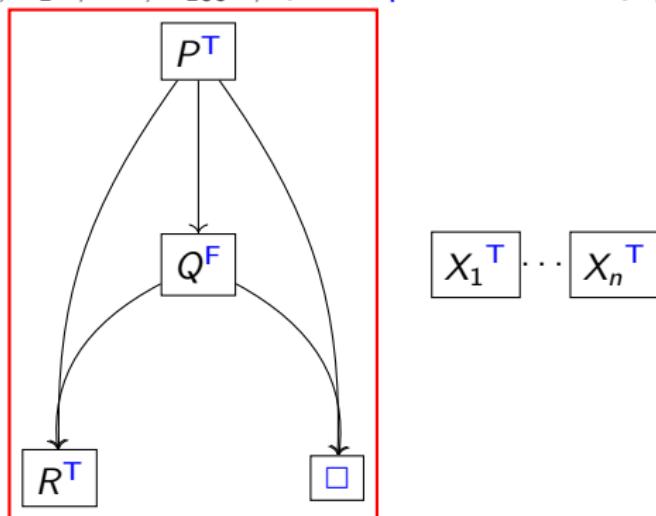
► **Example 5.5.** Continuing from 5.3:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$

$$\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$$

DPLL on $\Delta; \Theta; \Phi$ with $\Phi := P^F \vee Q^F$

Choice literals: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied literals: Q^F, R^T .



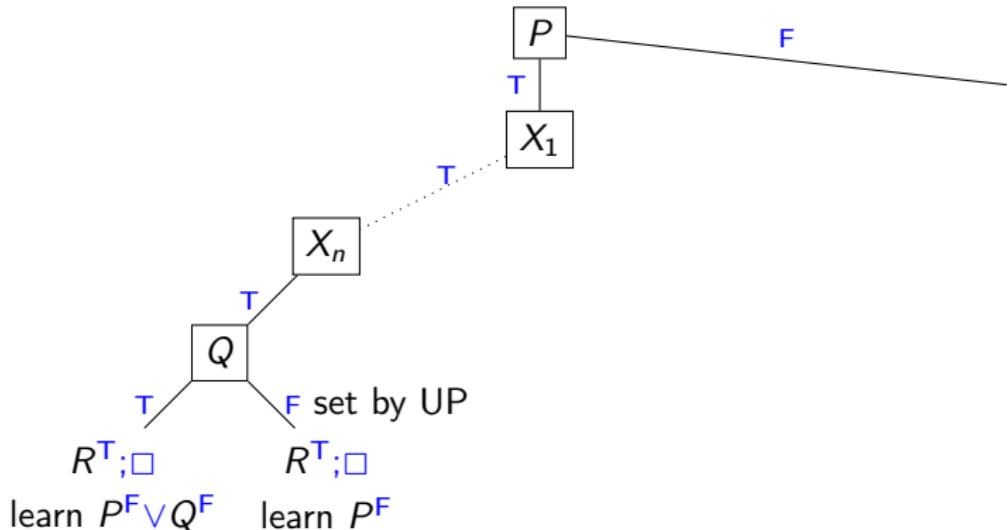
Learned clause: P^F

NOT the same Mistakes over Again: (Redundance1)

- **Example 5.6.** Continuing from 4.10:

$$\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$$

$$\text{DPLL on } \Delta; \Theta \text{ with } \Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$$



- **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 = tree resolution (from slide 356)
 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].
(In how far 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 371, 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 369 – 371 are merely meant to give a *rough intuition* on “backjumping”.
- **Definition 5.7 (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  $C = \overline{l_1} \vee \dots \vee \overline{l_k} \vee l'$ 
        if  $L = 0$  then return UNSAT
         $L := \max_{i=1}^k \text{level}(l_i)$ ; erase  $I$  below  $L$ 
        add  $C$  into  $\Delta$ ; 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  $L$ 
         $L := L + 1$ 
```

- ▶ 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 .

6 Phase Transitions: Where the *Really* Hard Problems Are

Where Are the Hard Problems?

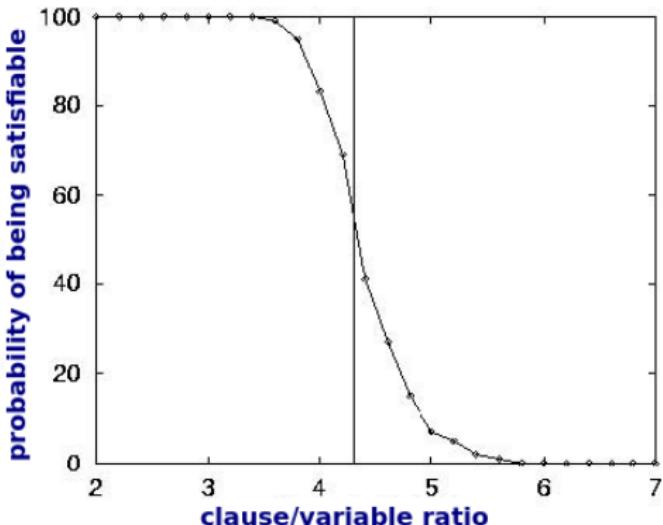
- ▶ SAT is NP-hard. Worst case for DPLL is 2^n , with n propositions.
- ▶ Imagine I gave you as homework to make a formula family $\{\varphi\}$ where DPLL runtime necessarily is in the order of 2^n .
 - ▶ I promise you're not gonna find this easy . . . (although it is of course possible: e.g., the "Pigeon 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.)

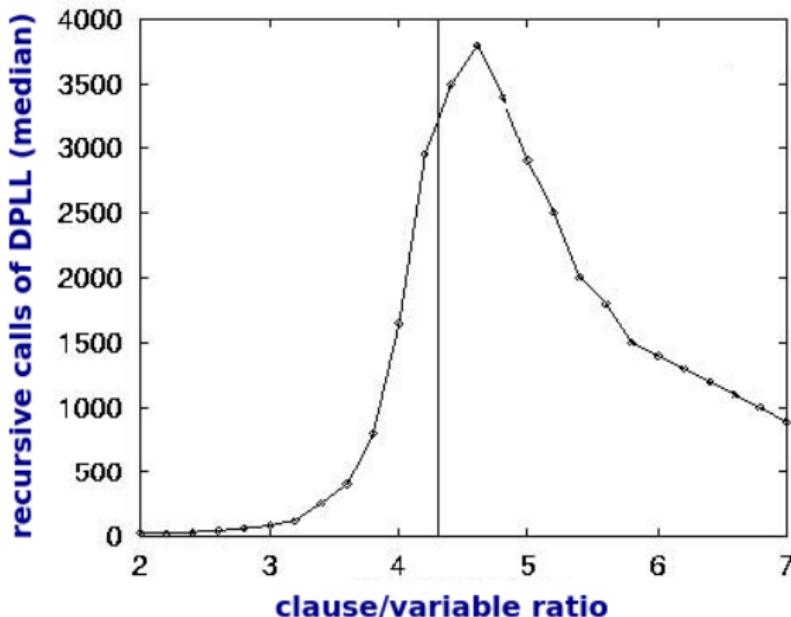
Phase Transitions in SAT [MSL92]

- 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 runtime 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

- ▶ Assertion (Phase Transition Conjecture) All **NP**-complete problems have at least one **order parameter**, and the hard to solve problems are around a critical value of this order parameter. This critical value (a **phase transition**) separates one region from another, such as over-constrained and under-constrained regions of the problem space.
- ▶ [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.

- ▶ **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.

7 Conclusion

- ▶ 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 $\hat{=}$ 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.

- ▶ 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 7.1. The **GSAT algorithm** OUTPUT: a satisfying truth assignment of Δ , if found

```
function GSAT ( $\Delta$ , MaxFlips MaxTries
    for  $i := 1$  to MaxTries
         $I :=$  a randomly-generated truth assignment
        for  $j := 1$  to MaxFlips
            if  $I$  satisfies  $\Delta$  then return  $I$ 
                 $X :=$  a proposition reversing whose truth assignment gives
                    the largest increase in the number of satisfied clauses
                 $I := I$  with the truth assignment of  $X$  reversed
        end for
    end for
    return "no satisfying assignment found"
```

- ▶ Local search is not as successful in SAT applications, and the underlying ideas are very similar to those presented in (Not covered here)

Topics We Didn't Cover Here

- ▶ **Variable/value selection heuristics:** A whole zoo is out there.
- ▶ **Implementation techniques:** One of the most intensely researched subjects. Famous “watched literals” technique for **UP** had huge practical impact.
- ▶ **Local search:** In space of all truth value assignments. GSAT (slide 384) 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** effectively?
- ▶ **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 effective than Y)?

Chapter 13 First Order Predicate Logic

1 Motivation: A more Expressive Language

Let's Talk About Blocks, Baby . . .

► Question: What do you see here?



Let's Talk About Blocks, Baby . . .

- ▶ Question: What do you see here?



- ▶ You say: "All blocks are red"; "All blocks are on the table"; "A is a block".
- ▶ And now: Say it in propositional logic!

Let's Talk About Blocks, Baby ...

- ▶ **Question:** What do you see here?



- ▶ **You say:** "All blocks are red"; "All blocks are on the table"; "A is a block".
- ▶ **And now:** Say it in propositional logic!
- ▶ **Answer:** "isRedA", "isRedB", ..., "onTableA", "onTableB", ..., "isBlockA", ...
- ▶ **Wait a sec!:** Why don't we just say, e.g., "AllBlocksAreRed" and "isBlockA"?
- ▶ **Problem:** Could we conclude that A is red? (No)
These statements are atomic (just strings); their inner structure ("all blocks", "is a block") is not captured.

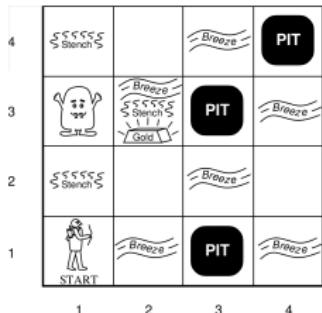
Let's Talk About Blocks, Baby . . .

- ▶ **Question:** What do you see here?



- ▶ **You say:** "All blocks are red"; "All blocks are on the table"; "A is a block".
- ▶ **And now:** Say it in propositional logic!
- ▶ **Answer:** "isRedA", "isRedB", . . . , "onTableA", "onTableB", . . . , "isBlockA", . . .
- ▶ **Wait a sec!:** Why don't we just say, e.g., "AllBlocksAreRed" and "isBlockA"?
- ▶ **Problem:** Could we conclude that A is red? (No)
These statements are atomic (just strings); their inner structure ("all blocks", "is a block") is not captured.
- ▶ **Idea:** Predicate Logic (PL1) extends propositional logic with the ability to explicitly speak about objects and their properties.
- ▶ **How?:** Variables ranging over objects, predicates describing object properties, . . .
- ▶ **Example 1.1.** " $\forall x.\text{block}(x) \Rightarrow \text{red}(x)$ "; " $\text{block}(A)$ "

Let's Talk About the Wumpus Instead?



Percepts: [Stench, Breeze, Glitter, Bump, Scream]

- ▶ ▶ Cell adjacent to Wumpus: Stench (else: None).
- ▶ Cell adjacent to Pit: Breeze (else: None).
- ▶ Cell that contains gold: Glitter (else: None).
- ▶ You walk into a wall: Bump (else: None).
- ▶ Wumpus shot by arrow: Scream (else: None).

▶ Say, in propositional logic: "Cell adjacent to Wumpus: Stench."

$$\begin{aligned} \blacktriangleright & W_{1,1} \Rightarrow S_{1,2} \wedge S_{2,1} \\ \blacktriangleright & W_{1,2} \Rightarrow S_{2,2} \wedge S_{1,1} \wedge S_{1,3} \\ \blacktriangleright & W_{1,3} \Rightarrow S_{2,3} \wedge S_{1,2} \wedge S_{1,4} \\ \blacktriangleright & \dots \end{aligned}$$

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

- ▶ Even worse!
- ▶ **Example 1.2 (Integers).** A limited vocabulary to talk about these
 - ▶ The objects: $\{1, 2, 3, \dots\}$.
 - ▶ Predicate 1: “`even(x)`” should be true iff x is even.
 - ▶ Predicate 2: “`eq(x, y)`” should be true iff $x = y$.
 - ▶ Function: `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: “`eq(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. even(x) \Rightarrow even(succ(succ(x)))`”.

- ▶ **Example 1.3.**

$$\forall n.\text{gt}(n, 2) \Rightarrow \neg(\exists a,b,c.\text{eq}(\text{plus}(\text{pow}(a, n), \text{pow}(b, n)), \text{pow}(c, n)))$$

- ▶ **Theorem proving in PL1!** Arbitrary theorems, in principle.
- ▶ Fermat's last theorem is of course infeasible, but interesting theorems can and have been proved automatically.
- ▶ See http://en.wikipedia.org/wiki/Automated_theorem_proving.
- ▶ **Note:** Need to **axiomatize** "Plus", "PowerOf", "Equals". See http://en.wikipedia.org/wiki/Peano_axioms

What Are the Practical Relevance/Applications?

- ▶ ... even asking this question is a sacrilege:

What Are the Practical Relevance/Applications?

- ▶ ... even asking this question is a sacrilege:
- ▶ (Quotes from Wikipedia)
 - ▶ *"In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics."*

What Are the Practical Relevance/Applications?

- ▶ **... even asking this question is a sacrilege:**
- ▶ (Quotes from Wikipedia)
 - ▶ *"In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics."*
 - ▶ *"The development of logic since Frege, Russell, and Wittgenstein had a profound influence on the practice of philosophy and the perceived nature of philosophical problems, and Philosophy of mathematics."*

What Are the Practical Relevance/Applications?

- ▶ **... even asking this question is a sacrilege:**
- ▶ (Quotes from Wikipedia)
 - ▶ *"In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics."*
 - ▶ *"The development of logic since Frege, Russell, and Wittgenstein had a profound influence on the practice of philosophy and the perceived nature of philosophical problems, and Philosophy of mathematics."*
 - ▶ *"During the later medieval period, major efforts were made to show that Aristotle's ideas were compatible with Christian faith."*
 - ▶ (In other words: the church issued for a long time that Aristotle's ideas were incompatible with Christian faith.)

What Are the Practical Relevance/Applications?

► You're asking it anyhow?

- ▶ Logic programming. Prolog et al.
- ▶ Databases. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
- ▶ Semantic technology. Mega-trend since > a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.

What Are the Practical Relevance/Applications?

- ▶ You're asking it anyhow?
 - ▶ Logic programming. Prolog et al.
 - ▶ Databases. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
 - ▶ Semantic technology. Mega-trend since > a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.
- ▶ Prominent PL1 fragment: Web Ontology Language OWL.

What Are the Practical Relevance/Applications?

- ▶ You're asking it anyhow?
 - ▶ Logic programming. Prolog et al.
 - ▶ Databases. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
 - ▶ Semantic technology. Mega-trend since > a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.
- ▶ Prominent PL1 fragment: Web Ontology Language **OWL**.
- ▶ Prominent data set: The WWW. (Semantic Web)

What Are the Practical Relevance/Applications?

- ▶ You're asking it anyhow?
 - ▶ Logic programming. Prolog et al.
 - ▶ Databases. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
 - ▶ Semantic technology. Mega-trend since > a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.
- ▶ Prominent PL1 fragment: Web Ontology Language **OWL**.
- ▶ Prominent data set: The WWW. (Semantic Web)
- ▶ Assorted quotes on Semantic Web and OWL:
 - ▶ "*The brain of humanity.*"
 - ▶ "*The Semantic Web will never work.*"
 - ▶ "*A TRULY meaningful way of interacting with the Web may finally be here: the Semantic Web. The idea was proposed 10 years ago. A triumvirate of internet heavyweights – Google, Twitter, and Facebook – are making it real.*"

(A Few) Semantic Technology Applications

Web Queries

The screenshot shows the BBC Sport homepage for the 2010 World Cup. The main headline is "WORLD CUP 2010". Below it, there's a summary for the match between England and the United States: "England 2-1 United States" with a timestamp of "6 June". To the right, there's a "Match report" section with a table showing the scoreline: England 1, United States 1; and a "Features" section with links like "German lessons" and "How to pronounce team names". At the bottom, there's a "Around the web" section with links to "BBC Sport's monthly page" and "England 2010 media".

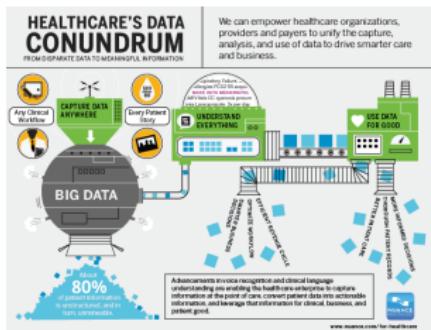
Jeopardy (IBM Watson)



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.
 - ▶ Algorithmic principles for reasoning about predicate logic.

2 First-Order Logic

First-Order Predicate Logic (PL¹)

- ▶ 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*, ...)

2.1 First-Order Logic: Syntax and Semantics

PL¹ Syntax (Signature and Variables)

- ▶ **Definition 2.1.** First order logic (PL¹), is a formal logical system extensively used in mathematics, philosophy, linguistics, and computer science. It combines propositional logic with the ability to quantify over individuals.
- ▶ PL¹ talks about two kinds of objects: (so we have two kinds of symbols)
 - ▶ truth values; sometimes annotated by type \circ (like in PL⁰)
 - ▶ individuals; sometimes annotated by type ι (numbers, foxes, Pokémons, ...)
- ▶ **Definition 2.2.** A first order signature consists of (all disjoint; $k \in \mathbb{N}$)
 - ▶ connectives: $\Sigma^\circ = \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \dots\}$ (functions on truth values)
 - ▶ function constants: $\Sigma^f_k = \{f, g, h, \dots\}$ (functions on individuals)
 - ▶ predicate constants: $\Sigma^p_k = \{p, q, r, \dots\}$ (relations among inds.)
 - ▶ (Skolem constants): $\Sigma^{sk}_k = \{f_k^1, f_k^2, \dots\}$ (witness constructors; countably ∞)
 - ▶ We take Σ_ι to be all of these together: $\Sigma_\iota := \Sigma^f \cup \Sigma^p \cup \Sigma^{sk}$, where $\Sigma^* := \bigcup_{k \in \mathbb{N}} \Sigma_k^*$ and define $\Sigma := \Sigma_\iota \cup \Sigma^\circ$.
- ▶ **Definition 2.3.** We assume a set of individual variables:
 $\mathcal{V}_\iota := \{X_\iota, Y_\iota, Z_\iota, X_\iota^1, X_\iota^2, \dots\}$. (countably ∞)

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

Alternative Notations for Quantifiers

Here	Elsewhere
$\forall x.A$	$\bigwedge x.A$ $(x)A$
$\exists x.A$	$\bigvee x.A$

- **Definition 2.9.** We call an occurrence of a variable X **bound** in a formula A , iff it occurs in a sub-formula $\forall X.B$ of A . We call a variable occurrence **free** otherwise.

For a formula A , we will use $BVar(A)$ (and $free(A)$) for the set of **bound (free)** variables of A , i.e. variables that have a free/bound occurrence in A .

- **Definition 2.10.** We define the set $free(A)$ of **frees** variable of a formula A :

$$free(X) := \{X\}$$

$$free(f(A_1, \dots, A_n)) := \bigcup_{1 \leq i \leq n} free(A_i)$$

$$free(p(A_1, \dots, A_n)) := \bigcup_{1 \leq i \leq n} free(A_i)$$

$$free(\neg A) := free(A)$$

$$free(A \wedge B) := free(A) \cup free(B)$$

$$free(\forall X.A) := free(A) \setminus \{X\}$$

- **Definition 2.11.** We call a formula A **closed** or **ground**, iff $free(A) = \emptyset$. We call a closed proposition a **sentence**, and denote the set of all ground terms with $cwff_\nu(\Sigma_\nu)$ and the set of sentences with $cwff_o(\Sigma_\nu)$.

- **Axiom 2.12.** **Bound** variables can be renamed, i.e. any subterm $\forall X.B$ of a formula A can be replaced by $A' := (\forall Y.B')$, where B' arises from B by replacing all $X \in free(B)$ with a new variable Y that does not occur in A . We call A an **alphabetical variant** of A .

- ▶ **Definition 2.13.** We inherit the universe $\mathcal{D}_o = \{\text{T}, \text{F}\}$ of truth values from PL⁰ and assume an arbitrary universe $\mathcal{D}_i \neq \emptyset$ of individuals (this choice is a parameter to the semantics)
- ▶ **Definition 2.14.** An interpretation \mathcal{I} assigns values to constants, e.g.
 - ▶ $\mathcal{I}(\neg): \mathcal{D}_o \rightarrow \mathcal{D}_o$ with $\text{T} \mapsto \text{F}$, $\text{F} \mapsto \text{T}$, and $\mathcal{I}(\wedge) = \dots$ (as in PL⁰)
 - ▶ $\mathcal{I}: \Sigma_k^f \rightarrow \mathcal{D}_i^k \rightarrow \mathcal{D}_i$ (interpret function symbols as arbitrary functions)
 - ▶ $\mathcal{I}: \Sigma_k^p \rightarrow \mathcal{P}(\mathcal{D}_i^k)$ (interpret predicates as arbitrary relations)
- ▶ **Definition 2.15.** A variable assignment $\varphi: \mathcal{V}_i \rightarrow \mathcal{D}_i$ maps variables into the universe.
- ▶ **Definition 2.16.** A model $\mathcal{M} = \langle \mathcal{D}_i, \mathcal{I} \rangle$ of PL¹ consists of a universe \mathcal{D}_i and an interpretation \mathcal{I} .

Semantics of PL¹ (Evaluation)

- **Definition 2.17.** Given a model $\langle \mathcal{D}, \mathcal{I} \rangle$, the **value function** \mathcal{I}_φ is recursively defined:
(two parts: terms & propositions)

- $\mathcal{I}_\varphi : wff_\iota(\Sigma_\iota) \rightarrow \mathcal{D}_\iota$ assigns values to terms.
 - $\mathcal{I}_\varphi(X) := \varphi(X)$ and
 - $\mathcal{I}_\varphi(f(A_1, \dots, A_k)) := \mathcal{I}(f)(\mathcal{I}_\varphi(A_1), \dots, \mathcal{I}_\varphi(A_k))$
- $\mathcal{I}_\varphi : wff_o(\Sigma) \rightarrow \mathcal{D}_o$ assigns values to formulae:
 - $\mathcal{I}_\varphi(T) = \mathcal{I}(T) = T$,
 - $\mathcal{I}_\varphi(\neg A) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(A))$
 - $\mathcal{I}_\varphi(A \wedge B) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(A), \mathcal{I}_\varphi(B))$
 - $\mathcal{I}_\varphi(p(A^1, \dots, A^k)) := T$, iff $\langle \mathcal{I}_\varphi(A^1), \dots, \mathcal{I}_\varphi(A^k) \rangle \in \mathcal{I}(p)$
 - $\mathcal{I}_\varphi(\forall X.A) := T$, iff $\mathcal{I}_{(\varphi, [a/X])}(A) = T$ for all $a \in \mathcal{D}_\iota$.

(just as in PL⁰)

- **Definition 2.18 (Assignment Extension).** Let φ be a variable assignment and $a \in \mathcal{D}_\iota$, then we denote with $\varphi, [a/X]$ the **extended assignment** $\{(Y, b) \in \varphi \mid Y \neq X\} \cup \{(X, a)\}$. ($\varphi, [a/X]$ coincides with φ off X , and gives the result a there)

Semantics Computation: Example

- **Example 2.19.** We define an instance of first-order logic:

- **Signature:** Let $\Sigma^f_0 := \{j, m\}$, $\Sigma^f_1 := \{f\}$, and $\Sigma^P_2 := \{o\}$
- **Universe:** $\mathcal{D}_t := \{J, M\}$
- **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 $a \in \mathcal{D}_t$ we have

$$\begin{aligned}\mathcal{I}_\varphi(\forall X.o(f(X), X)) = \top &\quad \text{iff} \quad \mathcal{I}_\psi(o(f(X), X)) = \top \text{ for all } a \in \mathcal{D}_t \\ &\quad \text{iff} \quad (\mathcal{I}_\psi(f(X)), \mathcal{I}_\psi(X)) \in \mathcal{I}(o) \text{ for all } a \in \{J, M\} \\ &\quad \text{iff} \quad (\mathcal{I}(f)(\mathcal{I}_\psi(X)), \psi(X)) \in \{(M, J)\} \text{ for all } a \in \{J, M\} \\ &\quad \text{iff} \quad (\mathcal{I}(f)(\psi(X)), a) = (M, J) \text{ for all } a \in \{J, M\} \\ &\quad \text{iff} \quad \mathcal{I}(f)(a) = M \text{ and } a = J \text{ for all } a \in \{J, M\}\end{aligned}$$

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

2.2 First-Order Substitutions

Substitutions on Terms

- ▶ **Intuition:** If B is a term and X is a variable, then we denote the result of systematically replacing all occurrences of X in a term A by B with $[B/X]A$.
- ▶ **Problem:** What about $[Z/Y],[Y/X]X$, is that Y or Z ?
- ▶ **Folklore:** $[Z/Y],[Y/X]X = Y$, but $[Z/Y][Y/X]X = Z$ of course. (Parallel application)
- ▶ **Definition 2.20.** We call $\sigma: \text{wff}_\nu(\Sigma_\nu) \rightarrow \text{wff}_\nu(\Sigma_\nu)$ a **substitution**, iff $\sigma f(A_1, \dots, A_n) = f(\sigma A_1, \dots, \sigma A_n)$ and the **support** $\text{supp}(\sigma) := \{X \mid \sigma X \neq X\}$ of σ is finite.
- ▶ **Observation 2.21.** Note that a substitution σ is determined by its values on variables alone, thus we can write σ as $\sigma|_{\nu_\nu} = \{[\sigma X/X] \mid X \in \text{supp}(\sigma)\}$.
- ▶ **Notation:** We denote the substitution σ with $\text{supp}(\sigma) = \{x^i \mid 1 \leq i \leq n\}$ and $\sigma x^i = A_i$ by $[A_1/x^1], \dots, [A_n/x^n]$.
- ▶ **Example 2.22.** $[a/x],[f(b)/y],[a/z]$ instantiates $g(x, y, h(z))$ to $g(a, f(b), h(a))$.
- ▶ **Definition 2.23.** We call $\text{intro}(\sigma) := \bigcup_{X \in \text{supp}(\sigma)} \text{free}(\sigma X)$ the set of variables **introduced** by σ .

- ▶ **Definition 2.24 (Substitution Extension).** Let σ be a substitution, then we denote with $\sigma,[A/X]$ the function $\{(Y,B) \in \sigma \mid Y \neq X\} \cup \{(X,A)\}$. ($\sigma,[A/X]$ coincides with σ off X , and gives the result A there.)
- ▶ **Note:** If σ is a substitution, then $\sigma,[A/X]$ is also a substitution.
- ▶ **Definition 2.25.** If σ is a substitution, then we call $\sigma,[A/X]$ the **extension** of σ by $[A/X]$.
- ▶ We also need the dual operation: removing a variable from the support:
- ▶ **Definition 2.26.** We can **discharge** a variable X from a substitution σ by $\sigma_{-X} := \sigma,[X/X]$.

Substitutions on Propositions

- ▶ **Problem:** We want to extend substitutions to propositions, in particular to quantified formulae: What is $\sigma(\forall X.A)$?
- ▶ **Idea:** σ should not instantiate bound variables. ($[A/X](\forall X.B) = \forall A.B'$ ill-formed)
- ▶ **Definition 2.27.** $\sigma(\forall X.A) := (\forall X.\sigma_{-X}A)$.
- ▶ **Problem:** This can lead to variable capture: $[f(X)/Y](\forall X.p(X, Y))$ would evaluate to $\forall X.p(X, f(X))$, where the second occurrence of X is bound after instantiation, whereas it was free before.
Solution: Rename away the bound variable X in $\forall X.p(X, Y)$ before applying the substitution.
- ▶ **Definition 2.28 (Capture-Avoiding Substitution Application).** Let σ be a substitution, A a formula, and A' an alphabetical variant of A , such that $\text{intro}(\sigma) \cap \text{BVar}(A) = \emptyset$. Then we define $\sigma A := \sigma A'$.

Substitution Value Lemma for Terms

- ▶ **Lemma 2.29.** Let A and B be terms, then $\mathcal{I}_\varphi([B/X]A) = \mathcal{I}_\psi(A)$, where $\psi = \varphi, [\mathcal{I}_\varphi(B)/X]$.
- ▶ **Proof:** by induction on the depth of A:

1.

1.1. **depth=0:**

1.1.1. Then A is a variable (say Y), or constant, so we have three cases

1.1.1.1. $A = Y = X$: then

$$\mathcal{I}_\varphi([B/X]A) = \mathcal{I}_\varphi([B/X]X) = \mathcal{I}_\varphi(B) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(A). \quad \square$$

1.1.1.2. $A = Y \neq X$: then $\mathcal{I}_\varphi([B/X]A) = \mathcal{I}_\varphi([B/X]Y) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(A).$ \square

1.1.1.3. A is a constant: analogous to the preceding case ($Y \neq X$) \square

1.1.2. This completes the base case (depth = 0). \square

1.2. **depth > 0:** then $A = f(A_1, \dots, A_n)$ and we have

$$\begin{aligned}\mathcal{I}_\varphi([B/X]A) &= \mathcal{I}(f)(\mathcal{I}_\varphi([B/X]A_1), \dots, \mathcal{I}_\varphi([B/X]A_n)) \\ &= \mathcal{I}(f)(\mathcal{I}_\psi(A_1), \dots, \mathcal{I}_\psi(A_n)) \\ &= \mathcal{I}_\psi(A).\end{aligned}$$

by inductive hypothesis

1.2.1. This completes the inductive case, and we have proven the assertion \square

Substitution Value Lemma for Propositions

- ▶ **Lemma 2.30.** $\mathcal{I}_\varphi([B/X]A) = \mathcal{I}_\psi(A)$, where $\psi = \varphi, [\mathcal{I}_\varphi(B)/X]$.
- ▶ **Proof:** by induction on the number n of **connectives** and **quantifiers** in A
 - 1.
 - 1.1. $n = 0$: then A is an atomic proposition, and we can argue like in the inductive case of the substitution value lemma for terms. \square
 - 1.2. $n > 0$ and $A = \neg B$ or $A = C \circ D$: Here we argue like in the inductive case of the term lemma as well. \square
 - 1.3. $n > 0$ and $A = \forall Y.C$ where (wlog) $X \neq Y$:
 - 1.3.1. then $\mathcal{I}_\psi(A) = \mathcal{I}_\psi(\forall Y.C) = \top$, iff $\mathcal{I}_{(\psi,[a/Y])}(C) = \top$ for all $a \in \mathcal{D}_\psi$.
 - 1.3.2. But $\mathcal{I}_{(\psi,[a/Y])}(C) = \mathcal{I}_{(\varphi,[a/Y])}([B/X]C) = \top$, by inductive hypothesis.
 - 1.3.3. So $\mathcal{I}_\psi(A) = \mathcal{I}_\varphi(\forall Y.[B/X]C) = \mathcal{I}_\varphi([B/X](\forall Y.C)) = \mathcal{I}_\varphi([B/X]A)$ \square

3 First-Order Natural Deduction

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

- ▶ Rules for **connectives** just as always
- ▶ **Definition 3.1 (New Quantifier Rules).** The **first-order natural deduction calculus** \mathcal{ND}^1 extends \mathcal{ND}^0 by the following four rules:

$$\frac{A}{\forall X.A} \forall I^* \quad \frac{\forall X.A}{[B/X]A} \forall E$$

$$[[c/X]A]^1$$

$$\frac{[B/X]A}{\exists X.A} \exists I \quad \frac{\exists X.A \quad \vdots \quad C \quad c \in \Sigma_0^{sk} \text{ new}}{C} \exists E^1$$

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

A Complex \mathcal{ND}^1 Example

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

$$\frac{\neg(\exists X.\neg P(X)) \vdash^1}{\frac{F}{\frac{\neg P(X) \vdash^2}{\frac{\neg\neg P(X) \vdash^3}{\frac{\neg E}{P(X) \vdash^4}}}} \frac{\forall I}{\frac{\neg(\forall X.P(X)) \vdash^5}{\frac{F}{\frac{\neg\neg(\exists X.\neg P(X)) \vdash^6}{\frac{\neg E}{\exists X.\neg P(X) \vdash^7}}}}}}$$

- ▶ Rules for **connectives** just as always
- ▶ **Definition 3.3 (New Quantifier Rules).**

$$\frac{\Gamma \vdash A \ X \notin \text{free}(\Gamma)}{\Gamma \vdash \forall X.A} \forall I$$

$$\frac{\Gamma \vdash \forall X.A}{\Gamma \vdash [B/X]A} \forall E$$

$$\frac{}{\Gamma \vdash \exists X.A} \exists I$$

$$\frac{\Gamma \vdash \exists X.A \quad \Gamma, [c/X]A \vdash C \quad c \in \Sigma_0^{sk} \text{ new}}{\Gamma \vdash C} \exists E$$

Natural Deduction with Equality

- ▶ **Definition 3.4 (First-Order Logic with Equality).** We extend PL^1 with a new logical symbol for equality $= \in \Sigma_2^P$ and fix its semantics to $\mathcal{I}(=) := \{(x, x) | x \in \mathcal{D}_t\}$. We call the extended logic **first-order logic with equality** ($\text{PL}_{=}^1$)
- ▶ We now extend natural deduction as well.
- ▶ **Definition 3.5.** For the calculus of natural deduction with equality $\text{ND}_{=}^1$ we add the following two equality rules to ND^1 to deal with equality:

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

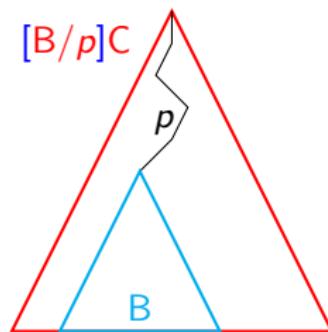
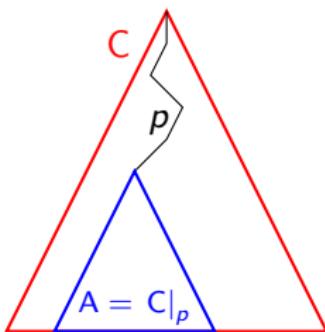
where $C[A]_p$ if the formula C has a subterm A at position p and $[B/p]C$ is the result of replacing that subterm with B .

- ▶ In many ways equivalence behaves like equality, we will use the following rules in ND^1
- ▶ **Definition 3.6.** $\Leftrightarrow I$ is derivable and $\Leftrightarrow E$ is admissible in ND^1 :

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

Positions in Formulae

- ▶ **Idea:** Formulae are (naturally) trees, so we can use tree positions to talk about subformulae
- ▶ **Definition 3.7.** A **formula position** p is a **tuple of natural numbers** that in each **node** of a **formula (tree)** specifies into which **child** to descend. For a **formula** A we denote the **subformula at p** with $A|_p$.
We will sometimes write a **formula** C as $C[A]_p$ to indicate that C the **subformula** A at **position** p .
- ▶ **Definition 3.8.** Let p be a **formula position**, then $[A/p]C$ is the **formula** obtained from C by **replacing** the **subformula at p** by A .
- ▶ **Example 3.9 (Schematically).**



ND¹ Example: $\sqrt{2}$ is Irrational

- ▶ We can do real Maths with ND¹:
- ▶ **Theorem 3.10.** $\sqrt{2}$ is irrational

Proof: We prove the assertion by contradiction

1. Assume that $\sqrt{2}$ is rational.
2. Then there are numbers p and q such that $\sqrt{2} = p/q$.
3. So we know $2q^2 = p^2$.
4. But $2q^2$ has an odd number of prime factors while p^2 an even number.
5. This is a contradiction (since they are equal), so we have proven the assertion



ND_{\equiv}^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		$\forall n, p. \text{prime}(p) \Rightarrow \#(pn) = \#(n) + 1$	lemma
4		$\forall x. \text{irr}(x) \Leftrightarrow (\neg \exists p, q. x = p/q)$	definition
5		$\text{irr}(\sqrt{2}) \Leftrightarrow (\neg \exists p, q. \sqrt{2} = p/q)$	$\forall E(4)$
6	6	$\neg \text{irr}(\sqrt{2})$	Ax
7	6	$\neg \neg (\exists p, q. \sqrt{2} = p/q)$	$\Leftrightarrow E(6, 5)$
8	6	$\exists p, q. \sqrt{2} = p/q$	$\neg E(7)$
9	6, 9	$\sqrt{2} = p/q$	Ax
10	6, 9	$2q^2 = p^2$	arith(9)
11	6, 9	$\#(p^2) = 2\#(p)$	$\forall E^2(2)$
12	6, 9	$\text{prime}(2) \Rightarrow \#(2q^2) = \#(q^2) + 1$	$\forall E^2(1)$

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

13	6,9	$\text{prime}(2)$	lemma
14	6,9	$\#(2q^2) = \#(q^2) + 1$	$\Rightarrow E(13, 12)$
15	6,9	$\#(q^2) = 2\#(q)$	$\forall E^2(2)$
16	6,9	$\#(2q^2) = 2\#(q) + 1$	$= E(14, 15)$
17		$\#(p^2) = \#(p^2)$	$= I$
18	6,9	$\#(2q^2) = \#(q^2)$	$= E(17, 10)$
19	6,9	$2\#(q) + 1 = \#(p^2)$	$= E(18, 16)$
20	6,9	$2\#(q) + 1 = 2\#(p)$	$= E(19, 11)$
21	6,9	$\neg(2\#(q) + 1) = (2\#(p))$	$\forall E^2(1)$
22	6,9	F	$FI(20, 21)$
23	6	F	$\exists E^6(22)$
24		$\neg\neg\text{irr}(\sqrt{2})$	$\neg I^6(23)$
25		$\text{irr}(\sqrt{2})$	$\neg E^2(23)$

4 Conclusion

Summary (Predicate Logic)

- ▶ Predicate logic allows to explicitly speak about objects and their properties. It is thus a more natural and compact representation language than propositional logic; it also enables us to speak about infinite sets of objects.
- ▶ Logic has thousands of years of history. A major current application in AI is Semantic Technology.
- ▶ First-order predicate logic (PL1) allows universal and existential quantification over objects.
- ▶ A PL1 interpretation consists of a universe U and a function I mapping constant symbols/predicate symbols/function symbols to elements/relations/functions on U .

Chapter 14 Automated Theorem Proving in First-Order Logic

1 First-Order Inference with Tableaux

1.1 First-Order Tableau Calculi

Test Calculi: Tableaux and Model Generation

- ▶ Idea: A tableau calculus is a test calculus that
 - ▶ analyzes a labeled formula in a tree to determine satisfiability,
 - ▶ its branches correspond to valuations (\leadsto models).
 - ▶ **Example 1.1.** Tableau calculi try to construct models for labeled formulae:

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

- ▶ **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)
 - ▶ **Satisfiable**, iff there are open branches (correspond to models)

Analytical Tableaux (Formal Treatment of \mathcal{T}_0)

- Idea: A test calculus where
 - A labeled formula is analyzed in a tree to determine satisfiability,
 - branches correspond to valuations (models)
- Definition 1.2. The propositional tableau calculus \mathcal{T}_0 has two inference rules per connective (one for each possible label)

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

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

- Definition 1.3. We call any tree ($|$ introduces branches) produced by the ProbTabCalc inference rules from a set Φ of labeled formulae a tableau for Φ .
- Definition 1.4. Call a tableau saturated, iff no rule applies, and a branch closed, iff it ends in \perp , else open.
- Idea: Open branches in saturated tableaux yield models.
- Definition 1.5 (\mathcal{T}_0 -Theorem/Derivability). A is a \mathcal{T}_0 -theorem ($\vdash_{\mathcal{T}_0} A$), iff there is a closed tableau with A^F at the root.

$\Phi \subseteq wff_o(\mathcal{V}_o)$ derivation relation A in \mathcal{T}_0 ($\Phi \vdash_{\mathcal{T}_0} A$), iff there is a closed tableau starting with A^F

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

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

- **Problem:** The rule $\mathcal{T}_1 \forall$ displays a case of “don’t know indeterminism”: to find a refutation we have to guess a formula C from the (usually infinite) set $\text{cwff}_\iota(\Sigma_\iota)$. For proof search, this means that we have to systematically try all, so $\mathcal{T}_1 \forall$ is infinitely branching in general.

Free variable Tableaux (\mathcal{T}_1^f)

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

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

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

$$\frac{\begin{array}{c} A^\alpha \\ B^\beta \end{array} \quad \alpha \neq \beta \quad \sigma A = \sigma B}{\perp : \sigma} \mathcal{T}_1^f \perp$$

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

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

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

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

Tableau Reasons about Blocks

- Example 1.9 (Reasoning about Blocks). returning to slide 386



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

$$\begin{array}{c} (\forall X.\text{block}(X) \Rightarrow \text{red}(X))^T \\ \text{block}(A)^T \\ \text{red}(A)^F \\ (\text{block}(Y) \Rightarrow \text{red}(Y))^T \\ \text{block}(Y)^F \mid \text{red}(A)^T \\ \perp : [A/Y] \mid \perp \end{array}$$

1.2 First-Order Unification

- ▶ **Definition 1.10.** For given terms A and B, **unification** is the problem of finding a substitution σ find, such that $\sigma A = \sigma B$.
- ▶ **Notation:** We write term pairs as $A=?B$ e.g. $f(X)=?f(g(Y))$.
- ▶ **Definition 1.11.** Solutions (e.g. $[g(a)/X], [a/Y], [g(g(a))/X], [g(a)/Y]$, or $[g(Z)/X], [Z/Y]$) are called **unifiers**, $U(A=?B) := \{\sigma \mid \sigma A = \sigma B\}$.
- ▶ **Idea:** Find representatives in $U(A=?B)$, that generate the set of solutions.
- ▶ **Definition 1.12.** Let σ and θ be substitutions and $W \subseteq \mathcal{V}_t$, we say that a substitution σ is **more general** than θ (on W ; write $\sigma \leq \theta[W]$), iff there is a substitution ρ , such that $\theta = (\rho \circ \sigma)[W]$, where $\sigma = \rho[W]$, iff $\sigma X = \rho X$ for all $X \in W$.
- ▶ **Definition 1.13.** σ is called a **most general unifier** of A and B, iff it is minimal in $U(A=?B)$ wrt. $\leq[(\text{free}(A) \cup \text{free}(B))]$.

Unification (Equational Systems)

- ▶ **Idea:** Unification is equation solving.
- ▶ **Definition 1.14.** We call a formula $A^1 =? B^1 \wedge \dots \wedge A^n =? B^n$ an **equational system** iff $A^i, B^i \in wff_{\iota}(\Sigma_{\iota}, \mathcal{V}_{\iota})$.
- ▶ We consider equational systems as sets of equations (\wedge is ACI), and equations as two-element multisets ($=?$ is C).

Solved forms and Most General Unifiers

- ▶ **Definition 1.15.** We call a pair $A=_?B$ **solved** in a unification problem \mathcal{E} , iff $A = X$, $\mathcal{E} = X=_?A \wedge \mathcal{E}$, and $X \notin \text{free}(A) \cup \text{free}(\mathcal{E})$. We call an unification problem \mathcal{E} a **solved form**, iff all its pairs are solved.
- ▶ **Lemma 1.16.** Solved forms are of the form $X^1=_?B^1 \wedge \dots \wedge X^n=_?B^n$ where the X^i are distinct and $X^i \notin \text{free}(B^j)$.
- ▶ **Definition 1.17.** Any substitution $\sigma = [B^1/X^1], \dots, [B^n/X^n]$ induces a solved unification problem $\mathcal{E}_\sigma := (X^1=_?B^1 \wedge \dots \wedge X^n=_?B^n)$.
- ▶ **Lemma 1.18.** If $\mathcal{E} = X^1=_?B^1 \wedge \dots \wedge X^n=_?B^n$ is a solved form, then \mathcal{E} has the unique most general unifier $\sigma_\mathcal{E} := [B^1/X^1], \dots, [B^n/X^n]$.
- ▶ **Proof:** Let $\theta \in U(\mathcal{E})$
 1. then $\theta X^i = \theta B^i = \theta \circ \sigma_\mathcal{E} X^i$
 2. and thus $\theta = (\theta \circ \sigma_\mathcal{E})[\text{supp}(\sigma)]$.
- ▶ **Note:** we can rename the introduced variables in most general unifiers!

□

Unification Algorithm

- **Definition 1.19.** The inference system \mathcal{U} consists of the following rules:

$$\frac{\mathcal{E} \wedge f(A^1, \dots, A^n) =? f(B^1, \dots, B^n)}{\mathcal{E} \wedge A^1 =? B^1 \wedge \dots \wedge A^n =? B^n} \mathcal{U}_{\text{dec}} \quad \frac{\mathcal{E} \wedge A =? A}{\mathcal{E}} \mathcal{U}_{\text{triv}}$$

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

- **Lemma 1.20.** \mathcal{U} is **correct**: $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ implies $\mathbf{U}(\mathcal{F}) \subseteq \mathbf{U}(\mathcal{E})$.
- **Lemma 1.21.** \mathcal{U} is **complete**: $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ implies $\mathbf{U}(\mathcal{E}) \subseteq \mathbf{U}(\mathcal{F})$.
- **Lemma 1.22.** \mathcal{U} is confluent: the order of derivations does not matter.
- **Corollary 1.23.** First-Order Unification is **unitary**: i.e. most general unifiers are unique up to renaming of introduced variables.
- **Proof Sketch:** the inference system \mathcal{U} is trivially branching □

Unification Examples

- Example 1.24. Two similar unification problems:

$$\begin{array}{c} f(g(X, X), h(a)) = ? f(g(a, Z), h(Z)) \\ \hline g(X, X) = ? g(a, Z) \wedge h(a) = ? h(Z) \quad \text{Udec} \\ \hline X = ? a \wedge X = ? Z \wedge h(a) = ? h(Z) \quad \text{Udec} \\ \hline X = ? a \wedge X = ? Z \wedge a = ? Z \quad \text{Udec} \\ \hline X = ? a \wedge a = ? Z \wedge a = ? Z \quad \text{Uelim} \\ \hline X = ? a \wedge Z = ? a \wedge a = ? a \quad \text{Uelim} \\ \hline X = ? a \wedge Z = ? a \quad \text{Utriv} \\ \hline X = ? a \wedge Z = ? a \end{array}$$

MGU: $[a/X], [a/Z]$

$$\begin{array}{c} f(g(X, X), h(a)) = ? f(g(b, Z), h(Z)) \\ \hline g(X, X) = ? g(b, Z) \wedge h(a) = ? h(Z) \quad \text{Udec} \\ \hline X = ? b \wedge X = ? Z \wedge h(a) = ? h(Z) \quad \text{Udec} \\ \hline X = ? b \wedge X = ? Z \wedge a = ? Z \quad \text{Udec} \\ \hline X = ? b \wedge b = ? Z \wedge a = ? Z \quad \text{Uelim} \\ \hline X = ? b \wedge Z = ? b \wedge a = ? b \quad \text{Uelim} \\ \hline X = ? b \wedge Z = ? b \end{array}$$

$a = ? b$ not unifiable

Unification (Termination)

- ▶ **Definition 1.25.** Let S and T be multisets and \prec 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 $s \prec t$ for all $s \in S'$. We call \prec^m the **multiset ordering** induced by \prec .
- ▶ **Lemma 1.26.** If \prec is linear/terminating on S , then \prec^m is linear/terminating on $\mathcal{P}(S)$.
- ▶ **Lemma 1.27.** \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. $\mathcal{U}\text{dec}$ and $\mathcal{U}\text{triv}$ decrease the multiset of term depths without increasing the unsolved variables
 - 2.2. $\mathcal{U}\text{elim}$ decreases the number of unsolved variables (by one), but may increase term depths.



First-Order Unification is Decidable

- ▶ **Definition 1.28.** We call an equational problem \mathcal{E} **\mathcal{U} -reducible**, iff there is a \mathcal{U} -step $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ from \mathcal{E} .
- ▶ **Lemma 1.29.** 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 $A = ?B$ in $\mathcal{E} = \mathcal{E} \wedge A = ?B'$.
2. we have two cases
- 2.1. $A, B \notin \mathcal{V}_\mathcal{U}$: then $A = f(A^1 \dots A^n)$ and $B = f(B^1 \dots B^n)$, and thus \mathcal{U}_{dec} is applicable \square
- 2.2. $A = X \in \text{free}(\mathcal{E})$: then \mathcal{U}_{elim} (if $B \neq X$) or \mathcal{U}_{triv} (if $B = X$) is applicable. \square

- ▶ **Corollary 1.30.** First-order unification is **decidable** in PL^1 .

- ▶ **Proof:**

1. \mathcal{U} -irreducible sets of equations can be obtained in finite time by termination.
2. They are either solved or unsolvable by , so they provide the answer. \square

1.3 Efficient Unification

Complexity of Unification

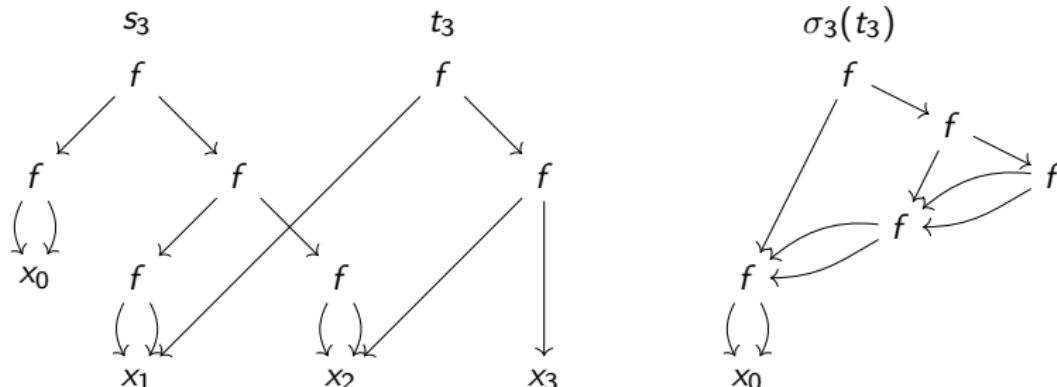
- ▶ **Observation:** Naive implementations of unification are exponential in time and space.
- ▶ **Example 1.31.** Consider the terms

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

- ▶ The most general unifier of s_n and t_n is
 $\sigma_n := [f(x_0, x_0)/x_1, [f(f(x_0, x_0), f(x_0, x_0))/x_2, [f(f(f(x_0, x_0), f(x_0, x_0)), f(f(x_0, x_0), f(x_0, x_0))) / x_3], \dots]$
- ▶ It contains $\sum_{i=1}^n 2^i = 2^{n+1} - 2$ occurrences of the variable x_0 . (exponential)
- ▶ **Problem:** The variable x_0 has been copied too often.
- ▶ **Idea:** Find a term representation that re-uses subterms.

Directed Acyclic Graphs (DAGs) for Terms

- ▶ Recall: Terms in first order logic are essentially trees.
- ▶ Concrete Idea: Use directed acyclic graphs for representing terms:
 - ▶ variables may only occur once in the DAG.
 - ▶ subterms can be referenced multiply. (subterm sharing)
 - ▶ we can even represent multiple terms in a common DAG
- ▶ **Observation 1.32.** Terms can be transformed into DAGs in linear time.
- ▶ **Example 1.33.** Continuing from 1.31 ... s_3 , t_3 , and $\sigma_3(s_3)$ as DAGs:



In general: s_n , t_n , and $\sigma_n(s_n)$ only need space in $\mathcal{O}(n)$. (just count)

DAG Unification Algorithm

- ▶ **Observation:** In \mathcal{U} , the $\mathcal{U}\text{elim}$ rule applies solved pairs \rightsquigarrow subterm duplication.
- ▶ **Idea:** Replace $\mathcal{U}\text{elim}$ the notion of solved forms by something better.
- ▶ **Definition 1.34.** We say that $X^1=?B^1 \wedge \dots \wedge X^n=?B^n$ is a **DAG solved form**, iff the X^i are distinct and $X^i \notin \text{free}(B^j)$ for $i \leq j$.
- ▶ **Definition 1.35.** The inference system \mathcal{DU} contains rules $\mathcal{U}\text{dec}$ and $\mathcal{U}\text{triv}$ from \mathcal{U} plus the following:

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

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

where $|A|$ is the number of symbols in A.

- ▶ The analysis for \mathcal{U} applies mutatis mutandis.

- ▶ Idea: Extend the Input-DAGs by edges that represent unifiers.
- ▶ write $n.a$, if a is the symbol of node n .
- ▶ (standard) auxiliary procedures: (all constant or linear time in DAGs)
 - ▶ $\text{find}(n)$ follows the path from n and returns the end node.
 - ▶ $\text{union}(n, m)$ adds an edge between n and m .
 - ▶ $\text{occur}(n, m)$ 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  $g \neq f$  then false
  else
    forall ( $i, j$ ) => dag-unify(find( $i$ ), find( $j$ )) (chld  $m$ , chld  $n$ )
  end
```

- ▶ **Observation 1.36.** dag-unify uses linear space, since no new nodes are created, and at most one link per variable.
- ▶ **Problem:** dag-unify still uses exponential time.
- ▶ **Example 1.37.** 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)

- ▶ **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 runtime.
- ▶ **Remark:** There are versions of uf-unify that are linear in time and space, but for most purposes, our algorithm suffices.

1.4 Implementing First-Order Tableaux

Termination and Multiplicity in Tableaux

- ▶ Recall: In \mathcal{T}_0 , all rules only needed to be applied once.
~ \mathcal{T}_0 terminates and thus induces a decision procedure for PL^0 .
- ▶ **Observation 1.38.** All \mathcal{T}_1^f rules except $\mathcal{T}_1^f \vee$ only need to be applied once.

Termination and Multiplicity in Tableaux

- ▶ **Recall:** In \mathcal{T}_0 , all rules only needed to be applied once.
~ \mathcal{T}_0 terminates and thus induces a decision procedure for PL^0 .
- ▶ **Observation 1.38.** All \mathcal{T}_1^f rules except $\mathcal{T}_1^f \forall$ only need to be applied once.
- ▶ **Example 1.39.** A tableau proof for $p(a) \vee p(b) \Rightarrow (\exists x.p(x))$.

Start, close left branch	use $\mathcal{T}_1^f \forall$ again (right branch)
$(p(a) \vee p(b) \Rightarrow (\exists x.p(x)))^F$ $(p(a) \vee p(b))^T$ $(\exists x.p(x))^F$ $(\forall x.\neg p(x))^T$ $\neg p(y)^T$ $p(y)^F$ $p(a)^T \quad \quad p(b)^T$ $\perp : [a/y]$	$(p(a) \vee p(b) \Rightarrow (\exists x.p(x)))^F$ $(p(a) \vee p(b))^T$ $(\exists x.p(x))^F$ $(\forall x.\neg p(x))^T$ $\neg p(a)^T$ $p(a)^F$ $p(a)^T \quad \quad p(b)^T$ $\perp : [a/y] \quad \perp : [b/z]$ $\neg p(z)^T$ $p(z)^F$

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

Termination and Multiplicity in Tableaux

- ▶ **Recall:** In \mathcal{T}_0 , all rules only needed to be applied once.
~ \mathcal{T}_0 terminates and thus induces a decision procedure for PL^0 .
- ▶ **Observation 1.38.** All \mathcal{T}_1^f rules except $\mathcal{T}_1^f \forall$ only need to be applied once.
- ▶ **Example 1.39.** A tableau proof for $p(a) \vee p(b) \Rightarrow (\exists x.p(x))$.
- ▶ **Definition 1.40.** Let \mathcal{T} be a tableau for A, and a positive occurrence of $\forall x.B$ in A, then we call the number of applications of $\mathcal{T}_1^f \forall$ to $\forall x.B$ its **multiplicity**.
- ▶ **Observation 1.41.** Given a prescribed multiplicity for each positive \forall , saturation with \mathcal{T}_1^f terminates.
- ▶ **Proof Sketch:** All \mathcal{T}_1^f rules reduce the number of **connectives** and negative \forall or the multiplicity of positive \forall . □

Termination and Multiplicity in Tableaux

- ▶ **Recall:** In \mathcal{T}_0 , all rules only needed to be applied once.
 $\leadsto \mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0 .
- ▶ **Observation 1.38.** All \mathcal{T}_1^f rules except $\mathcal{T}_1^f \forall$ only need to be applied once.
- ▶ **Example 1.39.** A tableau proof for $p(a) \vee p(b) \Rightarrow (\exists x.p(x))$.
- ▶ **Definition 1.40.** Let \mathcal{T} be a tableau for A , and a positive occurrence of $\forall x.B$ in A , then we call the number of applications of $\mathcal{T}_1^f \forall$ to $\forall x.B$ its **multiplicity**.
- ▶ **Observation 1.41.** Given a prescribed multiplicity for each positive \forall , saturation with \mathcal{T}_1^f terminates.
- ▶ **Proof Sketch:** All \mathcal{T}_1^f rules reduce the number of **connectives** and negative \forall or the multiplicity of positive \forall . □
- ▶ **Theorem 1.42.** \mathcal{T}_1^f is only complete with unbounded **multiplicities**. □
- ▶ **Proof Sketch:** Replace $p(a) \vee p(b)$ with $p(a_1) \vee \dots \vee p(a_n)$ in 1.39. □

Termination and Multiplicity in Tableaux

- ▶ **Recall:** In \mathcal{T}_0 , all rules only needed to be applied once.
 $\leadsto \mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0 .
- ▶ **Observation 1.38.** All \mathcal{T}_1^f rules except $\mathcal{T}_1^f \forall$ only need to be applied once.
- ▶ **Example 1.39.** A tableau proof for $p(a) \vee p(b) \Rightarrow (\exists x.p(x))$.
- ▶ **Definition 1.40.** Let \mathcal{T} be a tableau for A, and a positive occurrence of $\forall x.B$ in A, then we call the number of applications of $\mathcal{T}_1^f \forall$ to $\forall x.B$ its **multiplicity**.
- ▶ **Observation 1.41.** Given a prescribed multiplicity for each positive \forall , saturation with \mathcal{T}_1^f terminates.
- ▶ **Proof Sketch:** All \mathcal{T}_1^f rules reduce the number of **connectives** and negative \forall or the multiplicity of positive \forall . □
- ▶ **Theorem 1.42.** \mathcal{T}_1^f is only complete with unbounded **multiplicities**. □
- ▶ **Proof Sketch:** Replace $p(a) \vee p(b)$ with $p(a_1) \vee \dots \vee p(a_n)$ in 1.39. □
- ▶ **Remark:** Otherwise validity in PL^1 would be **decidable**.
- ▶ **Implementation:** We need an iterative **multiplicity**-deepening process.

Treating $\mathcal{T}_1^f \perp$

- ▶ Recall: The $\mathcal{T}_1^f \perp$ rule instantiates the whole tableau.
- ▶ Problem: There may be more than one $\mathcal{T}_1^f \perp$ opportunity on a branch.
- ▶ Example 1.43. Choosing which matters – this tableau does not close!

$$\begin{array}{c} (\exists x. (p(a) \wedge p(b) \Rightarrow p(x)) \wedge (q(b) \Rightarrow q(x)))^F \\ (p(a) \wedge p(b) \Rightarrow p(y) \wedge (q(b) \Rightarrow q(y)))^F \\ (p(a) \Rightarrow p(b) \Rightarrow p(y))^F \quad | \quad (q(b) \Rightarrow q(y))^F \\ p(a)^T \quad \quad \quad q(b)^T \\ p(b)^T \quad \quad \quad q(y)^F \\ p(y)^F \\ \perp : [a/y] \end{array}$$

choosing the other $\mathcal{T}_1^f \perp$ in the left branch allows closure.

- ▶ Idea: Two ways of systematic proof search in \mathcal{T}_1^f :
 - ▶ backtracking search over $\mathcal{T}_1^f \perp$ opportunities
 - ▶ saturate without $\mathcal{T}_1^f \perp$ and find spanning matings
- (next slide)

Spanning Matings for $\mathcal{T}_1^f \perp$

► **Observation 1.44.** \mathcal{T}_1^f without $\mathcal{T}_1^f \perp$ is terminating and confluent for given multiplicities.

► **Idea:** Saturate without $\mathcal{T}_1^f \perp$ and treat all cuts at the same time (later).

► **Definition 1.45.** Let \mathcal{T} be a \mathcal{T}_1^f tableau, then we call a unification problem $\mathcal{E} := A_1 =? B_1 \wedge \dots \wedge A_n =? B_n$ a **mating** for \mathcal{T} , iff A_i^T and B_i^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 A_i^T and B_i^F for some i .

► **Theorem 1.46.** 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 $\mathcal{T}_1^f \perp$, backtracking search for **spanning matings** with \mathcal{DU} , adding pairs incrementally.

2 First-Order Resolution

First-Order Resolution (and CNF)

- **Definition 2.1.** The calculus CNF^1 is given by the inference rules of CNF^0 extended by the following quantifier rules:

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

$$\frac{(\forall X.A)^F \vee C \ \{X_1, \dots, X_k\} = \text{free}(\forall X.A) \ f \in \Sigma_k^{sk} \text{ new}}{([f(X^1, \dots, X^k)/X]A)^F \vee C}$$

$\text{CNF}^1(\Phi)$ is the set of all clauses that can be derived from Φ .

- **Definition 2.2 (First-Order Resolution Calculus).** **First order resolution** is a test calculus that manipulates formulae in conjunctive normal form. \mathcal{R}^1 has two inference rules:

$$\frac{A^T \vee C \ B^F \vee D \ \sigma = \text{mgu}(A, B)}{(\sigma C) \vee (\sigma D)}$$

$$\frac{A^\alpha \vee B^\alpha \vee C \ \sigma = \text{mgu}(A, B)}{(\sigma A) \vee (\sigma C)}$$

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

$$\frac{(\exists X.A)^T \vee C \quad \{X_1, \dots, X_k\} = \text{free}(\forall X.A) \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X^1, \dots, X^k)/X]A)^T \vee C}$$

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

2.1 Resolution Examples

► **Example 2.4.** From [RN09]

The law says it is a crime for an American to sell weapons to hostile nations.

The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

Prove that Col. West is a criminal.

- **Remark:** Modern resolution theorem provers prove this in less than 50ms.
- **Problem:** That is only true, if we **only** give the theorem prover exactly the right laws and background knowledge. If we give it all of them, it drowns in the combinatory 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 $P^F \vee Q^F \vee R^F \vee S^T$.

Col. West, a Criminal?

- ▶ *It is a crime for an American to sell weapons to hostile nations:*

Clause: $\text{ami}(X_1) \wedge \text{weap}(Y_1) \wedge \text{sell}(X_1, Y_1, Z_1) \wedge \text{host}(Z_1) \Rightarrow \text{crook}(X_1)$

Col. West, a Criminal?

- ▶ It is a crime for an American to sell weapons to hostile nations:

Clause: $\text{ami}(X_1) \wedge \text{weap}(Y_1) \wedge \text{sell}(X_1, Y_1, Z_1) \wedge \text{host}(Z_1) \Rightarrow \text{crook}(X_1)$

- ▶ Nono has some missiles: $\exists X. \text{own}(\text{NN}, X) \wedge \text{mle}(X)$

Clauses: $\text{own}(\text{NN}, c)^T$ and $\text{mle}(c)$ (c is Skolem constant)

Col. West, a Criminal?

- ▶ It is a crime for an American to sell weapons to hostile nations:

Clause: $\text{ami}(X_1) \wedge \text{weap}(Y_1) \wedge \text{sell}(X_1, Y_1, Z_1) \wedge \text{host}(Z_1) \Rightarrow \text{crook}(X_1)$

- ▶ Nono has some missiles: $\exists X. \text{own}(\text{NN}, X) \wedge \text{mle}(X)$

Clauses: $\text{own}(\text{NN}, c)^T$ and $\text{mle}(c)$ (c is Skolem constant)

- ▶ All of Nono's missiles were sold to it by Colonel West.

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

Col. West, a Criminal?

- ▶ It is a crime for an American to sell weapons to hostile nations:

Clause: $\text{ami}(X_1) \wedge \text{weap}(Y_1) \wedge \text{sell}(X_1, Y_1, Z_1) \wedge \text{host}(Z_1) \Rightarrow \text{crook}(X_1)$

- ▶ Nono has some missiles: $\exists X. \text{own}(\text{NN}, X) \wedge \text{mle}(X)$

Clauses: $\text{own}(\text{NN}, c)^T$ and $\text{mle}(c)$ (c is Skolem constant)

- ▶ All of Nono's missiles were sold to it by Colonel West.

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

- ▶ Missiles are weapons:

Clause: $\text{mle}(X_3) \Rightarrow \text{weap}(X_3)$

Col. West, a Criminal?

- ▶ It is a crime for an American to sell weapons to hostile nations:

Clause: $\text{ami}(X_1) \wedge \text{weap}(Y_1) \wedge \text{sell}(X_1, Y_1, Z_1) \wedge \text{host}(Z_1) \Rightarrow \text{crook}(X_1)$

- ▶ Nono has some missiles: $\exists X. \text{own}(\text{NN}, X) \wedge \text{mle}(X)$

Clauses: $\text{own}(\text{NN}, c)^T$ and $\text{mle}(c)$ (c is Skolem constant)

- ▶ All of Nono's missiles were sold to it by Colonel West.

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

- ▶ Missiles are weapons:

Clause: $\text{mle}(X_3) \Rightarrow \text{weap}(X_3)$

- ▶ An enemy of America counts as "hostile":

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

Col. West, a Criminal?

- ▶ It is a crime for an American to sell weapons to hostile nations:

Clause: $\text{ami}(X_1) \wedge \text{weap}(Y_1) \wedge \text{sell}(X_1, Y_1, Z_1) \wedge \text{host}(Z_1) \Rightarrow \text{crook}(X_1)$

- ▶ Nono has some missiles: $\exists X. \text{own}(\text{NN}, X) \wedge \text{mle}(X)$

Clauses: $\text{own}(\text{NN}, c)^T$ and $\text{mle}(c)$ (c is Skolem constant)

- ▶ All of Nono's missiles were sold to it by Colonel West.

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

- ▶ Missiles are weapons:

Clause: $\text{mle}(X_3) \Rightarrow \text{weap}(X_3)$

- ▶ An enemy of America counts as "hostile":

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

- ▶ West is an American:

Clause: $\text{ami}(\text{West})$

Col. West, a Criminal?

- ▶ It is a crime for an American to sell weapons to hostile nations:

Clause: $\text{ami}(X_1) \wedge \text{weap}(Y_1) \wedge \text{sell}(X_1, Y_1, Z_1) \wedge \text{host}(Z_1) \Rightarrow \text{crook}(X_1)$

- ▶ Nono has some missiles: $\exists X. \text{own}(\text{NN}, X) \wedge \text{mle}(X)$

Clauses: $\text{own}(\text{NN}, c)^T$ and $\text{mle}(c)$ (c is Skolem constant)

- ▶ All of Nono's missiles were sold to it by Colonel West.

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

- ▶ Missiles are weapons:

Clause: $\text{mle}(X_3) \Rightarrow \text{weap}(X_3)$

- ▶ An enemy of America counts as "hostile":

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

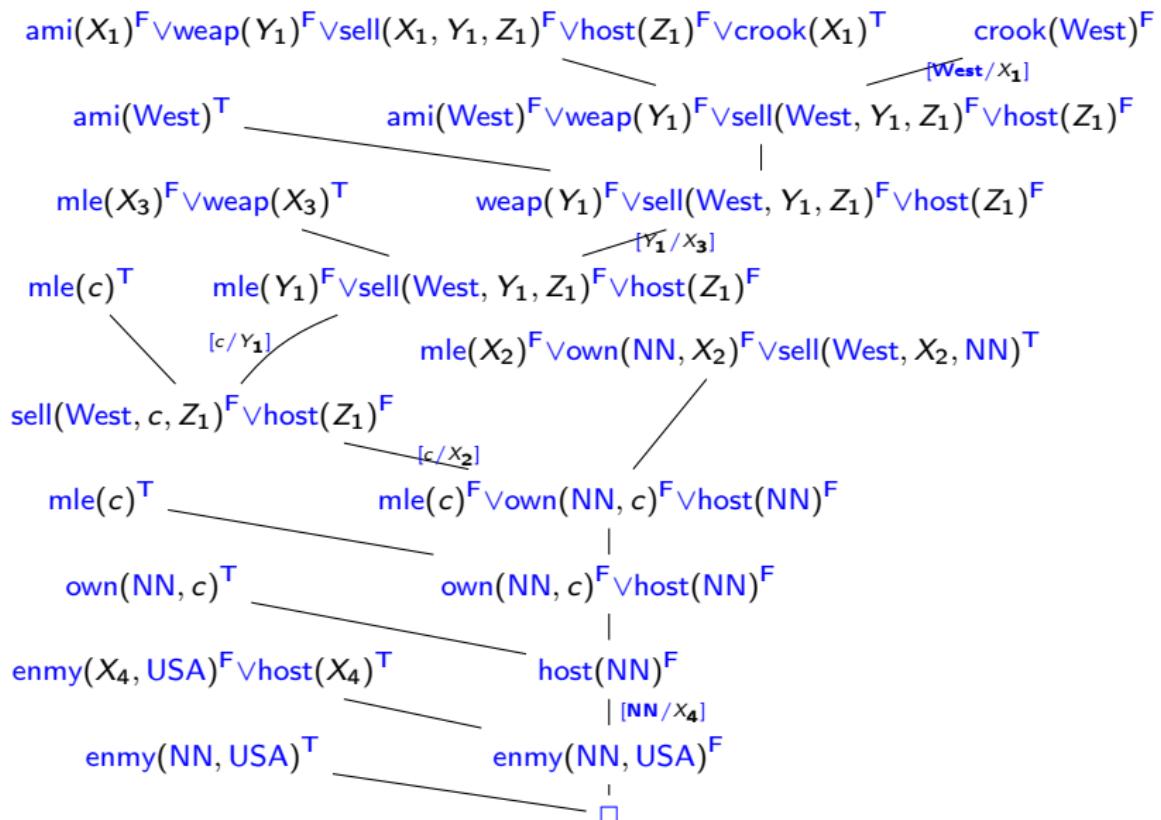
- ▶ West is an American:

Clause: $\text{ami}(\text{West})$

- ▶ The country Nono is an enemy of America:

$\text{enmy}(\text{NN}, \text{USA})$

Col. West, a Criminal! PL1 Resolution Proof



Curiosity Killed the Cat?

► Example 2.5. From [RN09]

Everyone who loves all animals is loved by someone.

Anyone who kills an animal is loved by no one.

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:*

$\forall X.(\forall Y.\text{animal}(Y) \Rightarrow \text{love}(X, Y)) \Rightarrow (\exists Z.\text{love}(Z, X))$

Clauses: $\text{animal}(g(X_1))^T \vee \text{love}(g(X_1), X_1)^T$ and
 $\text{love}(X_2, f(X_2))^F \vee \text{love}(g(X_2), X_2)^T$

- ▶ *Anyone who kills an animal is loved by no one:*

$\forall X. \exists Y. \text{animal}(Y) \wedge \text{kill}(X, Y) \Rightarrow (\forall Z. \neg \text{love}(Z, X))$

Clause: $\text{animal}(Y_3)^F \vee \text{kill}(X_3, Y_3)^F \vee \text{love}(Z_3, X_3)^F$

- ▶ *Jack loves all animals:*

Clause: $\text{animal}(X_4)^F \vee \text{love}(\text{jack}, X_4)^T$

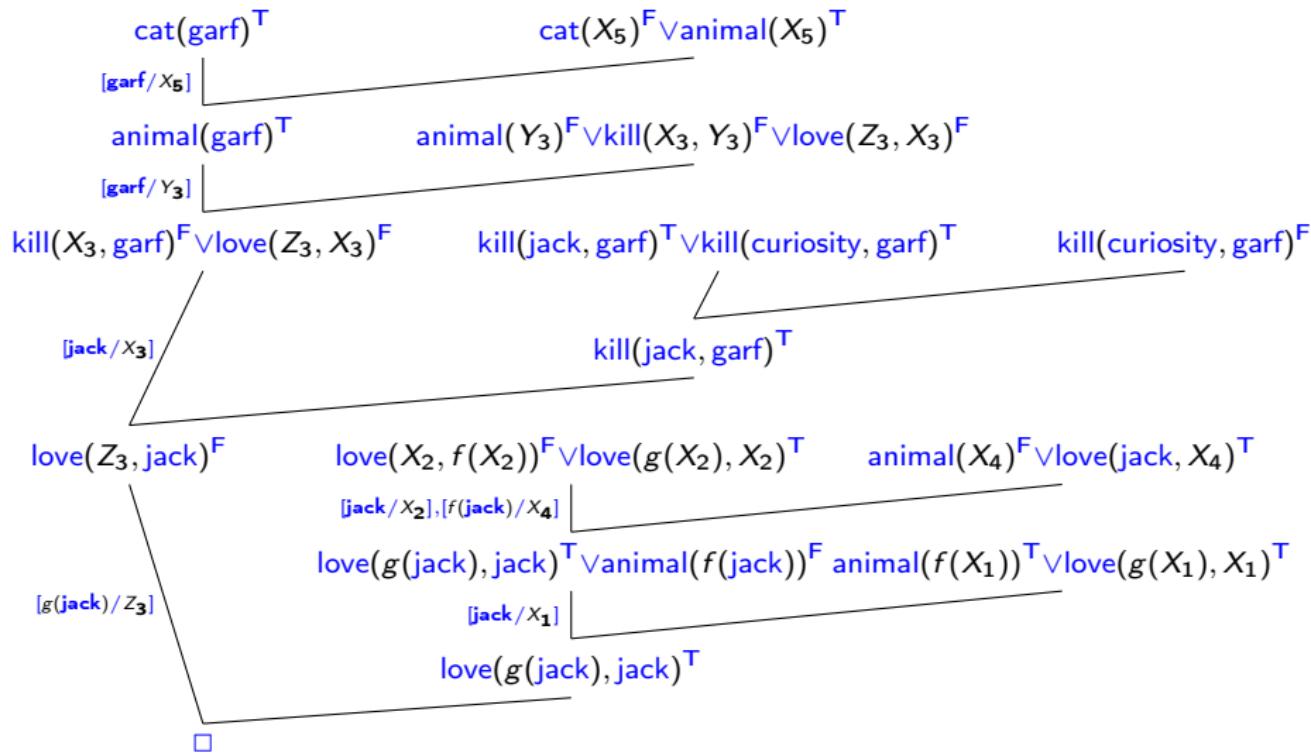
- ▶ *Cats are animals:*

Clause: $\text{cat}(X_5)^F \vee \text{animal}(X_5)^T$

- ▶ *Either Jack or curiosity killed the cat (whose name is "Garfield"):*

Clauses: $\text{kill}(\text{jack}, \text{garf})^T \vee \text{kill}(\text{curiosity}, \text{garf})^T$ and $\text{cat}(\text{garf})^T$

Curiosity Killed the Cat! PL1 Resolution Proof



3 Logic Programming as Resolution Theorem Proving

We know all this already

- ▶ Goals, goal-sets, rules, and facts are just clauses. (called “Horn clauses”)
- ▶ **Observation 3.1 (Rule).** $H:-B_1, \dots, B_n.$ corresponds to $H^T \vee B_1^F \vee \dots \vee B_n^F$ (head the only positive literal)
- ▶ **Observation 3.2 (Goal set).** $?-G_1, \dots, G_n.$ corresponds to $G_1^F \vee \dots \vee G_n^F$
- ▶ **Observation 3.3 (Fact).** $F.$ corresponds to the unit clause $F^T.$
- ▶ **Definition 3.4.** A **Horn clause** is a clause with at most one positive literal.
- ▶ Recall: Backchaining as search:
 - ▶ state = tuple of goals; goal state = empty list (of goals).
 - ▶ $\text{next}(\langle G, R_1, \dots, R_l \rangle) := \langle \sigma B_1, \dots, \sigma B_m, \sigma R_1, \dots, \sigma R_l \rangle$ if there is a rule $H:-B_1, \dots, B_m.$ and a substitution σ with $\sigma H = \sigma G.$
- ▶ Note: Backchaining becomes resolution

$$\frac{P^T \vee A \quad P^F \vee B}{A \vee B}$$

positive, unit-resulting hyperresolution (PURR)

- ▶ **Definition 3.5.** A clause is called a **Horn clause**, iff contains at most one positive literal, i.e. if it is of the form $B_1^F \vee \dots \vee B_n^F \vee A^T$ – i.e. $A:-B_1, \dots, B_n$. in ProLog notation.
 - ▶ **Rule clause:** general case, e.g. fallible(X) :- human(X).
 - ▶ **Fact clause:** no negative literals, e.g. human(sokrates).
 - ▶ **Program:** set of rule and fact clauses.
 - ▶ **Query:** no positive literals: e.g. ?— fallible(X),greek(X).
- ▶ **Definition 3.6.** **Horn logic** is the **formal system** whose language is the set of **Horn clauses** together with the **calculus \mathcal{H}** given by **MP**, **$\wedge I$** , and **Subst**.
- ▶ **Definition 3.7.** A **logic program P** **entails** a **query Q** with **answer substitution σ** , iff there is a **\mathcal{H}** derivation D of Q from P and σ is the combined substitution of the **Subst** instances in D .

PROLOG: Our Example

► Program:

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

- Example 3.8 (Query). ?— fallible(X),greek(X).
► Answer substitution: [sokrates/X]

Knowledge Base (Example)

- ▶ **Example 3.9.** $\text{car}(c)$. is in the knowledge base generated by

$\text{has_motor}(c)$.

$\text{has_wheels}(c, 4)$.

$\text{car}(X) :- \text{has_motor}(X), \text{has_wheels}(X, 4)$.

$$\frac{\frac{m(c) \quad w(c, 4)}{m(c) \wedge w(c, 4)} V!I \quad \frac{m(x) \wedge w(x, 4) \Rightarrow \text{car}(x)}{m(c) \wedge w(c, 4) \Rightarrow \text{car}(c)} \text{Subst}}{\text{car}(c)} \text{MP}$$

Why Only Horn Clauses?

- ▶ General clauses of the form $A_1, \dots, A_n :- B_1, \dots, B_n$.
- ▶ e.g. `greek(sokrates),greek(perikles)`
 - ▶ **Question:** Are there fallible greeks?
 - ▶ **Indefinite answer:** Yes, Perikles or Sokrates
 - ▶ **Warning:** how about **Sokrates and Perikles?**
- ▶ e.g. `greek(sokrates),roman(sokrates):-.`
 - ▶ **Query:** Are there fallible greeks?
 - ▶ **Answer:** Yes, Sokrates, if he is not a roman
 - ▶ **Is this abduction?????**

Three Principal Modes of Inference

- ▶ **Definition 3.10.** **Deduction** $\hat{=}$ knowledge extension

$$\frac{\text{rains} \Rightarrow \text{wet_street} \quad \text{rains}}{\text{wet_street}} D$$

- ▶ **Definition 3.11.** **Abduction** $\hat{=}$ explanation

$$\frac{\text{rains} \Rightarrow \text{wet_street} \quad \text{wet_street}}{\text{rains}} A$$

- ▶ **Definition 3.12.** **Induction** $\hat{=}$ learning rules

$$\frac{\text{wet_street} \quad \text{rains}}{\text{rains} \Rightarrow \text{wet_street}} I$$

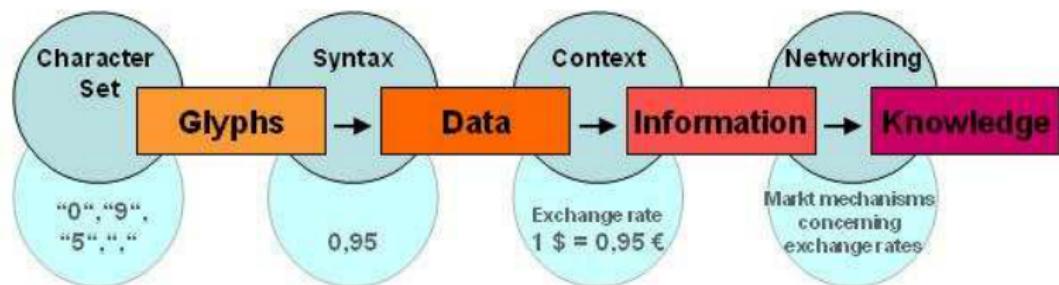
Chapter 15 Knowledge Representation and the Semantic Web

1 Introduction to Knowledge Representation

1.1 Knowledge & Representation

What is knowledge? Why Representation?

- ▶ 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 or grammatical function
structure	function

Knowledge Representation vs. Data Structures

- ▶ **Idea:** Representation as structure **and** function.
 - ▶ the **representation** determines the content theory (what is the data?)
 - ▶ the **function** determines the process model (what do we do with the data?)
- ▶ **Question:** Why do we use the term "knowledge representation" rather than
 - ▶ data structures? (sets, lists, ... above)
 - ▶ information representation? (it is information)
- ▶ **Answer:** No good reason other than AI practice, with the intuition that
 - ▶ data is simple and general (supports many algorithms)
 - ▶ knowledge is complex (has distinguished process model)

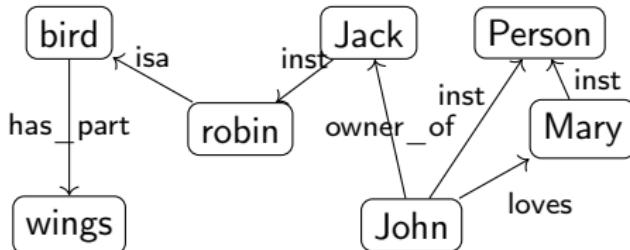
- ▶ GOFAI (good old-fashioned AI)
 - ▶ symbolic knowledge representation, process model based on heuristic search
- ▶ Statistical, corpus-based approaches.
 - ▶ symbolic representation, process model based on machine learning
 - ▶ knowledge is divided into symbolic- and statistical (search) knowledge
- ▶ The connectionist approach
 - ▶ sub-symbolic representation, process model based on primitive processing elements (nodes) and weighted links
 - ▶ knowledge is only present in activation patterns, etc.

- ▶ **Definition 1.1.** The **evaluation criteria** for knowledge representation approaches are:
 - ▶ **expressive adequacy:** What can be represented, what distinctions are supported.
 - ▶ **reasoning efficiency:** Can the representation support processing that generates results in acceptable speed?
 - ▶ **primitives:** What are the primitive elements of representation, are they intuitive, cognitively adequate?
 - ▶ **meta representation:** Knowledge about knowledge
 - ▶ **completeness:** The problems of reasoning with knowledge that is known to be incomplete.

1.2 Semantic Networks

Semantic Networks [CQ69]

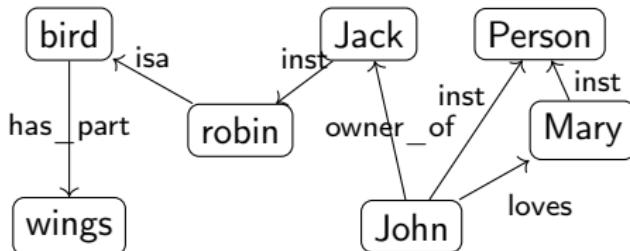
- ▶ **Definition 1.2.** A **semantic network** is a **directed graph** for representing knowledge:
 - ▶ nodes represent **objects** and **concepts** (classes of **objects**)
(e.g. John (**object**) and bird (**concept**))
 - ▶ edges (called **links**) represent relations between these (**isa**, **father_of**, **belongs_to**)
- ▶ **Example 1.3.** A **semantic network** for birds and persons:



- ▶ **Problem:** How do we derive new information from such a network?
- ▶ **Idea:** Encode taxonomic information about **objects** and **concepts** in special **links** ("isa" and "inst") and specify property inheritance along them in the process model.

Deriving Knowledge Implicit in Semantic Networks

- ▶ **Observation 1.4.** There is more knowledge in a semantic network than is explicitly written down.
- ▶ **Example 1.5.** In the network below, we “know” that *robins have wings* and in particular, *Jack has wings*.



- ▶ Idea: Links labeled with “isa” and “inst” are special: they propagate properties encoded by other links.
- ▶ **Definition 1.6.** We call links labeled by

- ▶ “isa” an **inclusion** or **isa link** (inclusion of concepts)
- ▶ “inst” **instance** or **inst link** (concept membership)

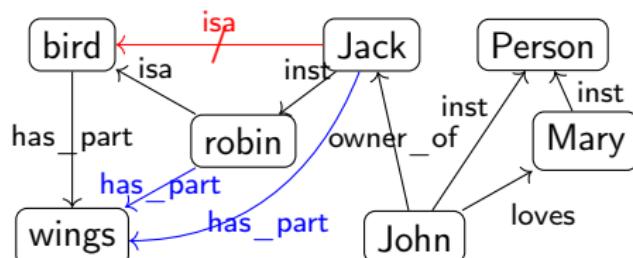
Deriving Knowledge Semantic Networks

- ▶ **Definition 1.7 (Inference in Semantic Networks).** We call all link labels except “inst” and “isa” in a semantic network **relations**.

Let N be a semantic network and R a relation in N such that $A \xrightarrow{\text{isa}} B \xrightarrow{R} C$ or $A \xrightarrow{\text{inst}} B \xrightarrow{R} C$, then we can **derive** a relation $A \xrightarrow{R} C$ in N .

The process of deriving new concepts and relations from existing ones is called **inference** and concepts/relations that are only available via inference **implicit** (in a semantic network).

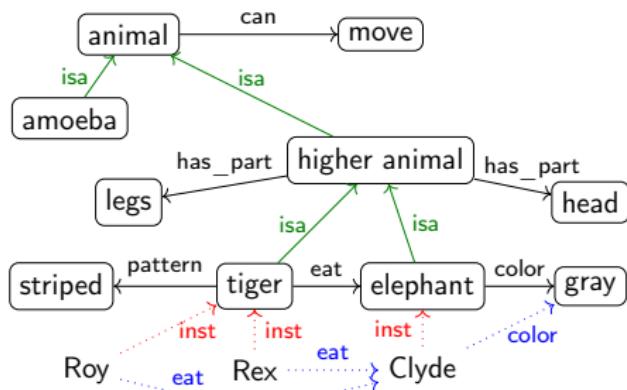
- ▶ **Intuition:** Derived relations represent knowledge that is implicit in the network; they could be added, but usually are not to avoid clutter.
- ▶ **Example 1.8.** Derived relations in 1.5



- ▶ **Slogan:** Get out more knowledge from a semantic networks than you put in.

Terminologies and Assertions

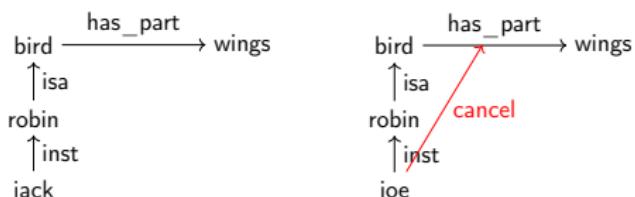
- ▶ **Remark 1.9.** We should distinguish **concepts** from **objects**.
- ▶ **Definition 1.10.** We call the **subgraph** of a **semantic network** N spanned by the **isa** links and **relations** between **concepts** the **terminology** (or **TBox**, or the famous **Isa-Hierarchy**) and the **subgraph** spanned by the **inst** links and **relations** between objects, the **assertions** (or **ABox**) of N .
- ▶ **Example 1.11.** In this network we keep **objects concept** apart notationally:



In particular we have **objects** “Rex”, “Roy”, and “Clyde”, which have (derived) **relations** (e.g. *Clyde* is *gray*).

Limitations of Semantic Networks

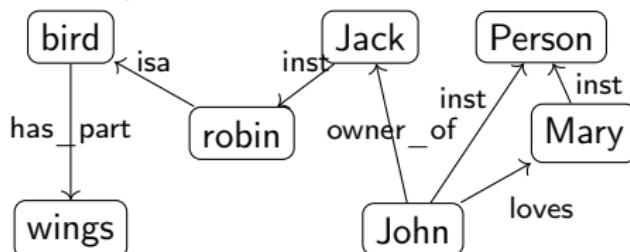
- ▶ What is the meaning of a link?
 - ▶ link labels are very suggestive (misleading for humans)
 - ▶ meaning of link types defined in the process model (no denotational semantics)
- ▶ Problem: No distinction of optional and defining traits!
- ▶ Example 1.12. Consider a robin that has lost its wings in an accident:



“Cancel-links” have been proposed, but their status and process model are debatable.

Another Notation for Semantic Networks

- ▶ **Definition 1.13.** **Function/argument notation** for semantic networks
 - ▶ interprets **nodes** as arguments
 - ▶ interprets **links** as functions
- ▶ **Example 1.14.**

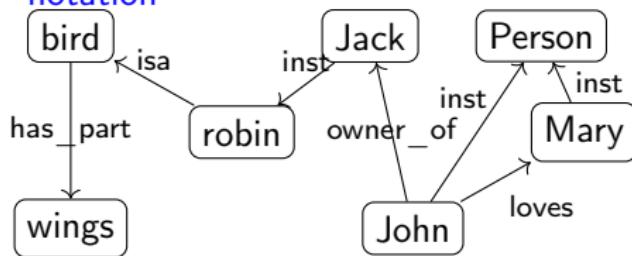


isa(robin,bird)
haspart(bird,wings)
inst(Jack,robin)
owner_of(John, robin)
loves(John,Mary)

- ▶ **Evaluation:**
 - + linear notation (equivalent, but better to implement on a computer)
 - + easy to give process model by deduction (e.g. in ProLog)
 - worse locality properties (networks are associative)

A Denotational Semantics for Semantic Networks

- ▶ **Observation:** If we handle **isa** and **inst** links specially in **function/argument notation**



robin \subseteq bird
haspart(bird,wings)
Jack \in robin
owner_of(John, Jack)
loves(John,Mary)

it looks like **first order logic**, if we take

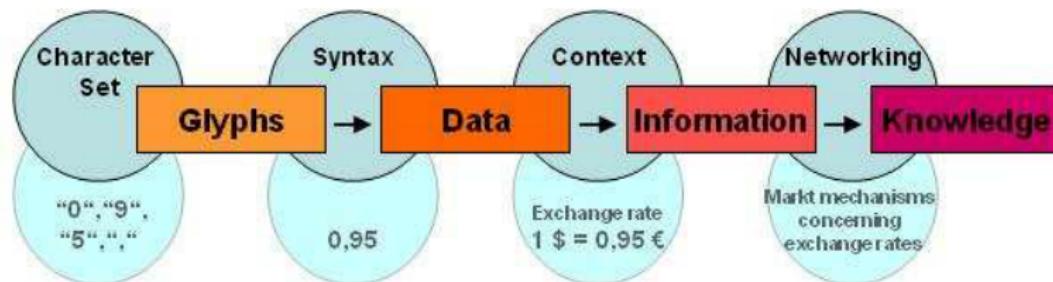
- ▶ $a \in S$ to mean $S(a)$ for an **object** a and a **concept** S .
- ▶ $A \subseteq B$ to mean $\forall X. A(X) \Rightarrow B(X)$ and **concepts** A and B
- ▶ $R(A, B)$ to mean $\forall X. A(X) \Rightarrow (\exists Y. B(Y) \wedge R(X, Y))$ for a **relation** R .

- ▶ **Idea:** Take first-order deduction as process model (gives inheritance for free)

1.3 The Semantic Web

The Semantic Web

- ▶ **Definition 1.15.** The **semantic web** is the result including of semantic content in web pages with the aim of converting the **WWW** into a machine-understandable “web of data”, where **inference**-based services can add value to the ecosystem.
- ▶ **Idea:** Move web content up the ladder, use **inference** to make connections.



- ▶ **Example 1.16.** Information not explicitly represented (in one place)
 - Query: *Who was US president when Barak Obama was born?*
 - Google: ... *BIRTH DATE: August 04, 1961...*
 - Query: *Who was US president in 1961?*
 - Google: *President: Dwight D. Eisenhower [...] John F. Kennedy (starting Jan. 20.)*
- Humans understand the text and combine the information to get the answer.
Machines need more than just text ~ semantic web technology.

What is the Information a User sees?

- ▶ **Example 1.17.** Take the following web-site with a conference announcement
WWW2002

The eleventh International World Wide Web Conference

Sheraton Waikiki Hotel

Honolulu, Hawaii, USA

7-11 May 2002

Registered participants coming from

Australia, Canada, Chile, Denmark, France, Germany, Ghana, Hong Kong, India, Ireland, Italy, Japan, Malta, New Zealand, The Netherlands, Norway, Singapore, Switzerland, the United Kingdom, the United States, Vietnam, Zaire

On the 7th May Honolulu will provide the backdrop of the eleventh International World Wide Web Conference.

Speakers confirmed

Tim Berners-Lee: Tim is the well known inventor of the Web,

Ian Foster: Ian is the pioneer of the Grid, the next generation internet.

What the machine sees

- **Example 1.18.** Here is what the machine “sees” from the conference announcement:

www.EIEE

$\mathcal{T}]]\uparrow\downarrow\subseteq\backslash\sqcup(\mathcal{I}\backslash\sqcup]\nabla\backslash\vdash\sqcup)\backslash\vdash\uparrow\mathcal{W}\nabla\uparrow[\mathcal{W}]$] \mathcal{W}] $\mathcal{C}\backslash\{\nabla\}\nabla\backslash\sqcup]$

S(¬ ⊥ \ W ⊥) ||> ||> H ⊥ ↑

KooomM-TEEE

$\mathcal{R} \} \} \rangle \cup \nabla \lceil \neg \nabla \cup \rangle \} \neg \cup \} \neg \} \{ \nabla \} \}$

$$\mathcal{A} \sqcap (\sqcup \nabla \dashv \vdash) \dashv \Leftrightarrow \mathcal{C} \dashv \vdash \dashv \Leftrightarrow \mathcal{C} \langle \rangle \uparrow \mathcal{D} \sqcap \uparrow \dashv \nabla \Leftrightarrow \mathcal{F} \nabla \dashv \vdash \dashv \Leftrightarrow \mathcal{G} \nabla \uparrow \dashv \vdash \Leftrightarrow \mathcal{G} \dashv \vdash \Leftrightarrow \mathcal{H} \sqcap \vdash \Leftrightarrow \mathcal{K} \sqcap \vdash \Leftrightarrow \mathcal{I} \vdash \Leftrightarrow$$

$\mathcal{IV} \vdash \neg (\neg \mathcal{I} \vdash \neg \mathcal{J}) \vdash \neg \neg \mathcal{M} \vdash \neg \mathcal{L} \vdash \neg \mathcal{N} \vdash \mathcal{Z} \vdash \neg (\neg \mathcal{T} \vdash \mathcal{M} \vdash \neg \mathcal{V} \vdash \neg \mathcal{S} \vdash \neg \mathcal{N} \vdash \neg \mathcal{Z})$

$\mathcal{S} \setminus \{\} \vdash \sqrt{\nabla} \Leftrightarrow \mathcal{S} \setminus \{\} \vdash \nabla \Downarrow \vdash \sqrt{\Leftrightarrow} \vdash (\nabla \setminus \{\}) \vdash \nabla \Downarrow \vdash \mathcal{S} \setminus \{\} \vdash \nabla \Downarrow \Leftrightarrow \mathcal{Z} \vdash \nabla}$

$\mathcal{O}(\log(M+H)\log\log n)$, $\nabla \in \Omega(1/\epsilon)^2 \lceil \nabla \rceil \cdot \log(1/\epsilon) \lceil \nabla \rceil$

$\mathcal{S}_{\sqrt{\cdot}} \vdash \|\nabla f\| \wedge \{\} \nabla \Downarrow \vdash$

$\mathcal{T} \Downarrow \mathcal{B} \models \nabla \backslash \nabla \sqcap \mathcal{L} \models \neg \mathcal{T} \Downarrow \mathcal{B} \models \sqcup \langle \exists \models \models \parallel \parallel \models \backslash \models \backslash \models \parallel \models \nabla \{ \sqcup \langle \mathcal{W} \parallel \models$

I→*\R*]*\U*]*\nabla*→*\I*→*\U*]*\U*[*\I*]*\U*]*\nabla*→{*\U*(*\G**\nabla*)}→*\U*(*\I*]*\U*){*\I*]*\nabla*→*\U*}{*\I*]*\U*]*\nabla*→*\U*

Solution: XML markup with “meaningful” Tags

- ▶ **Example 1.19.** Let's annotate (parts of) the meaning via XML markup

```
<title>WWW2002
The eleventh International World Wide Web Conference</title>
<place> Sheraton Waikiki Hotel Honolulu, Hawaii, USA</place>
<date> 7-11 May 2002</date>
<participants> 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
</participants>
<introduction> On the 7th May Honolulu will provide the backdrop of the eleventh
International World Wide Web Conference.</introduction>
<program>Speakers confirmed
<speaker>Tim Berners-Lee: Tim is the well known inventor of the Web</speaker>
<speaker>Ian Foster: Ian is the pioneer of the Grid, the next generation inter-
net</speaker>
</program>
```

What can we do with this?

- **Example 1.20.** Consider the following fragments:

<title>WWWELLE

$\mathcal{T}(\uparrow\uparrow\sqsubseteq\sqcup\langle\mathcal{I}\sqcup\nabla\setminus\sqcup\rangle\sqcap\sqcup\mathcal{W}\nabla\uparrow\mathcal{W}\sqcap\mathcal{W}\sqcap\mathcal{C}\{\nabla\}\sqcup)$

Given the markup above, a machine agent can

- ▶ parse $\text{May } 7-11 \text{ 2002}$ as the date May 7-11 2002 and add this to the user's calendar,
 - ▶ parse USA as a destination and find flights.
 - ▶ But: do not be deceived by your ability to understand English!

What the machine sees of the XML

- ▶ **Example 1.21.** Here is what the machine sees of the XML

`<title>WWW2002`

The eleventh International World Wide Web Conference`</title>`

`<place> Sheraton Waikiki Hotel Honolulu, Hawaii, USA</place>`

`<date> 7-11 May 2002</date>`

`<participants> 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

`</participants>`

`<introduction> On the 7th May Honolulu will provide the backdrop of the eleventh International World Wide Web Conference.</introduction>`

`<program>Speakers confirmed`

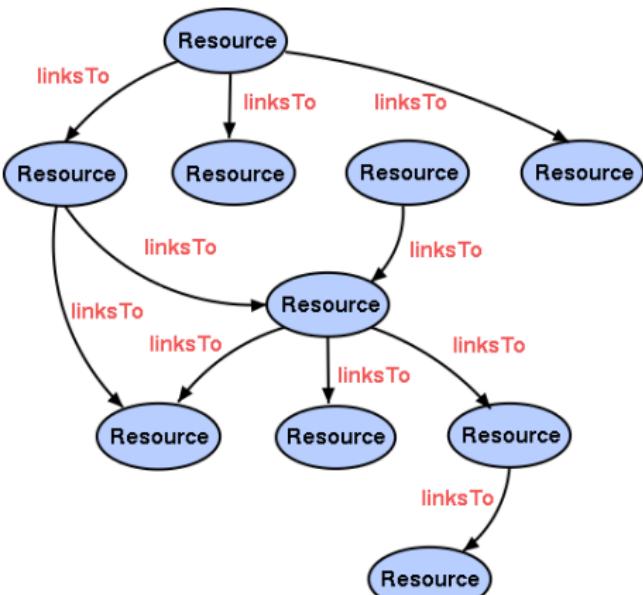
`<speaker>Tim Berners-Lee: Tim is the well known inventor of the Web</speaker>`

`<speaker>Ian Foster: Ian is the pioneer of the Grid, the next generation internet</speaker>`

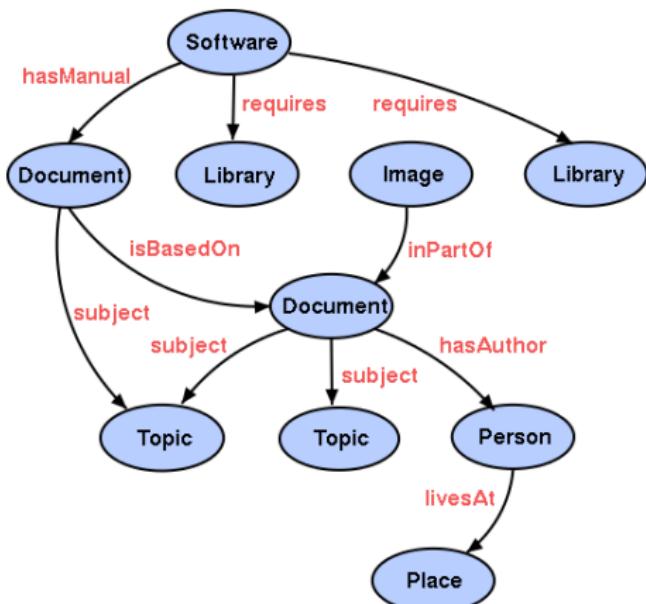
`</program>`

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.



- ▶ **Resources:** Globally identified by [URIs](#) or Locally scoped (Blank), Extensible, Relational
- ▶ **Links:** Identified by [URIs](#), Extensible, Relational
- ▶ **User:** Even more exciting world, richer user experience
- ▶ **Machine:** More processable information is available (Data Web)
- ▶ **Computers and people:** Work, learn and exchange knowledge effectively



Towards a “Machine-Actionable Web”

- ▶ **Recall:** We need external agreement on meaning of annotation tags.
- ▶ **Idea:** standardize them in a community process (e.g. DIN or ISO)
- ▶ **Problem:** Inflexible, Limited number of things can be expressed

Towards a “Machine-Actionable Web”

- ▶ **Recall:** We need external agreement on meaning of annotation tags.
- ▶ **Idea:** standardize them in a community process (e.g. DIN or ISO)
- ▶ **Problem:** Inflexible, Limited number of things can be expressed
- ▶ **Better:** Use ontologies to specify meaning of annotations
 - ▶ Ontologies provide a vocabulary of terms
 - ▶ New terms can be formed by combining existing ones
 - ▶ Meaning (semantics) of such terms is formally specified
 - ▶ Can also specify relationships between terms in multiple ontologies

Towards a “Machine-Actionable Web”

- ▶ **Recall:** We need external agreement on meaning of annotation tags.
- ▶ **Idea:** standardize them in a community process (e.g. DIN or ISO)
- ▶ **Problem:** Inflexible, Limited number of things can be expressed
- ▶ **Better:** Use ontologies to specify meaning of annotations
 - ▶ Ontologies provide a vocabulary of terms
 - ▶ New terms can be formed by combining existing ones
 - ▶ Meaning (semantics) of such terms is formally specified
 - ▶ Can also specify relationships between terms in multiple ontologies
- ▶ Inference with annotations and ontologies (get out more than you put in!)
- ▶ Standardize annotations in **RDF** [KC04] or **RDFa** [Her+13] and ontologies on **OWL** [OWL09]
- ▶ Harvest **RDF** and **RDFa** in to a triplestore or **OWL** reasoner.
- ▶ Query that for implied knowledge (e.g. chaining multiple facts from Wikipedia)
SPARQL: Who was US President when Barack Obama was Born?
DBpedia: John F. Kennedy (was president in August 1961)

1.4 Other Knowledge Representation Approaches

Frame Notation as Logic with Locality

► Predicate Logic:

(where is the locality?)

catch_22 ∈ *catch_object*

There is an instance of catching

*catcher(**catch_22*,*jack_2*)

Jack did the catching

*caught(**catch_22*,*ball_5*)

He caught a certain ball

► Definition 1.22. Frames

(group everything around the object)

(*catch_object* *catch_22*

 (*catcher* *jack_2*)

 (*caught* *ball_5*))

+ Once you have decided on a frame, all the information is local

+ easy to define schemes for concepts (aka. types in feature structures)

- how to determine frame, when to choose frame (log/chair)

KR involving Time (Scripts [Shank '77])

- ▶ **Idea:** Organize typical event sequences, actors and props into representation.
- ▶ **Definition 1.23.** A **script** is a structured representation describing a stereotyped sequence of events in a particular context. Structurally, **scripts** are very much like **frames**, except the values that fill the slots must be ordered.
- ▶ **Example 1.24.** getting your hair cut (at a beauty parlor)
 - ▶ props, actors as “script variables”
 - ▶ events in a (generalized) sequence
- ▶ use **script** material for
 - ▶ anaphora, bridging references
 - ▶ default common ground
 - ▶ to fill in missing material into situations



Other Representation Formats (not covered)

- ▶ Procedural Representations (production systems)
- ▶ Analogical representations (interesting but not here)
- ▶ Iconic representations (interesting but very difficult to formalize)
- ▶ If you are interested, come see me off-line

2 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 log. calculi (combinatorial explosion)
- ▶ One possible answer: description logics. (next couple of times)

2.1 Propositional Logic as a Set Description Language

Propositional Logic as Set Description Language

- Idea: Use propositional logic as a set description language: (variant syntax/semantics)
- **Definition 2.1.** Let PL_{DL}^0 be given by the following grammar for the PL_{DL}^0 concepts. (formulae)

$$\mathcal{L} ::= C \mid T \mid \perp \mid \overline{\mathcal{L}} \mid \mathcal{L} \sqcap \mathcal{L} \mid \mathcal{L} \sqcup \mathcal{L} \mid \mathcal{L} \sqsubseteq \mathcal{L} \mid \mathcal{L} \equiv \mathcal{L}$$

i.e. PL_{DL}^0 formed from

- atomic concept (\cong propositional variables)
- concept intersection (\sqcap) (\cong conjunction \wedge)
- concept complement ($\overline{\cdot}$) (\cong negation \neg)
- concept union (\sqcup), subsumption (\sqsubseteq), and equality (\equiv) defined from these. ($\cong \vee, \Rightarrow, \text{and } \Leftrightarrow$)

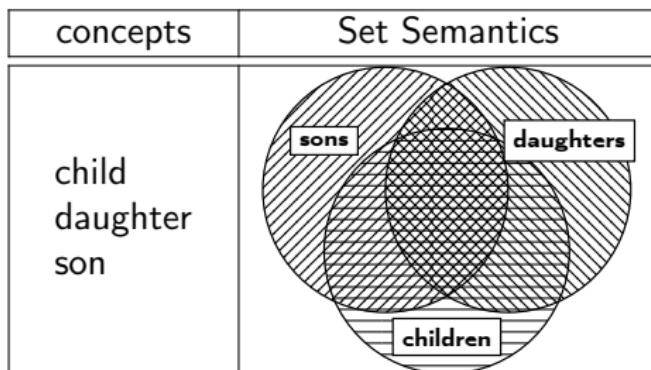
► **Definition 2.2 (Formal Semantics).**

Let \mathcal{D} be a given set (called the domain) and $\varphi: \mathcal{V}_o \rightarrow \mathcal{P}(\mathcal{D})$, then we define

- $\llbracket P \rrbracket := \varphi(P)$, (remember $\varphi(P) \subseteq \mathcal{D}$).
- $\llbracket A \sqcap B \rrbracket := \llbracket A \rrbracket \cap \llbracket B \rrbracket$ and $\llbracket \overline{A} \rrbracket := \mathcal{D} \setminus \llbracket A \rrbracket \dots$
- Note: $\langle \text{PL}_{\text{DL}}^0, \mathcal{S}, \llbracket \cdot \rrbracket \rangle$, where \mathcal{S} is the class of possible domains forms a logical system.

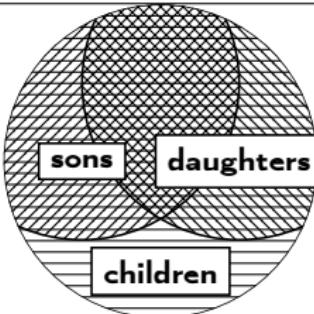
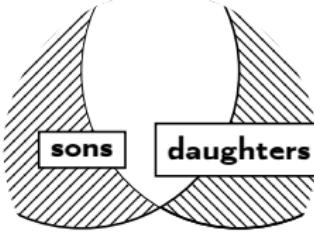
Concept Axioms

- ▶ **Observation:** Set-theoretic semantics of ‘true’ and ‘false’ ($T := \varphi \sqcup \bar{\varphi}$ $\perp := \varphi \sqcap \bar{\varphi}$)
 $\llbracket T \rrbracket = \llbracket p \rrbracket \cup \llbracket \bar{p} \rrbracket = \llbracket p \rrbracket \cup (\mathcal{D} \setminus \llbracket p \rrbracket) = \mathcal{D}$ Analogously: $\llbracket \perp \rrbracket = \emptyset$
- ▶ **Idea:** Use logical axioms to describe the world (Axioms restrict the class of admissible domain structures)
- ▶ **Definition 2.3.** A **concept axiom** is a PL_{DL}^0 formula A that is assumed to be true in the world.
- ▶ **Definition 2.4 (Set-Theoretic Semantics of Axioms).** A is **true** in domain \mathcal{D} iff $\llbracket A \rrbracket = \mathcal{D}$.
- ▶ **Example 2.5.** A world with three concepts and no concept axioms



Effects of Axioms to Siblings

- **Example 2.6.** We can use **concept axioms** to describe the world from 2.5.

Axioms	Semantics
$\text{son} \sqsubseteq \text{child}$ iff $\llbracket \text{son} \rrbracket \cup \llbracket \text{child} \rrbracket = \mathcal{D}$ iff $\llbracket \text{son} \rrbracket \subseteq \llbracket \text{child} \rrbracket$ $\text{daughter} \sqsubseteq \text{child}$ iff $\llbracket \text{daughter} \rrbracket \cup \llbracket \text{child} \rrbracket = \mathcal{D}$ iff $\llbracket \text{daughter} \rrbracket \subseteq \llbracket \text{child} \rrbracket$	
$\text{son} \sqcap \text{daughter}$ $\text{child} \sqsubseteq \text{son} \sqcup \text{daughter}$	

Propositional Identities

Name	for \sqcap	for \sqcup
Idenpot.	$\varphi \sqcap \varphi = \varphi$	$\varphi \sqcup \varphi = \varphi$
Identity	$\varphi \sqcap \top = \varphi$	$\varphi \sqcup \perp = \varphi$
Absorpt.	$\varphi \sqcup \top = \top$	$\varphi \sqcap \perp = \perp$
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	$\frac{\varphi \sqcap \psi}{\varphi} = \overline{\varphi} \sqcup \overline{\psi}$	$\frac{\varphi \sqcup \psi}{\varphi} = \overline{\varphi} \sqcap \overline{\psi}$
dneg		$\overline{\overline{\varphi}} = \varphi$

► **Definition 2.7.** Translation into PL¹

(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
$\bar{p}^{fo(x)} := p(x)$	
$\overline{\bar{A}}^{fo(x)} := \neg A^{fo(x)}$	
$(\bar{A} \sqcap \bar{B})^{fo(x)} := \bar{A}^{fo(x)} \wedge \bar{B}^{fo(x)}$	\wedge vs. \sqcap
$(\bar{A} \sqcup \bar{B})^{fo(x)} := \bar{A}^{fo(x)} \vee \bar{B}^{fo(x)}$	\vee vs. \sqcup
$(\bar{A} \sqsubseteq \bar{B})^{fo(x)} := \bar{A}^{fo(x)} \Rightarrow \bar{B}^{fo(x)}$	\Rightarrow vs. \sqsubseteq
$\bar{A} = \bar{B}^{fo(x)} := \bar{A}^{fo(x)} \Leftrightarrow \bar{B}^{fo(x)}$	\Leftrightarrow vs. $=$
$\bar{A}^{fo} := (\forall x. \bar{A}^{fo(x)})$	for formulae

- **Example 2.8.** We translate the concept axioms from 2.6 to fortify our intuition:

$$\begin{aligned}\overline{(\text{son} \sqsubseteq \text{child})}^{fo} &= \forall x.\text{son}(x) \Rightarrow \text{child}(x) \\ \overline{(\text{daughter} \sqsubseteq \text{child})}^{fo} &= \forall x.\text{daughter}(x) \Rightarrow \text{child}(x) \\ \overline{(\text{son} \sqcap \text{daughter})}^{fo} &= \forall x.\overline{\text{son}(x) \wedge \text{daughter}(x)} \\ \overline{(\text{child} \sqsubseteq \text{son} \sqcup \text{daughter})}^{fo} &= \forall x.\text{child}(x) \Rightarrow \text{son}(x) \vee \text{daughter}(x)\end{aligned}$$

- 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?)

2.2 Ontologies and Description Logics

Ontologies aka. "World Descriptions"

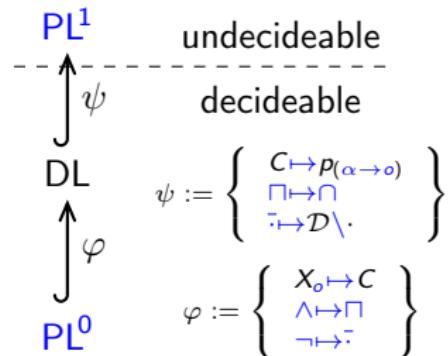
- ▶ **Definition 2.9 (Classical).** An **ontology** is a representation of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse.
- ▶ **Remark:** 2.9 is very general, and depends on what we mean by “representation”, “entities”, “types”, and “interrelationships”.
This may be a feature, and not a bug, since we can use the same intuitions across a variety of representations.
- ▶ **Definition 2.10 (Technical).** An **ontology** consists of a **logical systems** $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ and **concept axioms** (expressed in \mathcal{L}) about
 - ▶ **individuals:** concrete instances of objects in the domain,
 - ▶ **concepts:** classes of individuals that share properties and aspects, and
 - ▶ **relations:** ways in which classes and individuals can be related to one another.
- ▶ **Example 2.11.** Semantic networks are **ontologies**. (relatively informal)
- ▶ **Example 2.12.** PL_{DL}^0 is an ontology format. (formal, but relatively weak)
- ▶ **Example 2.13.** PL^1 is an ontology format as well. (formal, expressive)

The Description Logic Paradigm

- Idea: Build a whole family of logics for describing sets and their relations. (tailor their expressivity and computational properties)
- Definition 2.14. A **description logic** is a formal system for talking about sets 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 , \neg , \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 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
 - **sassertions** (or **ABox**): a set of **individuals** and statements about concept membership and role relationships for them.

- ▶ Let \mathcal{D} be a **description logic** with concepts \mathcal{C} .
- ▶ **Definition 2.16.** A **concept definition** is a pair $c = C$, where c is a new concept name and $C \in \mathcal{C}$ is a \mathcal{D} -formula.
- ▶ **Definition 2.17.** A concept definition $c = C$ is called **recursive**, iff c occurs in C .
- ▶ **Example 2.18.** We can define $\text{mother} = \text{woman} \sqcap \text{has_child}$.
- ▶ **Definition 2.19.** An **TBox** is a finite set of concept definitions and concept axioms. It is called **acyclic**, iff it does not contain recursive definitions.
- ▶ **Definition 2.20.** A formula A is called **normalized** wrt. an **TBox** \mathcal{T} , iff it does not contain concept names defined in \mathcal{T} . (convenient)
- ▶ **Definition 2.21 (Algorithm).** (for arbitrary DLs)
Input: A formula A and a **TBox** \mathcal{T} .
 - ▶ **While** [A contains concept name c and \mathcal{T} a concept definition $c = C$]
 - ▶ substitute c by C in A .
- ▶ **Lemma 2.22.** This algorithm terminates for acyclic **TBoxes**, but results can be exponentially large.

Kinds of Inference in Description Logics

- ▶ 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

► Example 2.23 (T-Box).

man	=	person \sqcap has _ Y	person with y-chromosome
woman	=	person \sqcap has <u>_ Y</u>	person without y-chromosome
hermaphrodite	=	man \sqcap woman	man and woman

- This specification is inconsistent, i.e. $[\text{hermaphrodite}] = \emptyset$ for all \mathcal{D}, φ .
- Algorithm: Propositional satisfiability test (NP complete)
we know how to do this, e.g. tableau, resolution.

Subsumption Test

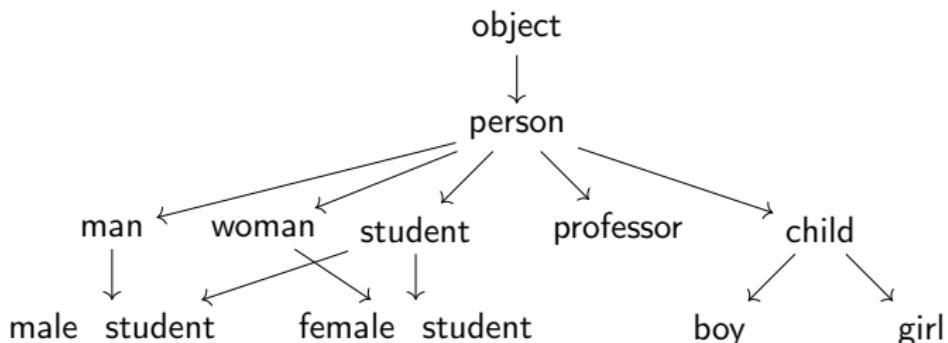
- ▶ **Example 2.24.** in this case trivial

Axioms	entailed subsumption relation
$\text{man} = \text{person} \sqcap \text{has_Y}$	$\text{man} \sqsubseteq \text{person}$
$\text{woman} = \text{person} \sqcap \text{has_Y}$	$\text{woman} \sqsubseteq \text{person}$

- ▶ Reduction to consistency test: (need to implement only one) $\text{Axioms} \Rightarrow A \Rightarrow B$ is valid iff $\text{Axioms} \wedge A \wedge \neg B$ is inconsistent.
- ▶ **Definition 2.25.** A **subsumes** B (modulo an axiom set \mathcal{A})
iff $\llbracket B \rrbracket \subseteq \llbracket A \rrbracket$ for all interpretations \mathcal{D} , that satisfy \mathcal{A}
iff $\mathcal{A} \Rightarrow B \Rightarrow A$ is valid
- ▶ In our example: person subsumes woman and man

Classification

- ▶ The subsumption relation among all concepts (subsumption graph)
- ▶ Visualization of the subsumption graph for inspection (plausibility)
- ▶ **Definition 2.26.** Classification is the computation of the subsumption graph.
- ▶ **Example 2.27.** (not always so trivial)



3 A simple Description Logic: ALC

3.1 Basic ALC: Concepts, Roles, and Quantification

Motivation for ALC (Prototype Description Logic)

- ▶ Propositional logic (PL^0) is not expressive enough
- ▶ **Example 3.1.** “mothers are women that have a child”
- ▶ **Reason:** there are no quantifiers in PL^0 (existential (\exists) and universal (\forall))
- ▶ **Idea:** Use first-order predicate logic (PL^1)

$$\forall x.\text{mother}(x) \Leftrightarrow (\text{woman}(x) \wedge \exists y.\text{has_child}(x, y))$$

- ▶ **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*.

- ▶ **Definition 3.2 (Concepts).** (aka. “predicates” in PL¹ or “propositional variables” in PL_{DL}⁰) concepts in DLs name classes of objects like in OOP.
- ▶ **Definition 3.3 (Special concepts).** The **top concept** \top (for “true” or “all”) and the **bottom concept** \perp (for “false” or “none”).
- ▶ **Example 3.4.** person, woman, man, mother, professor, student, car, BMW, computer, computer program, heart attack risk, furniture, table, leg of a chair, ...
- ▶ **Definition 3.5. Roles** name binary relations (like in PL¹)
- ▶ **Example 3.6.** has_child, has_son, has_daughter, loves, hates, gives_course, executes_computer_program, has_leg_of_table, has_wheel, has_motor, ...

► **Definition 3.7 (Grammar).**

$$F_{\mathcal{AC}} ::= C \mid T \mid \perp \mid \overline{F_{\mathcal{AC}}} \mid F_{\mathcal{AC}} \sqcap F_{\mathcal{AC}} \mid F_{\mathcal{AC}} \sqcup F_{\mathcal{AC}} \mid \exists R.F_{\mathcal{AC}} \mid \forall R.F_{\mathcal{AC}}$$

► Example 3.8.

- ▶ $\text{person} \sqcap \exists \text{has_child}.\text{student}$ (parents of students) (The set of persons that have a child which is a student)
 - ▶ $\text{person} \sqcap \exists \text{has_child.} \exists \text{has_child}.\text{student}$ (grandparents of students)
 - ▶ $\text{person} \sqcap \exists \text{has_child.} \exists \text{has_child.}(\text{student} \sqcup \text{teacher})$ (grandparents of students or teachers)
 - ▶ $\text{person} \sqcap \forall \text{has_child}.\text{student}$ (parents whose children are all students)
 - ▶ $\text{person} \sqcap \forall \text{haschild.} \exists \text{has_child}.\text{student}$ (grandparents, whose children all have at least one child that is a student)

- ▶ **Example 3.9.** $\text{car} \sqcap \exists \text{has_part}. \exists \text{made_in}.\overline{\text{EU}}$ (cars that have at least one part that has not been made in the EU)
- ▶ **Example 3.10.** $\text{student} \sqcap \forall \text{audits_course}. \text{graduatelevelcourse}$ (students, that only audit graduate level courses)
- ▶ **Example 3.11.** $\text{house} \sqcap \forall \text{has_parking}. \text{off_street}$ (houses with off-street parking)
- ▶ Note: $p \sqsubseteq q$ can still be used as an abbreviation for $\overline{p} \sqcup q$.
- ▶ **Example 3.12.** $\text{student} \sqcap \forall \text{audits_course}. (\exists \text{hasTutorial}. T \sqsubseteq \forall \text{has_TA}. \text{woman})$ (students that only audit courses that either have no tutorial or tutorials that are TAed by women)

- ▶ **Idea:** Define new concepts from known ones.
- ▶ **Definition 3.13.** A **concept definition** is a pair consisting of a new concept name (the **definiendum**) and an *ALC* formula (the **definiens**). Concept names are not **definienda** are called **primitive**.
- ▶ We extend the *ALC* grammar from 3.7 by the production $CD_{ALC} := C = F_{ALC}$.
- ▶ **Example 3.14.**

Definition	rec?
$\text{man} = \text{person} \sqcap \exists \text{has_chrom}. Y _ \text{chrom}$	-
$\text{woman} = \text{person} \sqcap \forall \text{has_chrom}. Y _ \text{chrom}$	-
$\text{mother} = \text{woman} \sqcap \exists \text{has_child}. \text{person}$	-
$\text{father} = \text{man} \sqcap \exists \text{has_child}. \text{person}$	-
$\text{grandparent} = \text{person} \sqcap \exists \text{has_child}. (\text{mother} \sqcup \text{father})$	-
$\text{german} = \text{person} \sqcap \exists \text{has_parents}. \text{german}$	+
$\text{number_list} = \text{empty_list} \sqcup \exists \text{is_first}. \text{number} \sqcap \exists \text{is_rest}. \text{number_list}$	+

- ▶ **Definition 3.15.** We call an \mathcal{ALC} formula φ **normalized** wrt. a set of **concept definitions**, iff all concept names occurring in φ are **primitive**.
- ▶ **Definition 3.16.** Given a set \mathcal{D} of **concept definitions**, **normalization** is the process of replacing in an \mathcal{ALC} formula φ all occurrences of **definienda** in \mathcal{D} with their **definientia**.
- ▶ **Example 3.17 (Normalizing grandparent).**

grandparent

$\mapsto \text{person} \sqcap \exists \text{has_child}.(\text{mother} \sqcup \text{father})$

$\mapsto \text{person} \sqcap \exists \text{has_child}.(\text{woman} \sqcap \exists \text{has_child}. \text{person} \sqcap \text{man} \sqcap \exists \text{has_child}. \text{person})$

$\mapsto \text{person} \sqcap \exists \text{has_child}.(\text{person} \sqcap \exists \text{has_chrom}. \text{Y_chrom} \sqcap \exists \text{has_child}. \text{person} \sqcap \text{person} \sqcap \exists \text{has_chrom}. \text{Y_chrom} \sqcap \exists \text{has_child}. \text{person})$

- ▶ **Observation 3.18.** Normalization results can be exponential. (contain redundancies)
- ▶ **Observation 3.19.** Normalization need not terminate on cyclic TBoxes.
- ▶ **Example 3.20.**

german $\mapsto \text{person} \sqcap \exists \text{has_parents}. \text{german}$

$\mapsto \text{person} \sqcap \exists \text{has_parents}. (\text{person} \sqcap \exists \text{has_parents}. \text{german})$

$\mapsto \dots$

Semantics of \mathcal{ALC}

- \mathcal{ALC} semantics is an extension of the set-semantics of propositional logic.
- **Definition 3.21.** A **model** for \mathcal{ALC} is a pair $\langle \mathcal{D}, [[\cdot]] \rangle$, where \mathcal{D} is a non-empty set called the **domain** and $[[\cdot]]$ a mapping called the **interpretation**, such that

Op.	formula semantics
	$[c] \subseteq \mathcal{D} = [\top] \quad [\perp] = \emptyset \quad [r] \subseteq \mathcal{D} \times \mathcal{D}$
\neg	$[\neg \varphi] = [\varphi] = \mathcal{D} \setminus [\varphi]$
\sqcap	$[(\varphi \sqcap \psi)] = [\varphi] \cap [\psi]$
\sqcup	$[(\varphi \sqcup \psi)] = [\varphi] \cup [\psi]$
$\exists R.$	$[(\exists R. \varphi)] = \{x \in \mathcal{D} \mid \exists y. \langle x, y \rangle \in [R] \text{ and } y \in [\varphi]\}$
$\forall R.$	$[(\forall R. \varphi)] = \{x \in \mathcal{D} \mid \forall y. \text{if } \langle x, y \rangle \in [R] \text{ then } y \in [\varphi]\}$

- Alternatively we can define the semantics of \mathcal{ALC} by translation into PL^1 .
- **Definition 3.22.** The translation of \mathcal{ALC} into PL^1 extends the one from 2.7 by the following **quantifier rules**:

$$\overline{(\forall R. \varphi)}^{fo(x)} := (\forall y. R(x, y) \Rightarrow \overline{\varphi}^{fo(y)}) \quad \overline{(\exists R. \varphi)}^{fo(x)} := (\exists y. R(x, y) \wedge \overline{\varphi}^{fo(y)})$$

- **Observation 3.23.** The set-theoretic semantics from 3.21 and the “semantics-by-translation” from 3.22 induce the same notion of satisfiability.

▶	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; padding: 2px;">1</td><td style="width: 33%; padding: 2px;">$\exists R.\varphi = \forall R.\overline{\varphi}$</td><td style="width: 33%; padding: 2px;">3</td><td style="width: 33%; padding: 2px;">$\forall R.\varphi = \exists R.\overline{\varphi}$</td></tr> <tr> <td style="padding: 2px;">2</td><td style="padding: 2px;">$\forall R.(\varphi \sqcap \psi) = \forall R.\varphi \sqcap \forall R.\psi$</td><td style="padding: 2px;">4</td><td style="padding: 2px;">$\exists R.(\varphi \sqcup \psi) = \exists R.\varphi \sqcup \exists R.\psi$</td></tr> </table>	1	$\exists R.\varphi = \forall R.\overline{\varphi}$	3	$\forall R.\varphi = \exists R.\overline{\varphi}$	2	$\forall R.(\varphi \sqcap \psi) = \forall R.\varphi \sqcap \forall R.\psi$	4	$\exists R.(\varphi \sqcup \psi) = \exists R.\varphi \sqcup \exists R.\psi$
1	$\exists R.\varphi = \forall R.\overline{\varphi}$	3	$\forall R.\varphi = \exists R.\overline{\varphi}$						
2	$\forall R.(\varphi \sqcap \psi) = \forall R.\varphi \sqcap \forall R.\psi$	4	$\exists R.(\varphi \sqcup \psi) = \exists R.\varphi \sqcup \exists R.\psi$						

▶ Proof of 1

$$\begin{aligned}
 [[\exists R.\varphi]] &= \mathcal{D} \setminus [[(\exists R.\varphi)]] = \mathcal{D} \setminus \{x \in \mathcal{D} \mid \exists y. (\langle x, y \rangle \in [[R]]) \text{ and } (y \in [[\varphi]])\} \\
 &= \{x \in \mathcal{D} \mid \text{not } \exists y. (\langle x, y \rangle \in [[R]]) \text{ and } (y \in [[\varphi]])\} \\
 &= \{x \in \mathcal{D} \mid \forall y. \text{if } (\langle x, y \rangle \in [[R]]) \text{ then } (y \notin [[\varphi]])\} \\
 &= \{x \in \mathcal{D} \mid \forall y. \text{if } (\langle x, y \rangle \in [[R]]) \text{ then } (y \in \mathcal{D} \setminus [[\varphi]])\} \\
 &= \{x \in \mathcal{D} \mid \forall y. \text{if } (\langle x, y \rangle \in [[R]]) \text{ then } (y \in [[\overline{\varphi}]])\} \\
 &= [[(\forall R.\overline{\varphi})]]
 \end{aligned}$$

Negation Normal Form

- ▶ **Definition 3.24 (NNF).** An \mathcal{ALC} formula is in **negation normal form (NNF)**, iff \neg is only applied to concept names.
- ▶ Use the \mathcal{ALC} identities as rules to compute it. (in linear time)

example	by rule
$\exists R.(\forall S.e \sqcap \forall S.d)$ $\mapsto \forall R.\forall S.e \sqcap \overline{\forall S.d}$ $\mapsto \forall R.(\overline{\forall S.e} \sqcup \overline{\forall S.d})$ $\mapsto \forall R.(\exists S.\overline{e} \sqcup \overline{\forall S.d})$ $\mapsto \forall R.(\exists S.\overline{e} \sqcup \forall S.d)$	$\exists \overline{R}.\varphi \mapsto \forall R.\overline{\varphi}$ $\overline{\varphi \sqcap \psi} \mapsto \overline{\varphi} \sqcup \overline{\psi}$ $\overline{\forall R.\varphi} \mapsto \exists R.\overline{\varphi}$ $\overline{\overline{\varphi}} \mapsto \varphi$

- **Definition 3.25.** We define the **assertions** for \mathcal{ALC}

- $a:\varphi$ (a is a φ)
- $a R b$ (a stands in relation R to b)

assertions make up the **ABox** in \mathcal{ALC} .

- **Definition 3.26.** Let $\langle \mathcal{D}, [[\cdot]] \rangle$ be a **model** for \mathcal{ALC} , then we define

- $[(a:\varphi)] = \text{T}$, iff $[a] \in [\varphi]$, and
- $[(a R b)] = \text{T}$, iff $([a], [b]) \in [R]$.

- **Definition 3.27.** We extend the PL^1 translation of \mathcal{ALC} to \mathcal{ALC} assertions:

- $\overline{(a:\varphi)}^{fo} := \overline{\varphi}^{fo(a)}$, and
- $\overline{a R b}^{fo} := R(a,b)$.

3.2 Inference for ALC

\mathcal{T}_{ALC} : A Tableau-Calculus for ALC

- ▶ **Recap Tableaux:** A tableau calculus develops an initial tableau in a tree-formed scheme using tableau extension rules.
A **saturated** tableau (no rules applicable) constitutes a refutation, if all branches are **closed** (end in \perp).
▶ **Definition 3.28.** The ALC **tableau calculus** \mathcal{T}_{ALC} acts on **assertions**

- ▶ $x:\varphi$ (x inhabits concept φ)
- ▶ $x R y$ (x and y are in relation R)

where φ is a **normalized ALC concept** in **negation normal form** with the following rules:

$$\frac{x:c}{\frac{x:\overline{c}}{\perp} \mathcal{T}_\perp} \quad \frac{x:\varphi \sqcap \psi}{\frac{x:\varphi}{x:\psi} \mathcal{T}_\sqcap} \quad \frac{x:\varphi \sqcup \psi}{\frac{x:\varphi \quad x:\psi}{x:\psi} \mathcal{T}_\sqcup} \quad \frac{x:\forall R.y}{\frac{x R y}{y:\varphi} \mathcal{T}_\forall} \quad \frac{x:\exists R.y}{\frac{x R y}{y:\varphi} \mathcal{T}_\exists}$$

- ▶ To test consistency of a concept φ , normalize φ to ψ , initialize the tableau with $x:\psi$, saturate. Open branches \leadsto consistent.
(x arbitrary)

- ▶ **Example 3.29.** We have two similar conjectures about children.
 - ▶ $x:\forall \text{has_child}.\text{man} \sqcap \exists \text{has_child}.\overline{\text{man}}$ (all sons, but a daughter)
 - ▶ $x:\forall \text{has_child}.\text{man} \sqcap \exists \text{has_child}.\text{man}$ (only sons, and at least one)

- ▶ **Example 3.29.** We have two similar conjectures about children.
 - ▶ $x:\forall \text{has_child}.\text{man} \sqcap \exists \text{has_child}.\overline{\text{man}}$ (all sons, but a daughter)
 - ▶ $x:\forall \text{has_child}.\text{man} \sqcap \exists \text{has_child}.\text{man}$ (only sons, and at least one)
- ▶ **Example 3.30 (Tableau Proof).**

1	$x:\forall \text{has_child}.\text{man} \sqcap \exists \text{has_child}.\overline{\text{man}}$	initial	$x:\forall \text{has_child}.\text{man} \sqcap \exists \text{has_child}.\overline{\text{man}}$
2	$x:\forall \text{has_child}.\text{man}$	\mathcal{T}_{\sqcap}	$x:\forall \text{has_child}.\text{man}$
3	$x:\exists \text{has_child}.\overline{\text{man}}$	\mathcal{T}_{\sqcap}	$x:\exists \text{has_child}.\overline{\text{man}}$
4	$x \text{ has_child } y$	\mathcal{T}_{\exists}	$x \text{ has_child } y$
5	$y:\overline{\text{man}}$	\mathcal{T}_{\exists}	$y:\overline{\text{man}}$
6	$y:\text{man}$	\mathcal{T}_{\forall}	open
7	\perp	\mathcal{T}_{\perp}	
inconsistent			

The right tableau has a model: there are two persons, x and y . y is the only child of x , y is a man

Another \mathcal{TAC} Example

- ▶ **Example 3.31.** $\forall \text{has_child}.\text{(ugrad} \sqcup \text{grad}) \sqcap \exists \text{has_child}.\overline{\text{ugrad}}$ is satisfiable.
 - ▶ Let's try it on the board

Another \mathcal{TAC} Example

► **Example 3.31.** $\forall \text{has_child.}(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has_child.} \overline{\text{ugrad}}$ is satisfiable.

- Let's try it on the board
- Tableau proof for the notes

1	$x: \forall \text{has_child.}(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has_child.} \overline{\text{ugrad}}$	
2	$x: \forall \text{has_child.}(\text{ugrad} \sqcup \text{grad})$	\mathcal{T}_{\forall}
3	$x: \exists \text{has_child.} \overline{\text{ugrad}}$	\mathcal{T}_{\exists}
4	$x \text{ has_child } y$	\mathcal{T}_{\exists}
5	$y: \overline{\text{ugrad}}$	\mathcal{T}_{\exists}
6	$y: \text{ugrad} \sqcup \text{grad}$	\mathcal{T}_{\forall}
7	$y: \text{ugrad}$	\mathcal{T}_{\sqcup}
8	\perp	open

The left branch is closed, the right one represents a model: y is a child of x , y is a graduate student, x has exactly one child: y .

- ▶ We study the following properties of a tableau calculus \mathcal{C} :
 - ▶ **Termination:** there are no infinite sequences of rule applications.
 - ▶ **Correctness:** 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 the satisfiability itself **(as a benchmark)**

► **Lemma 3.32.** If φ satisfiable, then T_{AC} terminates on $x:\varphi$ with open branch.

► **Proof:** Let $\mathcal{M}:=\langle \mathcal{D}, [\cdot] \rangle$ be a model for φ and $w \in [\varphi]$.

$$\mathcal{I} \models (x:\psi) \quad \text{iff} \quad [x] \in [\psi]$$

1. We define $[x]:=w$ and $\mathcal{I} \models x R y \quad \text{iff} \quad \langle x,y \rangle \in [R]$

$$\mathcal{I} \models S \quad \text{iff} \quad \mathcal{I} \models c \text{ for all } c \in S$$

2. This gives us $\mathcal{M} \models (x:\varphi)$

(base case)

3. (Case analysis) If the branch is satisfiable, then either

► no rule applicable to leaf,

(open branch)

► or rule applicable and one new branch satisfiable.

(inductive case)

4. (Consequence) There must be an open branch.

(by termination)



Case analysis on the rules

\mathcal{T}_\sqcap applies then $\mathcal{I} \models (x:\varphi \sqcap \psi)$, i.e. $\llbracket x \rrbracket \in \llbracket (\varphi \sqcap \psi) \rrbracket$

so $\llbracket x \rrbracket \in \llbracket \varphi \rrbracket$ and $\llbracket x \rrbracket \in \llbracket \psi \rrbracket$, thus $\mathcal{I} \models (x:\varphi)$ and $\mathcal{I} \models (x:\psi)$.

\mathcal{T}_\sqcup applies then $\mathcal{I} \models (x:\varphi \sqcup \psi)$, i.e. $\llbracket x \rrbracket \in \llbracket (\varphi \sqcup \psi) \rrbracket$

so $\llbracket x \rrbracket \in \llbracket \varphi \rrbracket$ or $\llbracket x \rrbracket \in \llbracket \psi \rrbracket$, thus $\mathcal{I} \models (x:\varphi)$ or $\mathcal{I} \models (x:\psi)$,
wlog. $\mathcal{I} \models (x:\varphi)$.

\mathcal{T}_\forall applies then $\mathcal{I} \models (x:\forall R.\varphi)$ and $\mathcal{I} \models x R y$, i.e. $\llbracket x \rrbracket \in \llbracket (\forall R.\varphi) \rrbracket$ and $\langle x,y \rangle \in \llbracket R \rrbracket$,

so $\llbracket y \rrbracket \in \llbracket \varphi \rrbracket$

\mathcal{T}_\exists applies then $\mathcal{I} \models (x:\exists R.\varphi)$, i.e. $\llbracket x \rrbracket \in \llbracket (\exists R.\varphi) \rrbracket$,

so there is a $v \in D$ with $\langle \llbracket x \rrbracket, v \rangle \in \llbracket R \rrbracket$ and $v \in \llbracket \varphi \rrbracket$.

We define $\llbracket y \rrbracket := v$, then $\mathcal{I} \models x R y$ and $\mathcal{I} \models (y:\varphi)$

Completeness of the Tableau Calculus

- ▶ **Lemma 3.33.** Open saturated tableau branches for φ induce models for φ .
- ▶ **Proof:** construct a model for the branch and verify for φ
 1. (**Model Construction**) Let \mathcal{B} be an open saturated branch
 - ▶ we define

$$\mathcal{D} := \{x \mid (x:\psi) \in \mathcal{B} \text{ or } z R x \in \mathcal{B}\}$$

$$[c] := \{x \mid (x:c) \in \mathcal{B}\}$$

$$[R] := \{(x,y) \mid x R y \in \mathcal{B}\}$$

- ▶ well-defined since never $(x:c), (x:\bar{c}) \in \mathcal{B}$ (otherwise \mathcal{T}_\perp applies)
 - ▶ \mathcal{M} satisfies all constraints $x:c$, $x:\bar{c}$ and $x R y$, (by construction)
2. (**Induction**) $\mathcal{M} \models (y:\psi)$, for all $(y:\psi) \in \mathcal{B}$ (on $k = \text{size}(\psi)$ next slide)
 3. (**Consequence**) $\mathcal{M} \models (x:\varphi)$.



Case Analysis for Induction

case $y:\psi = y:\psi_1 \sqcap \psi_2$ Then $\{y:\psi_1, y:\psi_2\} \subseteq \mathcal{B}$ (\mathcal{T}_{\sqcap} -rule, saturation) so $\mathcal{M} \models (y:\psi_1)$ and $\mathcal{M} \models (y:\psi_2)$ and $\mathcal{M} \models (y:\psi_1 \sqcap \psi_2)$ (IH, Definition)

case $y:\psi = y:\psi_1 \sqcup \psi_2$ Then $(y:\psi_1) \in \mathcal{B}$ or $(y:\psi_2) \in \mathcal{B}$ (\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 $y:\psi = y:\exists R.\theta$ then $\{y R z, z:\theta\} \subseteq \mathcal{B}$ (z new variable) (\mathcal{T}_{\exists} -rules, saturation) so $\mathcal{M} \models (z:\theta)$ and $\mathcal{M} \models y R z$, thus $\mathcal{M} \models (y:\exists R.\theta)$. (IH, Definition)

case $y:\psi = y:\forall R.\theta$ Let $\langle [y], v \rangle \in [R]$ for some $r \in \mathcal{D}$
then $v = z$ for some variable z with $y R z \in \mathcal{B}$ (construction of $[R]$) So
 $(z:\theta) \in \mathcal{B}$ and $\mathcal{M} \models (z:\theta)$. (\mathcal{T}_{\forall} -rule, saturation, Def) Since v was arbitrary we
have $\mathcal{M} \models (y:\forall R.\theta)$.

- ▶ **Theorem 3.34.** \mathcal{T}_{AC} terminates
- ▶ To prove termination of a tableau algorithm, find a well-founded measure (function) that is decreased by all rules

$$\begin{array}{c} x:c \\ \frac{x:\overline{c}}{\perp} \mathcal{T}_{\perp} & \frac{x:\varphi \sqcap \psi}{x:\varphi \quad x:\psi} \mathcal{T}_{\sqcap} & \frac{x:\varphi \sqcup \psi}{x:\varphi \quad | \quad x:\psi} \mathcal{T}_{\sqcup} & \frac{x:\forall R.\varphi}{x R y \quad y:\varphi} \mathcal{T}_{\forall} & \frac{x:\exists R.\varphi}{x R y \quad y:\varphi} \mathcal{T}_{\exists} \end{array}$$

- ▶ **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 $x:\forall R.\psi$ at most as the number of R-successors of x .
(those y with $x R y$ above)
3. The R-successors are generated by $x:\exists R.\theta$ 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 .



- ▶ Idea: Work of tableau branches one after the other. (Branch size $\hat{=}$ space complexity)
- ▶ **Observation 3.35.** The size of the branches is *polynomial* in the size of the input formula:

$$\text{branch size} = |\text{input formulae}| + \#(\exists\text{-formulae}) \cdot \#(\forall\text{-formulae})$$

- ▶ **Proof Sketch:** Re-examine the termination proof and count: the first summand comes from P.3, the second one from P.3 and P.2 □
- ▶ **Theorem 3.36.** The satisfiability problem for \mathcal{ALC} is in **PSPACE**.
- ▶ **Theorem 3.37.** The satisfiability problem for \mathcal{ALC} is **PSPACE**-Complete.
- ▶ **Proof Sketch:** Reduce a **PSPACE**-complete problem to \mathcal{ALC} -satisfiability □
- ▶ **Theorem 3.38 (Time Complexity).**
The \mathcal{ALC} satisfiability problem is in **EXPTIME**.
- ▶ **Proof Sketch:** There can be exponentially many branches (already for \mathcal{PL}^0) □

The functional Algorithm for \mathcal{ALC}

- ▶ Observation: (leads to a better treatment for \exists)
 - ▶ the \mathcal{T}_\exists -rule generates the constraints $x \in R y$ and $y : \psi$ from $x : \exists R. \psi$
 - ▶ this triggers the \mathcal{T}_\forall -rule for $x : \forall R. \theta_i$, which generate $y : \theta_1, \dots, y : \theta_n$
 - ▶ for y we have $y : \psi$ and $y : \theta_1, \dots, y : \theta_n$. (do all of this in a single step)
 - ▶ we are only interested in non-emptiness, not in particular witnesses (leave them out)

- ▶ Definition 3.39. The **functional algorithm for \mathcal{ALC}** is

consistent(S) =

```
if {c, c̄} ⊆ S then false
elif '(φ □ ψ)' ∈ S and ('φ' ∉ S or 'ψ' ∉ S)
    then consistent(S ∪ {φ, ψ})
elif '(φ ∎ ψ)' ∈ S and {φ, ψ} ∉ S
    then consistent(S ∪ {φ}) or consistent(S ∪ {ψ})
elif forall '∃ R. ψ' ∈ S
    consistent({ψ} ∪ {θ ∈ θ | '∀ R. θ' ∈ S})
else true
```

- ▶ Relatively simple to implement. (good implementations optimized)
- ▶ But: This is restricted to \mathcal{ALC} . (extension to other DL difficult)

Extending the Tableau Algorithm by Concept Axioms

- ▶ Concept axioms, e.g. $\text{child} \sqsubseteq \text{son} \sqcup \text{daughter}$ cannot be handled in \mathcal{T}_{AC} 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 $\{x:\varphi\} \cup CA$ (CA : = set of concept axioms)
 - ▶ New rule for \exists :
$$\frac{x:\exists R.\varphi \quad CA = \{\alpha_1, \dots, \alpha_n\}}{y:\varphi \quad x R y \quad y:\alpha_1 \quad \vdots \quad y:\alpha_n} \mathcal{T}_{CA}^\exists$$
 (instead of \mathcal{T}_\exists)
- ▶ Problem: $CA := \{\exists R.c\}$ and start tableau with $x:d$ (non-termination)

Non-Termination of \mathcal{ALC} with Concept Axioms

- ▶ Problem: $CA := \{\exists R.c\}$ and start tableau with $x:d$. (non-termination)

$x:d$	start
$x:\exists R.c$	in CA
$x R y_1$	\mathcal{T}_\exists
$y_1:c$	\mathcal{T}_\exists
$y_1:\exists R.c$	\mathcal{T}_{CA}^\exists
$y_1 R y_2$	\mathcal{T}_\exists
$y_2:c$	\mathcal{T}_\exists
$y_2:\exists R.c$	\mathcal{T}_{CA}^\exists
...	

Solution: Loop-Check:

- ▶ Instead of a new variable y take an old variable z , if we can guarantee that whatever holds for y already holds for z .
- ▶ We can only do this, iff the \mathcal{T}_\forall -rule has been exhaustively applied.

- ▶ **Theorem 3.40.** The consistency problem of \mathcal{ALC} with concept axioms is decidable.
- ▶ Proof Sketch: \mathcal{ALC} with a suitable loop check terminates. □

3.3 ABoxes, Instance Testing, and ALC

Instance Test: Concept Membership

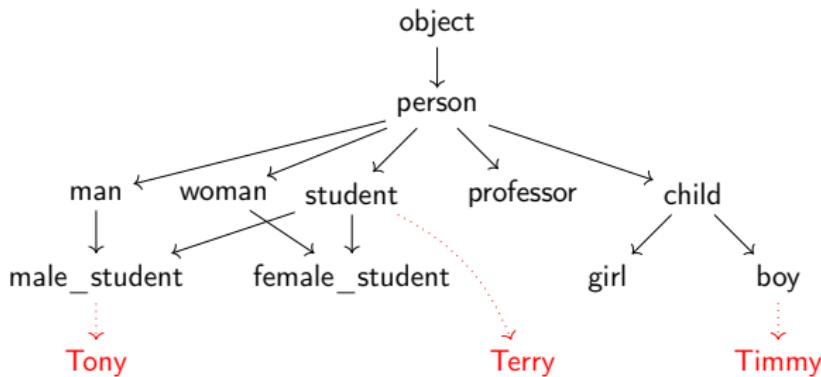
- ▶ **Definition 3.41.** An **instance test** computes whether – given an \mathcal{ALC} ontology – an individual is a member of a given class.
- ▶ **Example 3.42 (An Ontology).**

TBox (terminological Box)	ABox (assertional Box, data base)
woman = person \sqcap has_Y man = person \sqcap has_Y	tony:person tony:has_Y
	Tony is a person Tony has a y-chrom

This **entails**: tony:man (Tony is a man).

- ▶ **Problem:** Can we compute this?

- ▶ **Definition 3.43.** **Realization** is the computation of all instance relations between ABox objects and TBox concepts.
- ▶ **Observation:** It is sufficient to remember the lowest concepts in the subsumption graph.
(rest by subsumption)



- ▶ **Example 3.44.** If **tony:male_student** is known, we do not need **tony:man**.

- ▶ There are different kinds of interactions between TBox and ABox in \mathcal{ALC} and in description logics in general.
- ▶ **Example 3.45.**

property	example
internally inconsistent	tony:student, tony: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

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)
- ▶ **Necessary changes:** (no big deal)
 - ▶ Normalize ABox wrt. TBox.
 - ▶ Initialize the tableau with ABox in NNF. (definition expansion) (so it can be used)
- ▶ **Example 3.46.**

Example: add <u>mary:genius</u> to determine ABox, $TBox \models mary:genius$			
TBox	$\text{happy_prof} = \text{prof} \sqcap \forall \text{has_grad}. \text{genius}$	tony:prof $\sqcap \forall \text{has_grad}. \text{genius}$ tony has_grad mary mary:genius tony:prof tony: $\forall \text{has_grad}. \text{genius}$ mary:genius \perp	TBox ABox Query T_N T_P T_V T_{\perp}
ABox	tony:happy_prof tony has_grad mary		

- ▶ **Note:** The instance test is the base for realization. (remember?)
- ▶ **Idea:** Extend to more complex ABox queries. (e.g. give me all instances of φ)

4 Description Logics and the Semantic Web

- ▶ **Definition 4.1.** The **Resource Description Framework (RDF)** is a framework for describing resources on the web. It is an **XML** vocabulary developed by the W3C.
- ▶ **Note:** **RDF** is designed to be read and understood by computers, not to be displayed to people. (it shows)
- ▶ **Example 4.2.** **RDF** can be used for describing (all "objects on the WWW")
 - ▶ properties for shopping items, such as price and availability
 - ▶ time schedules for web events
 - ▶ information about web pages (content, author, created and modified date)
 - ▶ content and rating for web pictures
 - ▶ content for search engines
 - ▶ electronic libraries

- ▶ RDF describes resources with properties and property values.
- ▶ RDF uses Web identifiers (URLs) to identify resources.
- ▶ **Definition 4.3.** A **resource** is anything that can have a **URI**, such as <http://www.fau.de>.
- ▶ **Definition 4.4.** A **property** is a resource that has a name, such as *author* or *homepage*, and a **property value** is the value of a property, such as *Michael Kohlhase* or <http://kwarc.info/kohlhase>. (a **property value** can be another **resource**)
- ▶ **Definition 4.5.** A **RDF statement** s (also known as a **triple**) consists of a resource (the **subject** of s), a **property** (the **predicate** of s), and a **property value** (the **object** of s). A set of RDF triples is called an **RDF graph**.
- ▶ **Example 4.6.** Statement: $[This\ slide]^{subj} \text{ has been } [author]^{pred} \text{ ed by } [Michael Kohlhase]^{obj}$

- ▶ RDF is a concrete XML vocabulary for writing statements
- ▶ **Example 4.7.** The following RDF document could describe the slides as a resource

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc= "http://purl.org/dc/elements/1.1/">
  <rdf:Description about="https://.../CompLog/kr/en/rdf.tex">
    <dc:creator>Michael Kohlhase</dc:creator>
    <dc:source>http://www.w3schools.com/rdf</dc:source>
  </rdf:Description>
</rdf:RDF>
```

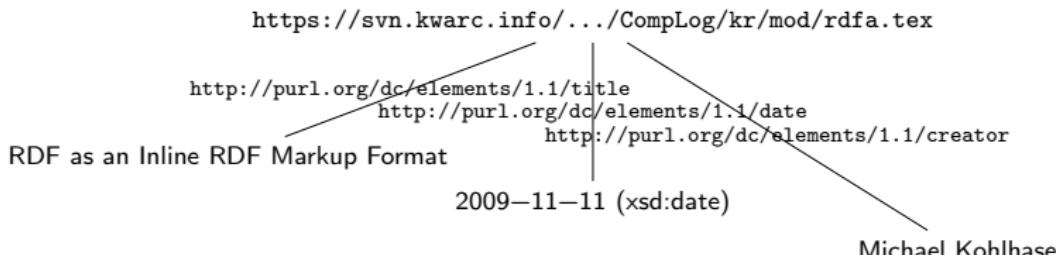
This RDF document makes two statements:

- ▶ The subject of both is given in the about attribute of the rdf:Description element
- ▶ The predicates are given by the element names of its children
- ▶ The objects are given in the elements as URIs or literal content.
- ▶ **Intuitively:** RDF is a web-scalable way to write down ABox information.

RDFa as an Inline RDF Markup Format

- ▶ Problem: RDF is a standoff markup format (annotate by URIs pointing into other files)
- ▶ Example 4.8.

```
<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>
```



- ▶ Idea: RDF triples are ABox entries $h \text{ R } s$ or $h:\varphi$.
- ▶ Example 4.9. h is the resource for Ian Horrocks, s is the resource for Ulrike Sattler, R is the relation “hasColleague”, and φ is the class foaf:Person

```
<rdf:Description about="some.uri/person/ian_horrocks">
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <hasColleague resource="some.uri/person/uli_sattler"/>
</rdf:Description>
```

- ▶ Idea: Now, we need an similar language for TBoxes

(based on \mathcal{ALC})

OWL as an Ontology Language for the Semantic Web

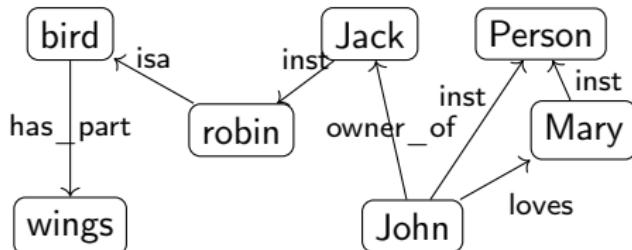
- ▶ **Task:** Complement RDF (**ABox**) with a **TBox** language.
- ▶ **Idea:** Make use of resources that are values in `rdf:type`. (called **Classes**)
- ▶ **Definition 4.10.** OWL (the **ontology web language**) is a language for encoding **TBox** information about **RDF** classes.
- ▶ **Example 4.11 (A concept definition for “Mother”).**

$\text{Mother} = \text{Woman} \sqcap \text{Parent}$ is represented as

XML Syntax	Functional Syntax
<pre><EquivalentClasses> <Class IRI="Mother"/> <ObjectIntersectionOf> <Class IRI="Woman"/> <Class IRI="Parent"/> </ObjectIntersectionOf> </EquivalentClasses></pre>	<pre>EquivalentClasses(:Mother ObjectIntersectionOf(:Woman :Parent))</pre>

Extended OWL Example in Functional Syntax

- **Example 4.12.** The semantic network from 1.5 can be expressed in OWL (in functional syntax)



- ClassAssertion formalizes the “inst” relation,
- ObjectPropertyAssertion formalizes relations,
- SubClassOf formalizes the “isa” relation,
- for the “has_part” relation, we have to specify that *all birds have a part that is a wing* or equivalently *the class of birds is a subclass of all objects that have some wing*.

Extended OWL Example in Functional Syntax

- ▶ **Example 4.12.** The semantic network from 1.5 can be expressed in OWL (in functional syntax)

```
ClassAssertion (:Jack :robin)
ClassAssertion(:John :person)
ClassAssertion (:Mary :person)
ObjectPropertyAssertion(:loves :John :Mary)
ObjectPropertyAssertion(:owner :John :Jack)
SubClassOf(:robin :bird)
SubClassOf (:bird ObjectSomeValuesFrom(:hasPart :wing))
```

- ▶ ClassAssertion formalizes the “inst” relation,
- ▶ ObjectPropertyAssertion formalizes relations,
- ▶ SubClassOf formalizes the “isa” relation,
- ▶ for the “has_part” relation, we have to specify that *all birds have a part that is a wing* or equivalently *the class of birds is a subclass of all objects that have some wing*.

SPARQL an RDF Query language

- ▶ **Definition 4.13.** SPARQL, the “SPARQL Protocol and RDF Query Language” is an [RDF](#) query language, able to retrieve and manipulate data stored in [RDF](#). The SPARQL language was standardized by the World Wide Web Consortium in 2008 [PS08].
- ▶ SPARQL is pronounced like the word “*sparkle*”.
- ▶ **Definition 4.14.** A system is called a [SPARQL endpoint](#), iff it answers SPARQL queries.
- ▶ **Example 4.15.**

Query for person names and their e-mails from a triple store with FOAF data.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?name ?email
```

```
WHERE {
```

```
    ?person a foaf:Person.
```

```
    ?person foaf:name ?name.
```

```
    ?person foaf:mbox ?email.
```

```
}
```

SPARQL Applications: DBpedia

Emmy Noether



- ▶ **Typical Application:** DBpedia screen-scrapes Wikipedia fact boxes for RDF triples and uses SPARQL for querying the induced triple store.

▶ Example 4.16 (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.

Born	Amalie Emmy Noether 23 March 1882 Erlangen, Bavaria, German Empire
Died	14 April 1935 (aged 53) Bryn Mawr, Pennsylvania, United States
Nationality	German
Alma mater	University of Erlangen
Known for	Abstract algebra Theoretical physics Noether's theorem

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_Benabdelah	:Algeria	:Wydad_Casablanca	:Morocco	67000
:Ailton_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_Ferreyya	: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		:FC_Red_Bull_Salzburg	:Austria	31000

- ▶ **Definition 4.17.** A **triplestore** or **RDF store** is a purpose-built database for the storage **RDF graphs** and retrieval of **RDF triples** usually through variants of SPARQL.
- ▶ Common **triplestores** include
 - ▶ Virtuoso: <https://virtuoso.openlinksw.com/> (used in DBpedia)
 - ▶ GraphDB: <http://graphdb.ontotext.com/> (often used in WissKI)
 - ▶ blazegraph: <https://blazegraph.com/> (open source; used in WikiData)
- ▶ **Definition 4.18.** A **description logic reasoner** implements reasoning services based on a satisfiability 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**.

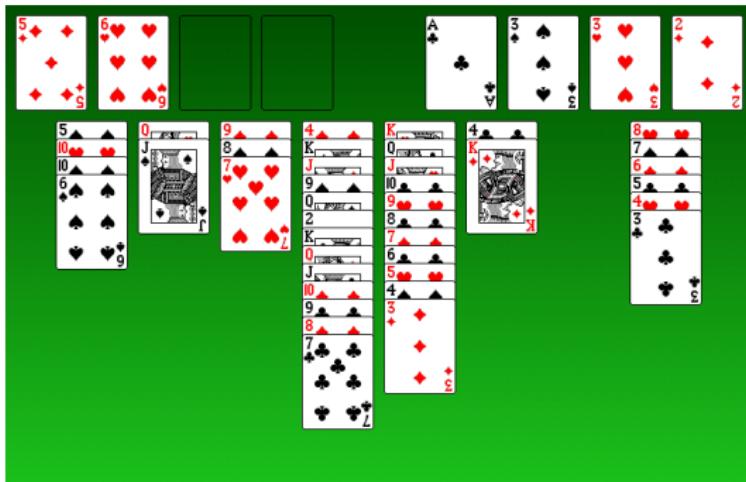
Part IV Planning & Acting

Chapter 16 Planning I: Framework

1 Planning: Introduction

Reminder: Classical Search Problems

► Example 1.1 (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.

- ▶ Ambition: Write one program that can solve all classical [search problems](#).
- ▶ **Definition 1.2.** Let Π be a [search problem](#) (see)
 - ▶ The **blackbox description** of Π is an [API](#) providing functionality allowing to construct the state space: `InitialState()`, `GoalTest(s)`, ...
 - ▶ “Specifying the problem” $\hat{=}$ 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” $\hat{=}$ writing a problem description.
- ▶ Here, “[problem description language](#)” $\hat{=}$ planning language.

How does a planning language describe a problem?

- ▶ **Definition 1.3.** A **planning language** is a logical language for the components of a **search problem**; in particular a *logical description* of the
 - ▶ possible **states** (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)

How does a planning language describe a problem?

- ▶ **Definition 1.3.** A **planning language** is a logical language for the components of a **search problem**; in particular a *logical description* of the
 - ▶ possible states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
 - ▶ initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)

How does a planning language describe a problem?

- ▶ **Definition 1.3.** A **planning language** is a logical language for the components of a **search problem**; in particular a *logical description* of the
 - ▶ possible states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
 - ▶ initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)
 - ▶ goal test G (vs. a goal-test function). (E.g.: $Eq(x,2)$.)

How does a planning language describe a problem?

- ▶ **Definition 1.3.** A **planning language** is a logical language for the components of a **search problem**; in particular a *logical description* of the
 - ▶ possible **states** (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$)
 - ▶ **initial state** I (vs. data structures). (E.g.: $Eq(x, 1)$.)
 - ▶ **goal test** G (vs. a goal-test function). (E.g.: $Eq(x, 2)$.)
 - ▶ set A of **actions** in terms of **preconditions** and **effects** (vs. functions returning applicable actions and successor states). (E.g.: “increment x : pre $Eq(x, 1)$, eff $Eq(x, 2) \wedge \neg Eq(x, 1)$ ”.)

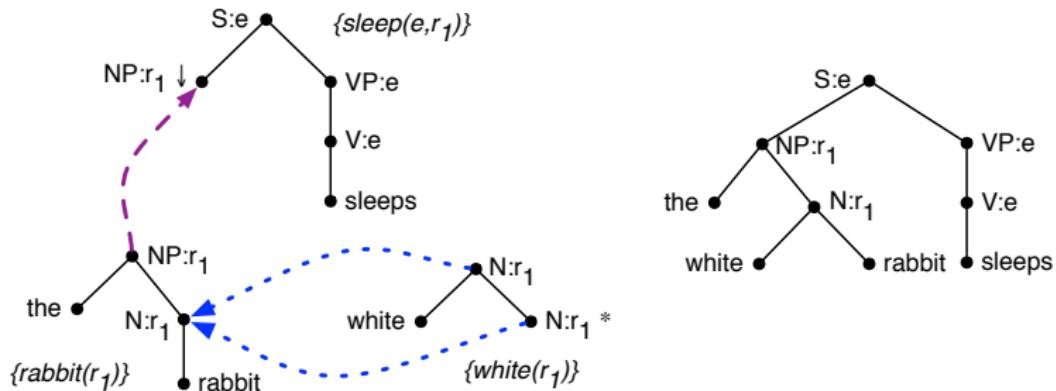
A logical description of all of these is called a **planning task**.

How does a planning language describe a problem?

- ▶ **Definition 1.3.** A **planning language** is a logical language for the components of a **search problem**; in particular a *logical description* of the
 - ▶ possible **states** (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$)
 - ▶ **initial state** I (vs. data structures). (E.g.: $Eq(x, 1)$.)
 - ▶ **goal test** G (vs. a goal-test function). (E.g.: $Eq(x, 2)$.)
 - ▶ set A of **actions** in terms of **preconditions** and **effects** (vs. functions returning applicable actions and successor states). (E.g.: “increment x : pre $Eq(x, 1)$, eff $Eq(x, 2) \wedge \neg Eq(x, 1)$ ”.)
- A logical description of all of these is called a **planning task**.
- ▶ **Definition 1.4.** Solution (**plan**) $\hat{=}$ sequence of actions from A , transforming I into a state that satisfies G . (E.g.: “increment x ”.)

- ▶ Disclaimer: Planning languages go way beyond classical search problems. There are variants for inaccessible, stochastic, dynamic, continuous, and multi-agent settings.
- ▶ We focus on classical search for simplicity (and practical relevance).
- ▶ For a comprehensive overview, see [GNT04].

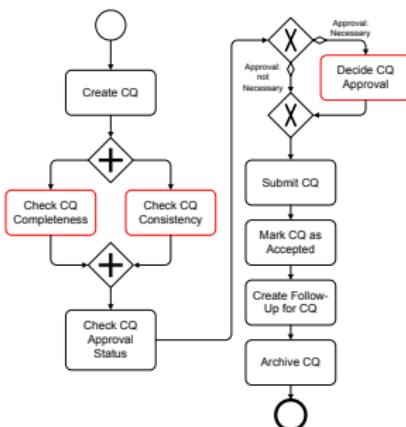
Application: Natural Language Generation



- ▶ **Input:** Tree-adjoining grammar, intended meaning.
- ▶ **Output:** Sentence expressing that meaning.

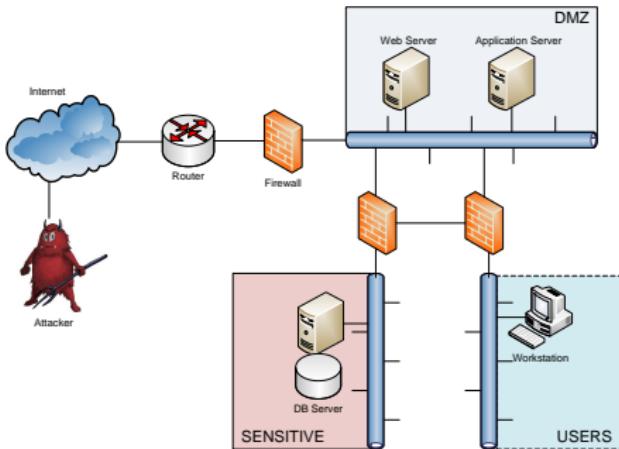
Application: Business Process Templates at SAP

Action name	precondition	effect
Check CQ Completeness	CQ.archiving:notArchived	CQ.completeness:complete OR CQ.completeness:notComplete
Check CQ Consistency	CQ.archiving:notArchived	CQ.consistency:consistent OR CQ.consistency:notConsistent
Check CQ Approval Status	CQ.archiving:notArchived AND CQ.approval:notChecked AND CQ.completeness:complete AND CQ.consistency:consistent	CQ.approval:necessary OR CQ.approval:notNecessary
Decide CQ Approval	CQ.archiving:notArchived AND CQ.approval:necessary	CQ.approval:granted OR CQ.approval:notGranted
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Create Follow-Up for CQ	CQ.archiving:notArchived AND CQ.acceptance:accepted	CQ.followUp:documentCreated
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived



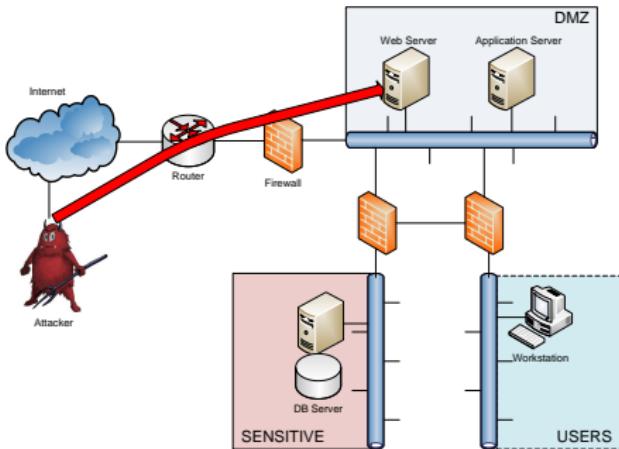
- ▶ **Input:** model of behavior of activities on business objects, process endpoint.
- ▶ **Output:** Process template leading to this point.

Application: Automatic Hacking



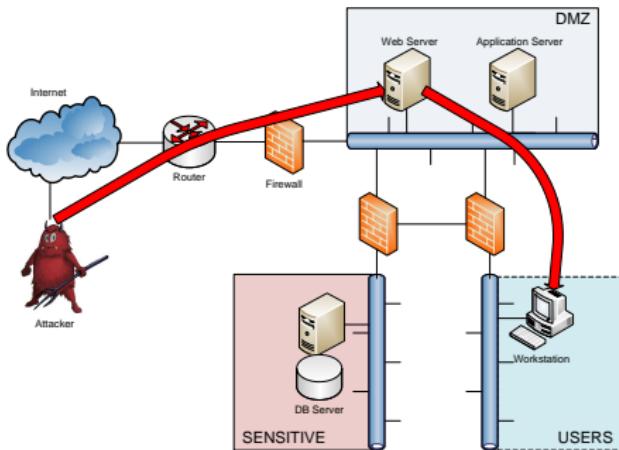
- ▶ **Input:** Network configuration, location of sensible data.
- ▶ **Output:** Sequence of exploits giving access to that data.

Application: Automatic Hacking



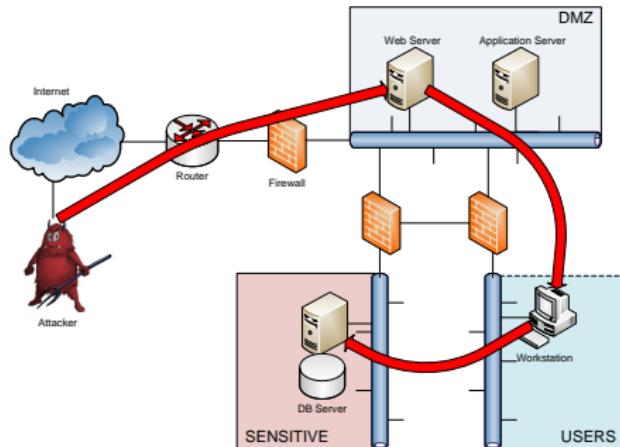
- ▶ **Input:** Network configuration, location of sensible data.
- ▶ **Output:** Sequence of exploits giving access to that data.

Application: Automatic Hacking



- ▶ **Input:** Network configuration, location of sensible data.
- ▶ **Output:** Sequence of exploits giving access to that data.

Application: Automatic Hacking



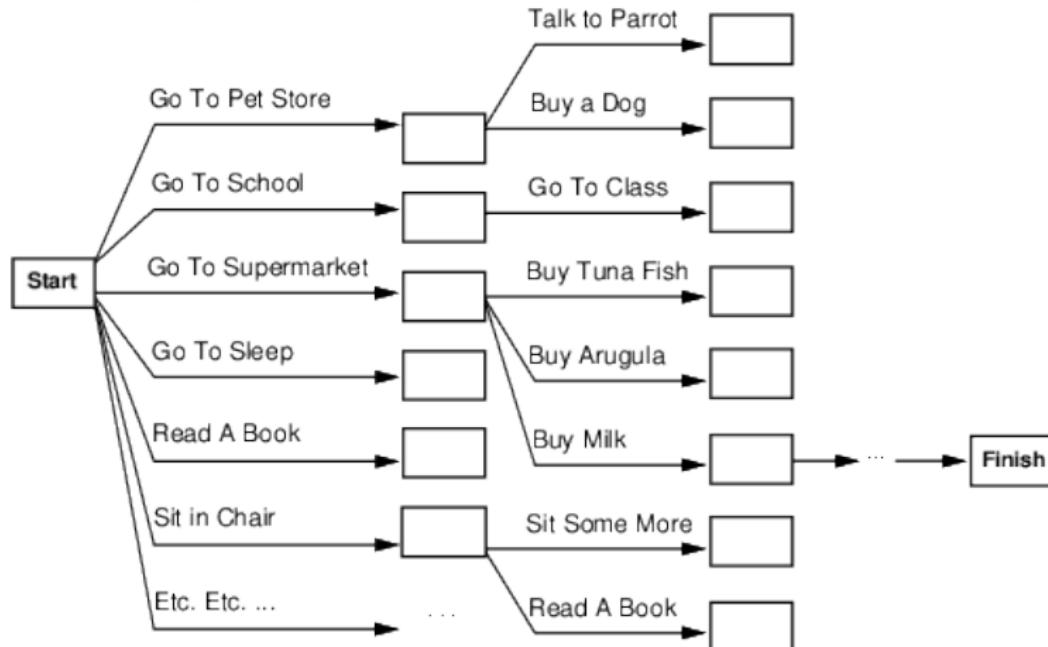
- ▶ **Input:** Network configuration, location of sensible data.
- ▶ **Output:** Sequence of exploits giving access to that data.

Reminder: General Problem Solving, Pros and Cons

- ▶ **Powerful:** In some applications, generality is absolutely necessary. (E.g. SAP)
- ▶ **Quick:** Rapid prototyping: 10s **lines** of problem description vs. 1000s **lines** of C++ code. (E.g. language generation)
- ▶ **Flexible:** Adapt/maintain *the description*. (E.g. network security)
- ▶ **Intelligent:** Determines automatically how to solve a complex problem effectively! (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 effective".
- ▶ **Research Question:** How to make fully automatic algorithms effective?

Search vs. planning

- ▶ Consider the task *get milk, bananas, and a cordless drill*
- ▶ Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate

- ▶ Planning systems do the following:

1. open up action and goal representation to allow selection
- FAU divide-and-conquer by subgoaling

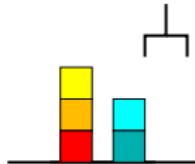
Reminder: Greedy Best-First Search and A*

- ▶ Duplicate elimination omitted for simplicity:

```
function Greedy_Best-First_Search [ $A^*$ ](problem) returns a solution, or failure
    node := a node  $n$  with  $n.state=problem.InitialState$ 
    frontier := a priority queue ordered by ascending  $h$  [ $g + h$ ], only element  $n$ 
    loop do
        if Empty?(frontier) then return failure
         $n := \text{Pop}(\text{frontier})$ 
        if problem.GoalTest( $n.State$ ) then return Solution( $n$ )
        for each action  $a$  in problem.Actions( $n.State$ ) do
             $n' := \text{ChildNode}(\text{problem}, n, a)$ 
            Insert( $n'$ ,  $h(n') [g(n') + h(n')]$ ), frontier)
```

- ▶ Is Greedy Best-First Search optimal? No \leadsto *satisficing* planning.
- ▶ Is A^* optimal? Yes, but only if h is admissible \leadsto *optimal* planning, **with such h .**

► Example 1.5.



- ▶ n blocks, 1 hand.
- ▶ A single action either takes a block with the hand or puts a block we’re holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

- ▶ ***Observation 1.6.*** 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!

- ▶ **Definition 1.7.** 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 effective in practice.
- ▶ **Definition 1.8.** 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.

2 The History of Planning

- ▶ 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.

- ▶ **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 $move(A, B)$; $\rightsquigarrow open = \{at(A)\} \dots$
 - ▶ **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. $F_0 = \text{at}(A)$; $A_0 = \{\text{move}(A,B)\}$; $F_1 = \{\text{at}(B)\}$;
mutex $A_0 = \{\text{move}(A,B), \text{move}(A,C)\}$.
- Popular when: 1995 – 2000.
- Approach: In a forward phase, build a layered “planning graph” whose “time steps” capture which pairs of actions 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 $\text{at}(A)_0$, $\text{at}(B)_0$, $\text{move}(A,B)_0$, $\text{move}(A,C)_0$, $\text{at}(A)_1$, $\text{at}(B)_1$; clauses to encode transition behavior e.g. $\text{at}(B)_1 \xrightarrow{F} \vee \text{move}(A,B)_0 \xrightarrow{T}$; unit clauses to encode initial state $\text{at}(A)_0 \xrightarrow{T}$, $\text{at}(B)_0 \xrightarrow{T}$; unit clauses to encode goal $\text{at}(B)_1 \xrightarrow{T}$.
- 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,

...

- ▶ 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 2.1.** The **International Planning Competition (IPC)** is an event for benchmarking **planners** (<http://ipc.icaps-conference.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: \approx 50 **domains**, \gg 1000 **instances**, 74 (!!) planners in 2011

- ▶ **Example 2.2.** 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:

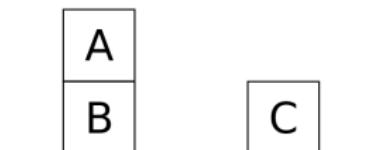
Planning History, p.s.: Planning is Non-Trivial!

- ▶ **Example 2.2.** The **Sussman anomaly** is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\text{on}(A, B)$ and $\text{on}(B, C)$ fail:

- ▶ If we pursue $\text{on}(A, B)$ by unstacking C , and moving A onto B , we achieve the first subgoal, but cannot achieve the second without undoing the first.



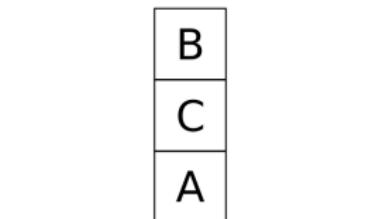
Planning History, p.s.: Planning is Non-Trivial!

- ▶ **Example 2.2.** The **Sussman anomaly** is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\text{on}(A, B)$ and $\text{on}(B, C)$ fail:

- ▶ If we pursue $\text{on}(A, B)$ by unstacking C , and moving A onto B , we achieve the first subgoal, but cannot achieve the second without undoing the first.
- ▶ If we pursue $\text{on}(B, C)$ by moving B onto C , we achieve the second subgoal, but cannot achieve the first without undoing the second.



3 The STRIPS Planning Formalism

- ▶ **Definition 3.1.** 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.

- ▶ **Definition 3.2.** A **STRIPS task** is a quadruple $\Pi = \langle P, A, I, G \rangle$ where:
 - ▶ P is a finite set of **facts** (aka **propositions**).
 - ▶ A is a finite set of **actions**; each $a \in A$ is a triple $a = \langle \text{pre}_a, \text{add}_a, \text{del}_a \rangle$ of subsets of P referred to as the **action's precondition**, **add list**, and **delete list** respectively; we require that $\text{add}_a \cap \text{del}_a = \emptyset$.
 - ▶ $I \subseteq P$ is the **initial state**.
 - ▶ $G \subseteq P$ is the **goal**.
- ▶ 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 3.3 (Salesman Travelling in Australia).



Strictly speaking, this is not actually a **TSP** problem instance;
simplified/adapted for illustration.

STRIPS Encoding of “TSP”

► Example 3.4 (continuing).



- Facts P : $\{at(x), vis(x) | x \in \{Sy, Ad, Br, Pe, Da\}\}$.
- Initial state I : $\{at(Sy), vis(Sy)\}$.
- Goal G : $\{at(Sy)\} \cup \{vis(x) | x \in \{Sy, Ad, Br, Pe, Da\}\}$.
- Actions $a \in A$: $drv(x, y)$ where x and y have a road.
 - Precondition pre_a : $\{at(x)\}$.
 - Add list add_a : $\{at(y), vis(y)\}$.
 - Delete list del_a : $\{at(x)\}$.
- Plan: $\langle drv(Sy, Br), drv(Br, Sy), drv(Sy, Ad), drv(Ad, Pe), drv(Pe, Ad), \dots, drv(Ad, Da), drv(Da, Ad), drv(Ad, Sy) \rangle$

STRIPS Planning: Semantics

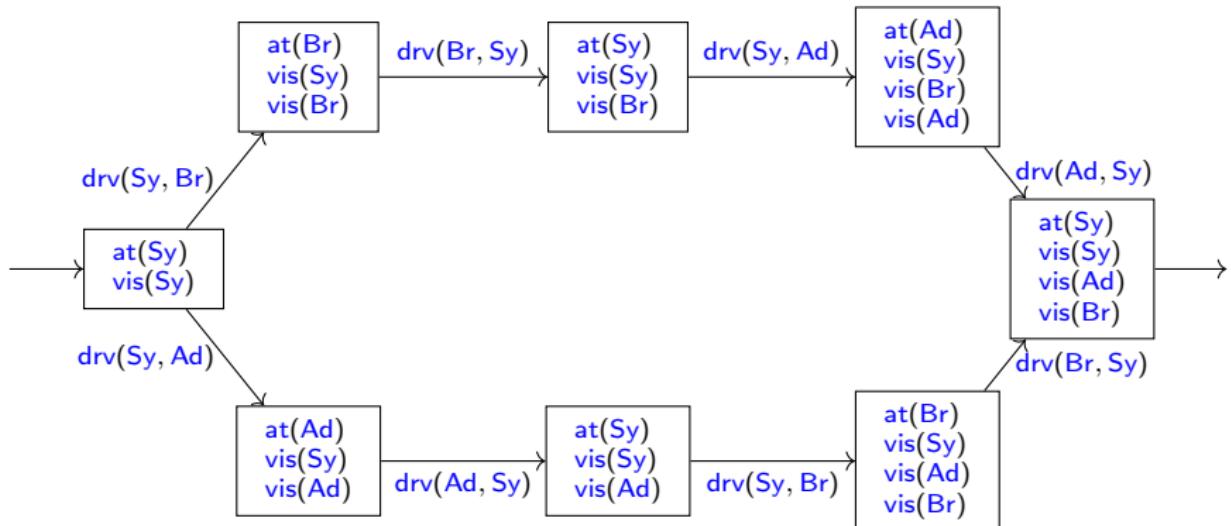
- Idea: We define a **plan** for a STRIPS task Π as a **solution** to an induced **search problem** Θ_Π .
- **Definition 3.5.** Let $\Pi = \langle P, A, I, G \rangle$ be a STRIPS task. The **state space** of Π is a **search problem** $\Theta_\Pi = \langle S, A, T, I, S^G \rangle$ where:
 - The **states** (also **world states**) $S := \mathcal{P}(P)$ are the subsets of P .
 - A is just Π 's action set. (so we can define **plans** easily)
 - The **transition model** T is $\{s \xrightarrow{a} \text{apply}(s, a) \mid \text{pre}_a \subseteq s\}$.
If $\text{pre}_a \subseteq s$, then a is **applicable** in s and $\text{apply}(s, a) := s \cup \text{add}_a \setminus \text{del}_a$. If $\text{pre}_a \not\subseteq s$, then $\text{apply}(s, a)$ is undefined.
 - I is Π 's initial state.
 - The **goal states** $S^G = \{s \in S \mid G \subseteq s\}$ are those that satisfy Π 's goal.
- An (optimal) **plan** for $s \in S$ is an (optimal) **solution** for s in Θ_Π , i.e., a path from s to some $s' \in S^G$. A **solution** for I is called a **plan** for Π . Π is **solvable** if a plan for Π exists.
For $a = \langle a_1, \dots, a_n \rangle$, $\text{apply}(s, a) := \text{apply}(s, \text{apply}(s, a_2 \dots \text{apply}(s, a_n)))$ if each a_i is applicable in the respective state; else, $\text{apply}(s, a)$ is undefined.
- **Note:** This is exactly like the state spaces from , without a cost function. Solutions are defined as before (paths from I to a state in S^G).

STRIPS Encoding of Simplified “TSP”



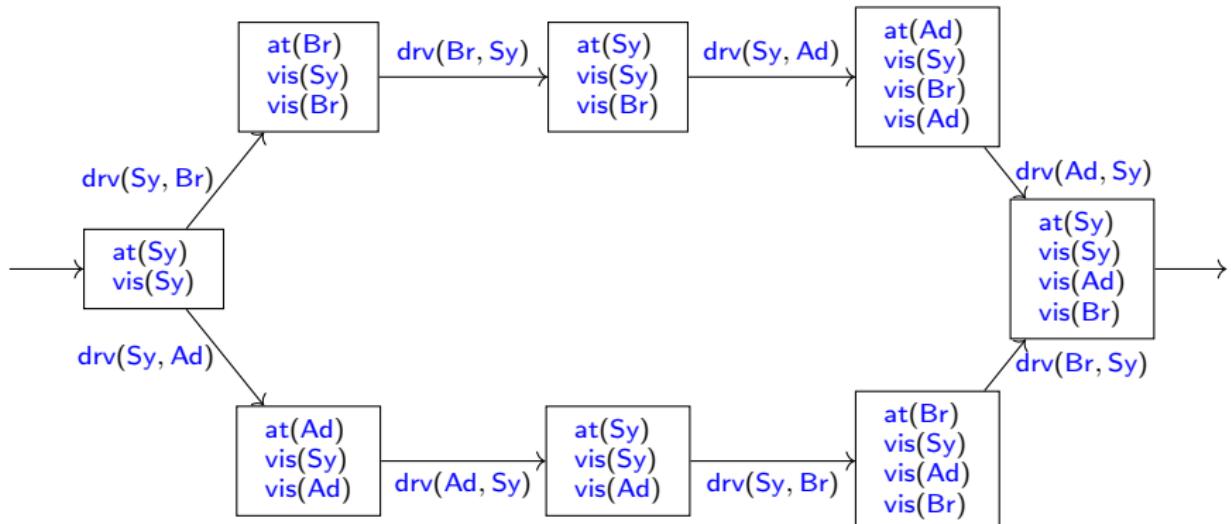
- ▶ Facts P : $\{\text{at}(x), \text{vis}(x) | x \in \{\text{Sy}, \text{Ad}, \text{Br}\}\}$.
- ▶ Initial state I : $\{\text{at}(\text{Sy}), \text{vis}(\text{Sy})\}$.
- ▶ Goal G : $\{\text{vis}(x) | x \in \{\text{Sy}, \text{Ad}, \text{Br}\}\}$.
(Note: no “ $\text{at}(\text{Sy})$ ”)
- ▶ Actions $a \in A$: $\text{drv}(x, y)$ where x y have a road.
 - ▶ Precondition pre_a : $\{\text{at}(x)\}$.
 - ▶ Add list add_a : $\{\text{at}(y), \text{vis}(y)\}$.
 - ▶ Delete list del_a : $\{\text{at}(x)\}$.

Encoding of Simplified “TSP”: State Space



► Question: Are there any solutions in this graph?

Encoding of Simplified “TSP”: State Space



► Question: Are there any solutions in this graph?

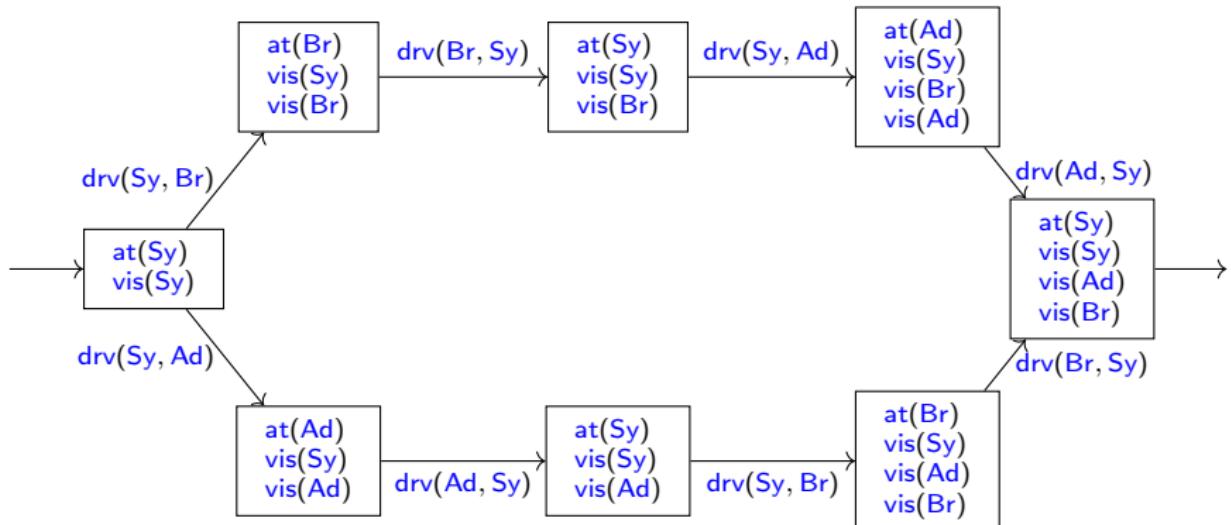
► Answer: Yes, two:

- $\text{drv(Sy, Br)}, \text{drv(Ad, Sy)}, \text{drv(Sy, Br)}$
- $\text{drv(Sy, Ad)}, \text{drv(Ad, Sy)}, \text{drv(Sy, Br)}$.

(upper path)

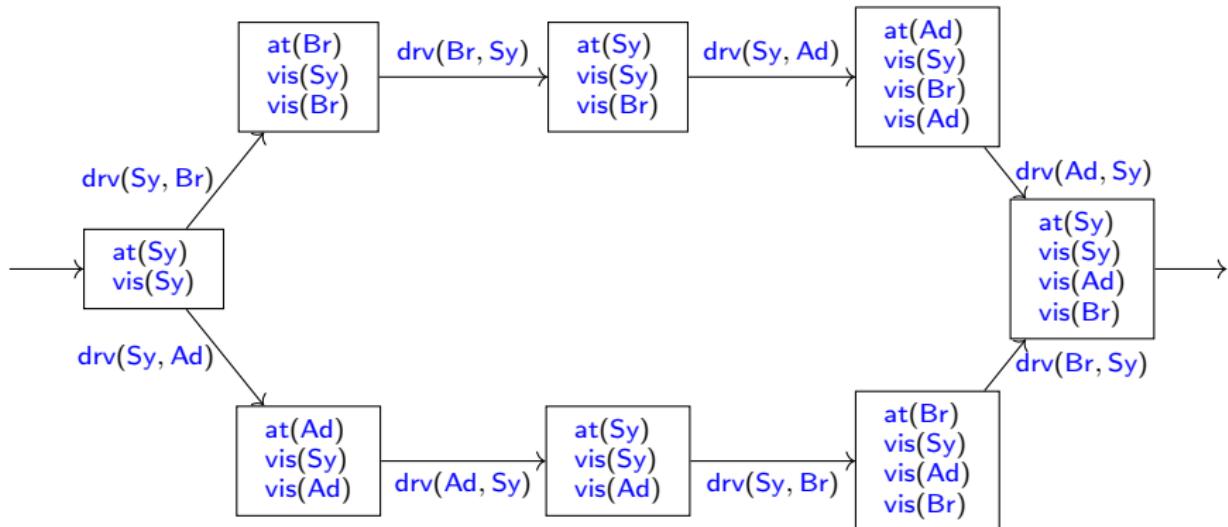
(lower path)

Encoding of Simplified “TSP”: State Space



- ▶ **Question:** Are there any solutions in this graph?
- ▶ **Answer:** Yes, two:
 - ▶ `drv(Sy, Br), drv(Br, Sy), drv(Sy, Ad)` (upper path)
 - ▶ `drv(Sy, Ad), drv(Ad, Sy), drv(Sy, Br)`. (lower path)
- ▶ **Question:** Is the graph above actually the state space?

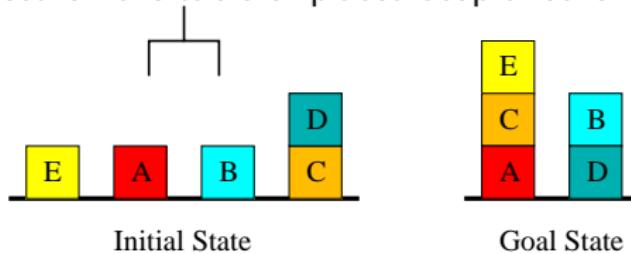
Encoding of Simplified “TSP”: State Space



- ▶ Question: Are there any solutions in this graph?
- ▶ Answer: Yes, two:
 - ▶ $\text{drv}(\text{Sy}, \text{Br}), \text{drv}(\text{Br}, \text{Sy}), \text{drv}(\text{Sy}, \text{Ad})$ (upper path)
 - ▶ $\text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Ad}, \text{Sy}), \text{drv}(\text{Sy}, \text{Br})$. (lower path)
- ▶ Question: Is the graph above actually the state space?
- ▶ Answer: No, only the reachable part. E.g., Θ_{Π} also includes the states $\{\text{vis}(\text{Sy})\}$ and $\{\text{at}(\text{Sy}), \text{at}(\text{Br})\}$.

The Blocksworld

- ▶ **Definition 3.6.** The **blocks world** is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- ▶ **Example 3.7.**

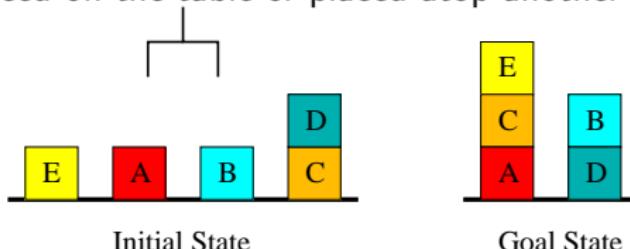


- ▶ Facts: `on(x, y)`, `onTable(x)`, `clear(x)`, `holding(x)`, `armEmpty`.

The Blocksworld

- ▶ **Definition 3.6.** The **blocks world** is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.

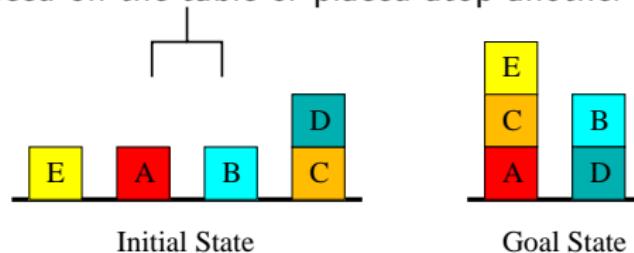
- ▶ **Example 3.7.**



- ▶ Facts: `on(x, y)`, `onTable(x)`, `clear(x)`, `holding(x)`, `armEmpty`.
- ▶ Initial state: `{onTable(E), clear(E), ..., onTable(C), on(D, C), clear(D), armEmpty}`.

The Blocksworld

- ▶ **Definition 3.6.** The **blocks world** is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
 - ▶ **Example 3.7.**

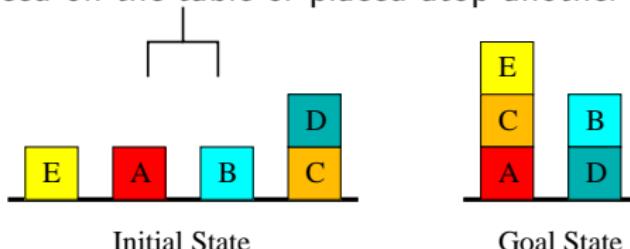


- Facts: $\text{on}(x, y)$, $\text{onTable}(x)$, $\text{clear}(x)$, $\text{holding}(x)$, armEmpty .
 - Initial state: $\{\text{onTable}(E), \text{clear}(E), \dots, \text{onTable}(C), \text{on}(D, C), \text{clear}(D), \text{armEmpty}\}$.
 - Goal: $\{\text{on}(E, C), \text{on}(C, A), \text{on}(B, D)\}$.

The Blocksworld

- ▶ **Definition 3.6.** The **blocks world** is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.

- ▶ **Example 3.7.**

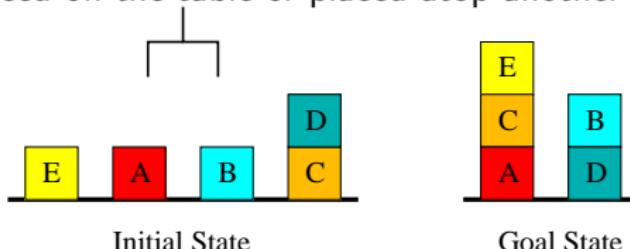


- ▶ Facts: `on(x, y)`, `onTable(x)`, `clear(x)`, `holding(x)`, `armEmpty`.
- ▶ Initial state: `{onTable(E), clear(E), ..., onTable(C), on(D, C), clear(D), armEmpty}`.
- ▶ Goal: `{on(E, C), on(C, A), on(B, D)}`.
- ▶ Actions: `stack(x, y)`, `unstack(x, y)`, `putdown(x)`, `pickup(x)`.

The Blocksworld

- ▶ **Definition 3.6.** The **blocks world** is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.

- ▶ **Example 3.7.**



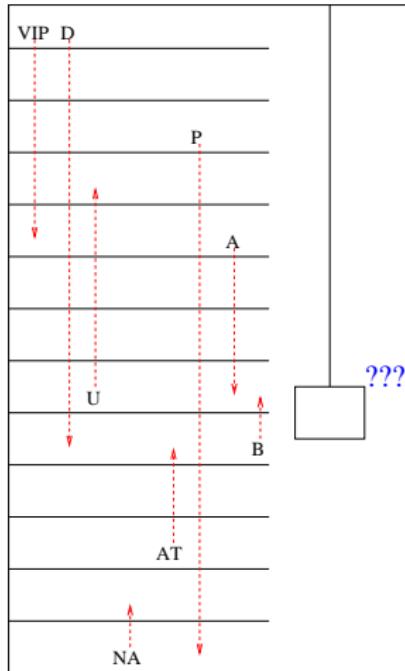
- ▶ Facts: `on(x,y)`, `onTable(x)`, `clear(x)`, `holding(x)`, `armEmpty`.
 - ▶ Initial state: `{onTable(E), clear(E), ..., onTable(C), on(D, C), clear(D), armEmpty}`.
 - ▶ Goal: `{on(E, C), on(C, A), on(B, D)}`.
 - ▶ Actions: `stack(x, y)`, `unstack(x, y)`, `putdown(x)`, `pickup(x)`.
 - ▶ **stack(x, y)?**
- `pre : {holding(x), clear(y)}`
`add : {on(x, y), armEmpty}`
`del : {holding(x), clear(y)}`.

Miconic-10: A Real-World Example

- ▶ **Example 3.8.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP:
- ▶ D:
- ▶ NA:
- ▶ AT:
- ▶ A, B:
- ▶ P:

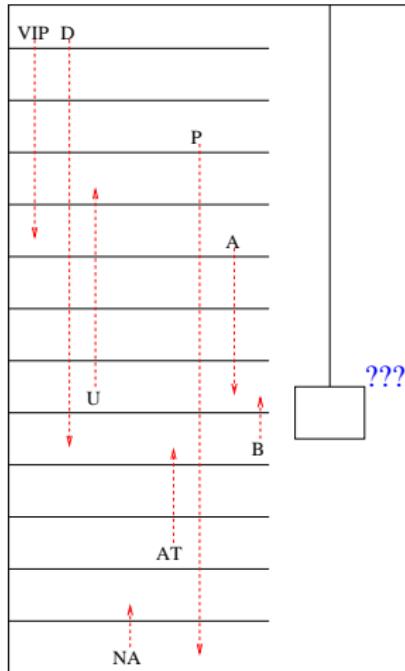


Miconic-10: A Real-World Example

- ▶ **Example 3.8.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D:
- ▶ NA:
- ▶ AT:
- ▶ A, B:
- ▶ P:

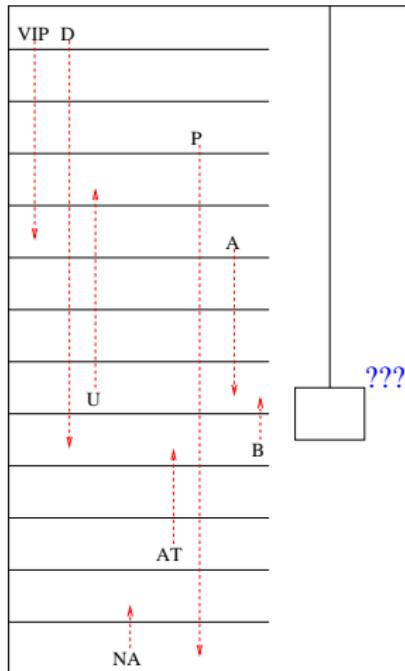


Miconic-10: A Real-World Example

- ▶ **Example 3.8.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D: Lift may only go *down* when inside; similar for U.
- ▶ NA:
- ▶ AT:
- ▶ A, B:
- ▶ P:

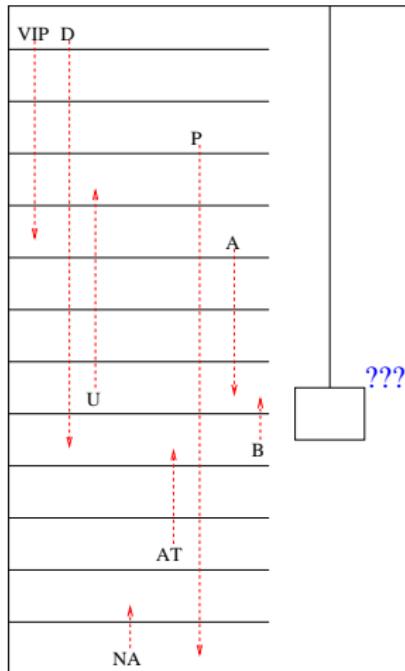


Miconic-10: A Real-World Example

- ▶ **Example 3.8.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D: Lift may only go *down* when inside; similar for U.
- ▶ NA: Never-alone
- ▶ AT:
- ▶ A, B:
- ▶ P:

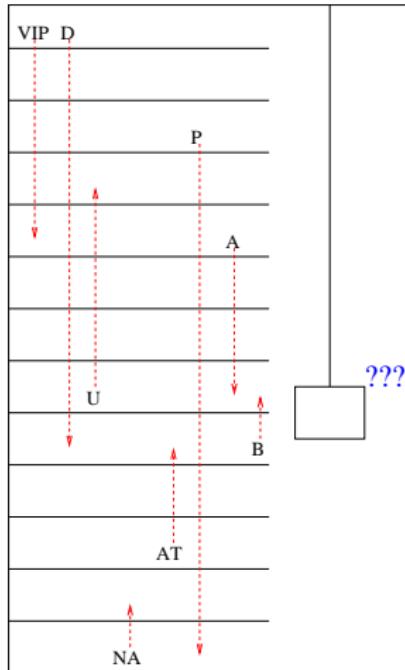


Miconic-10: A Real-World Example

- ▶ **Example 3.8.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D: Lift may only go *down* when inside; similar for U.
- ▶ NA: Never-alone
- ▶ AT: Attendant.
- ▶ A, B:
- ▶ P:

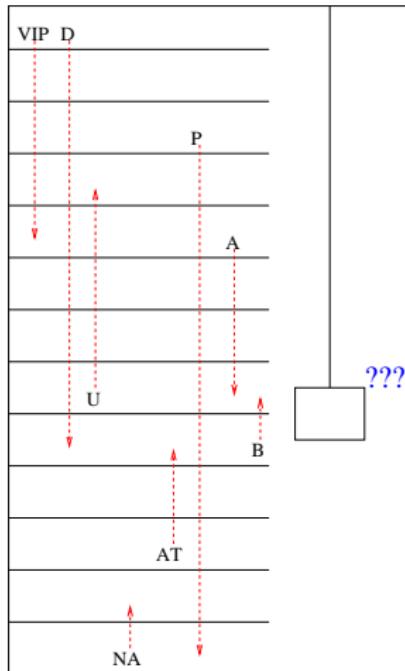


Miconic-10: A Real-World Example

- ▶ **Example 3.8.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D: Lift may only go *down* when inside; similar for U.
- ▶ NA: Never-alone
- ▶ AT: Attendant.
- ▶ A, B: Never together in the same elevator
- ▶ P:

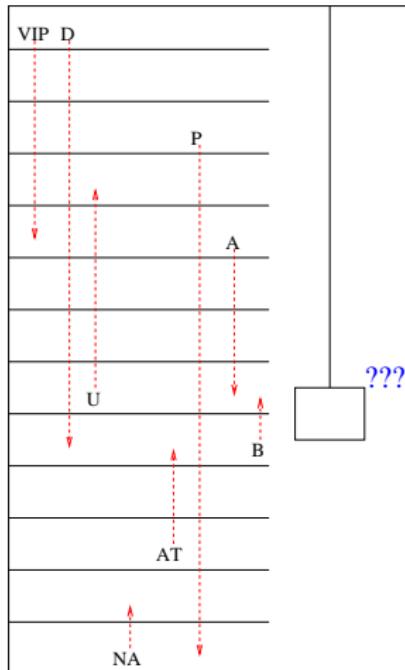


Miconic-10: A Real-World Example

- ▶ **Example 3.8.** 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



4 Partial Order Planning

Planning History, p.s.: Planning is Non-Trivial!

- ▶ **Example 4.1.** 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:

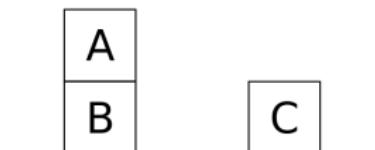
Planning History, p.s.: Planning is Non-Trivial!

- ▶ **Example 4.1.** 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 $\text{on}(A, B)$ by unstacking C , and moving A onto B , we achieve the first subgoal, but cannot achieve the second without undoing the first.



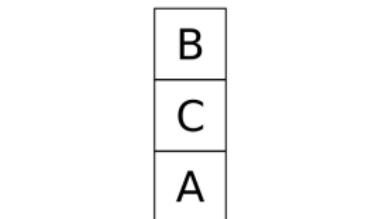
Planning History, p.s.: Planning is Non-Trivial!

- ▶ **Example 4.1.** 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 $\text{on}(A, B)$ by unstacking C , and moving A onto B , we achieve the first subgoal, but cannot achieve the second without undoing the first.
- ▶ If we pursue $\text{on}(B, C)$ by moving B onto C , we achieve the second subgoal, but cannot achieve the first without undoing the second.



Before we go into the details, let us try to understand the main ideas of [partial order planning](#).

- ▶ **Definition 4.2.** Any algorithm that can place two actions into a plan without specifying which comes first is called as **partial order planning**.

- ▶ **Definition 4.2.** 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. q
 - ▶ additional temporal constraints resolve subgoal interactions and induce a linear order.

- ▶ **Definition 4.2.** 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. q
 - ▶ additional temporal constraints resolve subgoal interactions and induce a linear order.
- ▶ **Advantages of partial order planning:**
 - ▶ problems can be decomposed \leadsto can work well with non-cooperative environments.
 - ▶ efficient by least-commitment strategy
 - ▶ causal links (edges) pinpoint unworkable subplans early.

Partially Ordered Plans

- ▶ **Definition 4.3.** Let $\Pi = \langle P, A, I, G \rangle$ 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 4.4.** Let be a partially ordered plan Π , 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 4.5.** A **precondition** is **achieved** iff it is the effect of an earlier **step** and no **possibly intervening** step undoes it.
- ▶ **Definition 4.6.** A partially ordered plan Π is called **complete** iff every precondition is achieved.
- ▶ **Definition 4.7.** **Partial order planning** is the process of computing **complete** and **acyclic** partially ordered plans for a given **planning task**.

- ▶ Notation: Write STRIPS actions into boxes with **preconditions** above and effects below.

▶ Example 4.8.

- ▶ Actions: $Buy(x)$

$At(p) \ Sells(p, x)$

- ▶ Preconditions: $At(p)$, $Sells(p, x)$

$Buy(x)$

- ▶ Effects: $Have(x)$

$Have(x)$

- ▶ 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.

- ▶ **Definition 4.9.** **Partial order planning** is search in the space of partial plans via the following operations:
 - ▶ **add link** from an existing action to an open precondition,
 - ▶ **add step** (an action with links to other **steps**) to fulfil an open condition,
 - ▶ **order** one **step** wrt. another to remove possible conflicts.
- ▶ **Idea:** Gradually move from incomplete/vague plans to complete, correct plans. Backtrack if an open condition is unachievable or if a conflict is unresolvable.

Example: Shopping for Bananas, Milk, and a Cordless Drill

Start

At(Home)

Sells(HWS,Drill)

Sells(SM,Milk)

Sells(SM,Ban.)

Have(Milk)

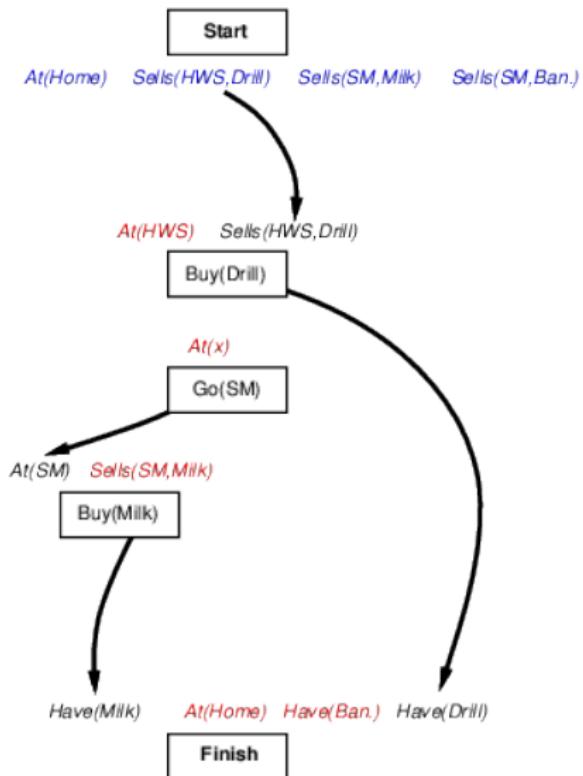
At(Home)

Have(Ban.)

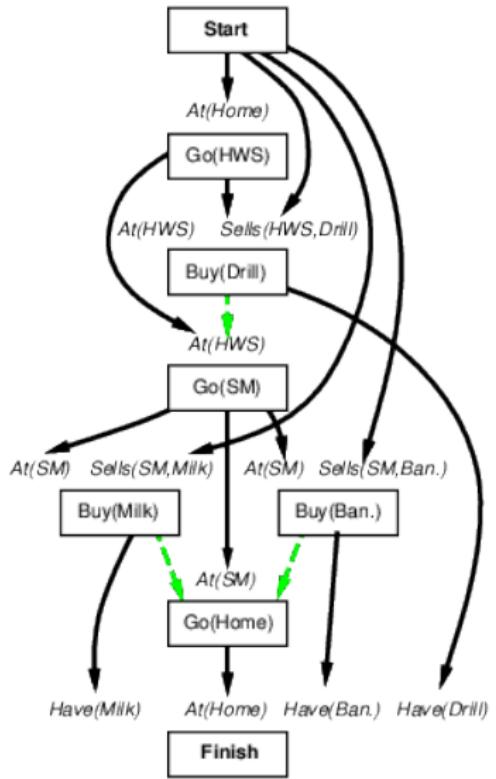
Have(Drill)

Finish

Example: Shopping for Bananas, Milk, and a Cordless Drill

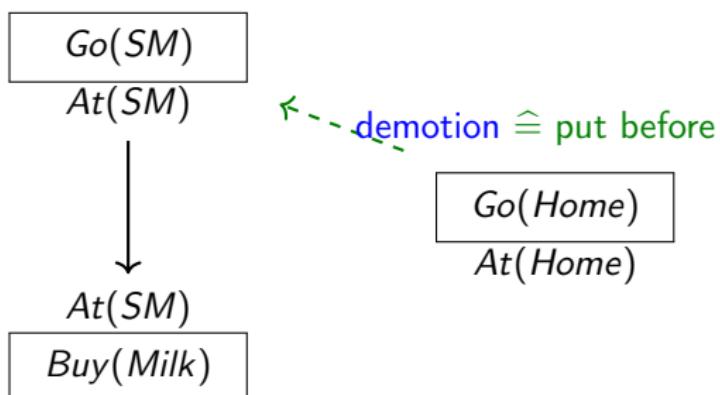


Example: Shopping for Bananas, Milk, and a Cordless Drill



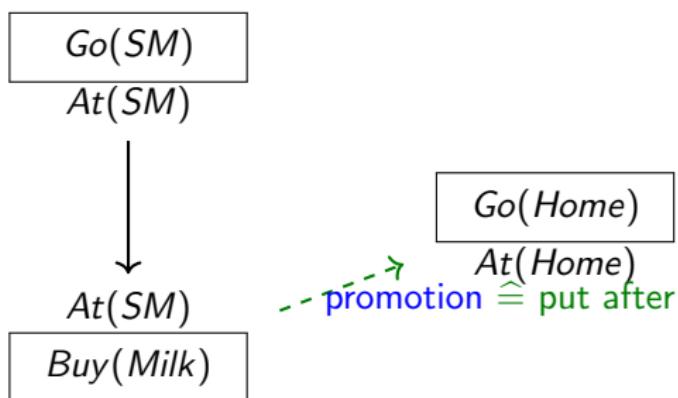
Clobbering and Promotion/Demotion

- ▶ **Definition 4.10.** 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 4.11.** 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 $C \leftarrow S$ to Π , or
 - ▶ **promotion:** add $T \leftarrow C$ to Π .
- ▶ **Example 4.12.** $Go(Home)$ clobbers $At(Supermarket)$:



Clobbering and Promotion/Demotion

- ▶ **Definition 4.10.** 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 4.11.** 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 4.12.** $Go(Home)$ clobbers $At(Supermarket)$:



POP algorithm sketch

- ▶ **Definition 4.13.** 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?(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 4.14.** The missing parts for the POP algorithm.

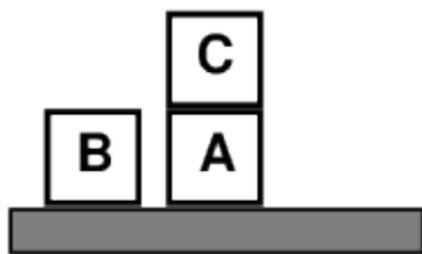
```
function Choose-Operator (plan, operators,  $S_{need}$ , c)
    choose a step  $S_{add}$  from operators or Steps(plan) that has c as an effect
    if there is no such step then fail
    add the ausal-link  $S_{add} \xrightarrow{c} S_{need}$  to Links(plan)
    add the temporal-constraint  $S_{add} \leftarrow S_{need}$  to Orderings(plan)
    if  $S_{add}$  is a newly added step from operators then
        add  $S_{add}$  to Steps(plan)
        add Start  $\leftarrow S_{add} \leftarrow$  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  $S_{threat} \leftarrow S_i$  to Orderings(plan)
            promotion: Add  $S_j \leftarrow S_{threat}$  to Orderings(plan)
    if not Consistent(plan) then fail
```

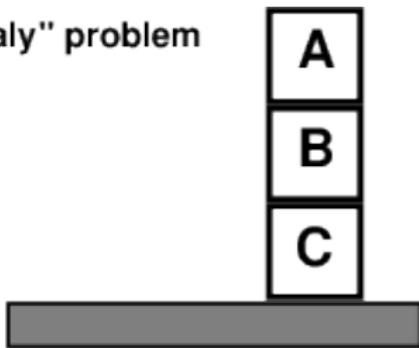
- ▶ Nondeterministic algorithm: backtracks at **choice points** on failure:
 - ▶ choice of S_{add} to achieve S_{need} ,
 - ▶ choice of demotion or promotion for clobberer,
 - ▶ selection of S_{need} is irrevocable.
- ▶ ***Observation 4.15.*** POP is sound, complete, and **systematic** – i.e. no repetition
- ▶ 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

"Sussman anomaly" problem



Start State



Goal State

$\text{Clear}(x) \text{ On}(x,z) \text{ Clear}(y)$

$\boxed{\text{PutOn}(x,y)}$

$\sim \text{On}(x,z) \sim \text{Clear}(y)$
 $\text{Clear}(z) \text{ On}(x,y)$

$\text{Clear}(x) \text{ On}(x,z)$

$\boxed{\text{PutOnTable}(x)}$

$\sim \text{On}(x,z) \text{ Clear}(z) \text{ On}(x,\text{Table})$

+ several inequality constraints

Example: Solving the Sussman Anomaly (contd.)

► Example 4.16. Solving the Sussman anomaly



$On(C, A)$ $On(A, T)$ $Cl(B)$ $On(B, T)$ $Cl(C)$

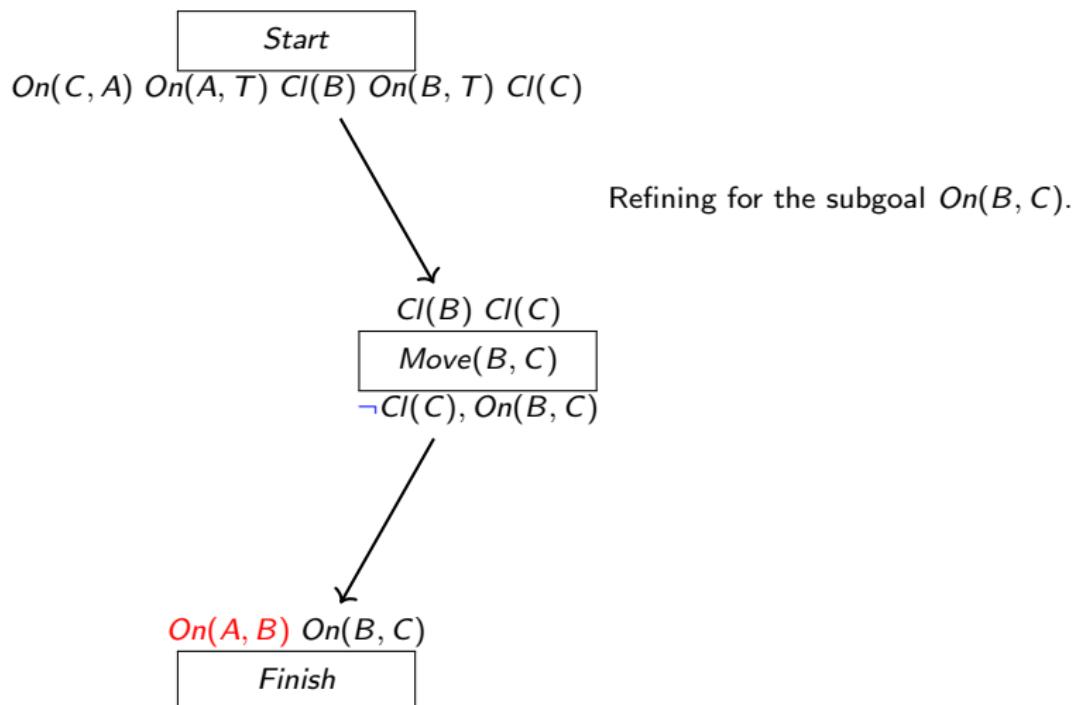
Initializing the partial order plan with Start and Finish.

$On(A, B)$ $On(B, C)$



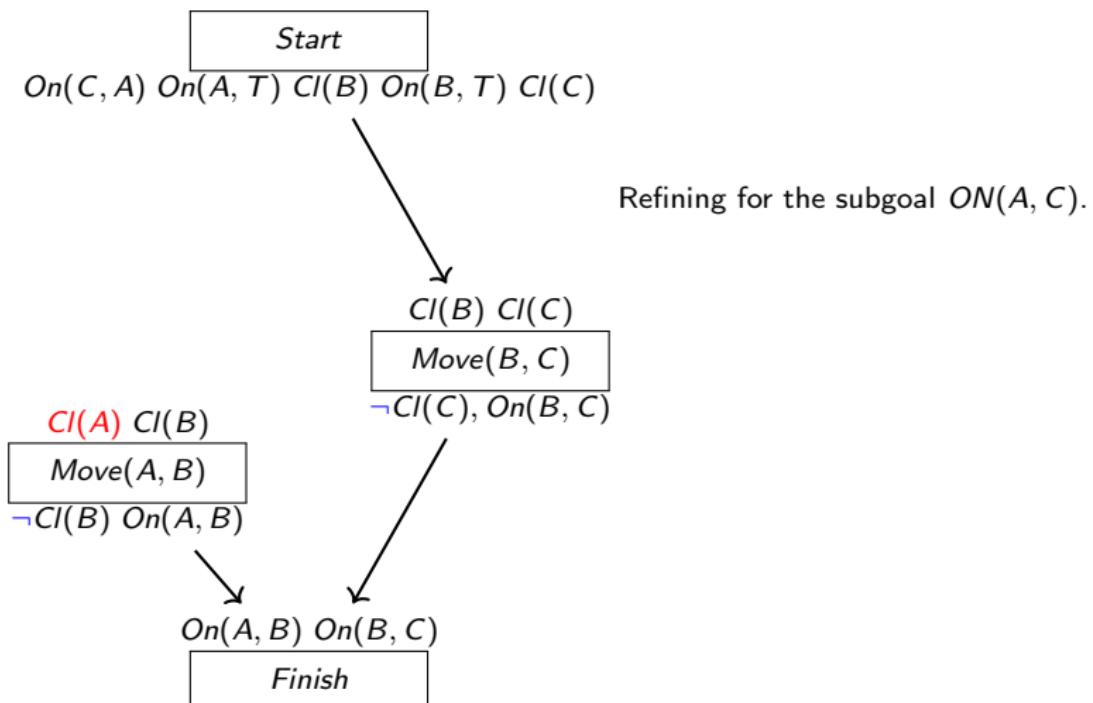
Example: Solving the Sussman Anomaly (contd.)

► Example 4.16. Solving the Sussman anomaly



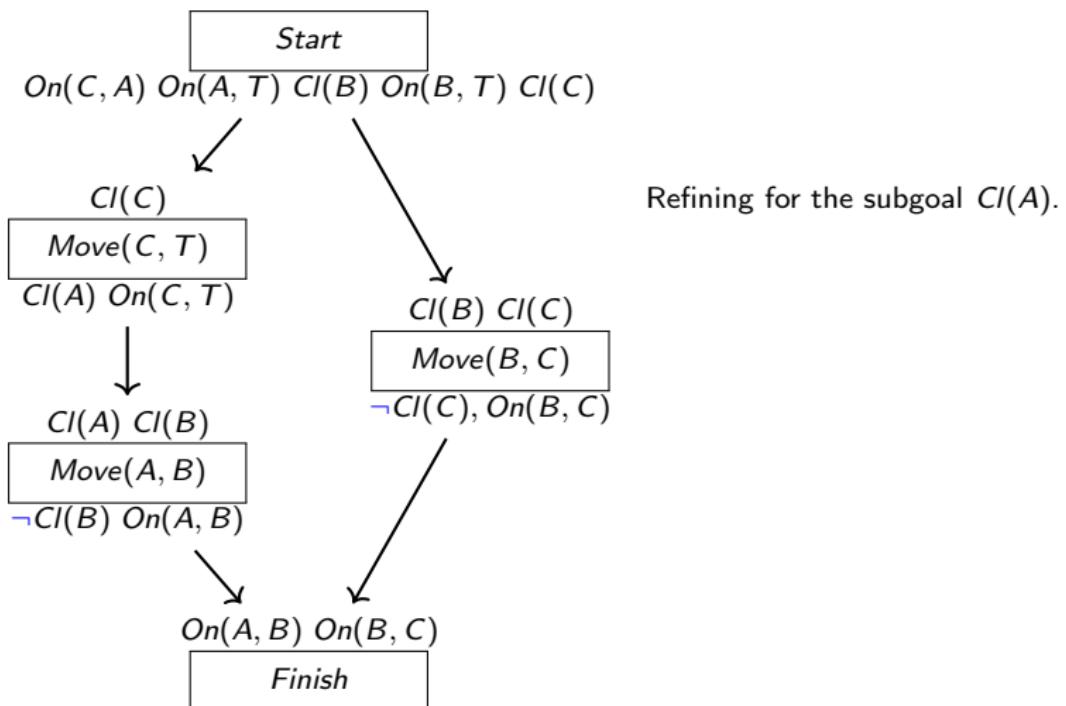
Example: Solving the Sussman Anomaly (contd.)

► Example 4.16. Solving the Sussman anomaly



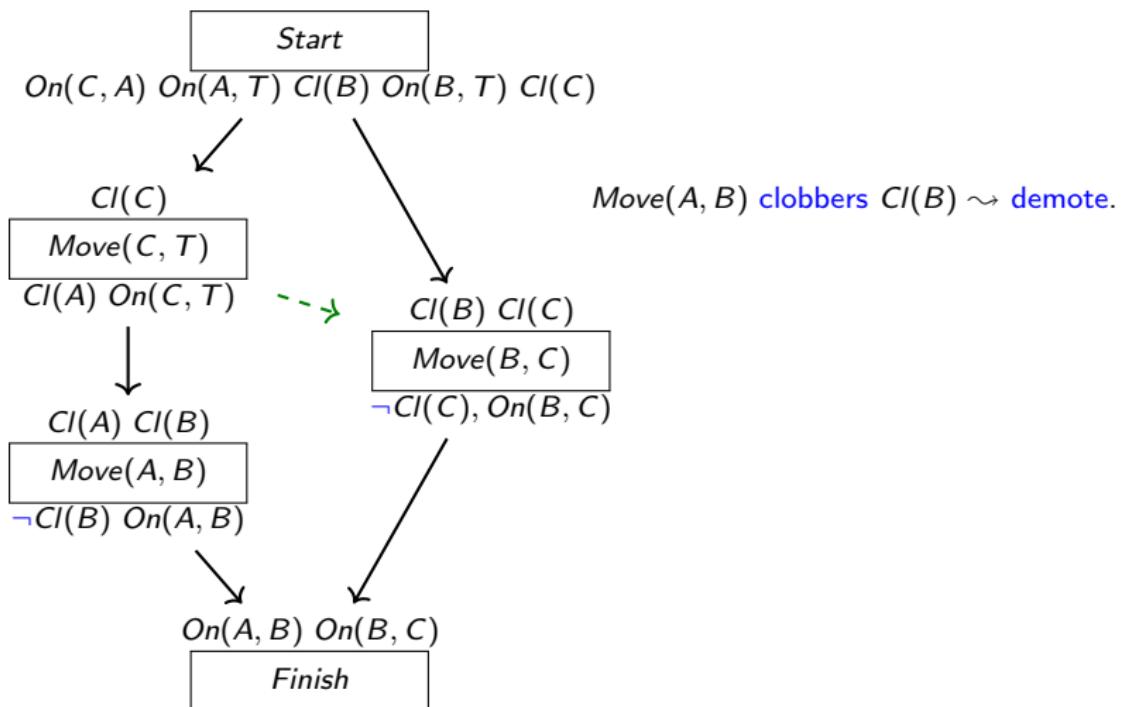
Example: Solving the Sussman Anomaly (contd.)

► Example 4.16. Solving the Sussman anomaly



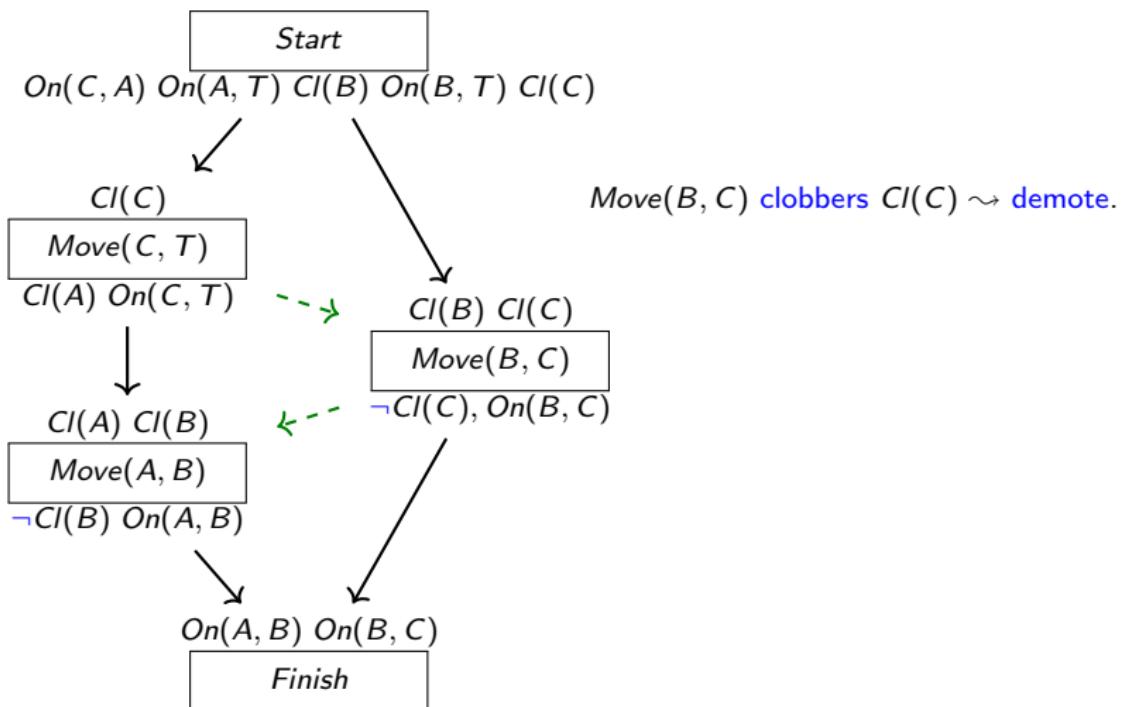
Example: Solving the Sussman Anomaly (contd.)

► Example 4.16. Solving the Sussman anomaly



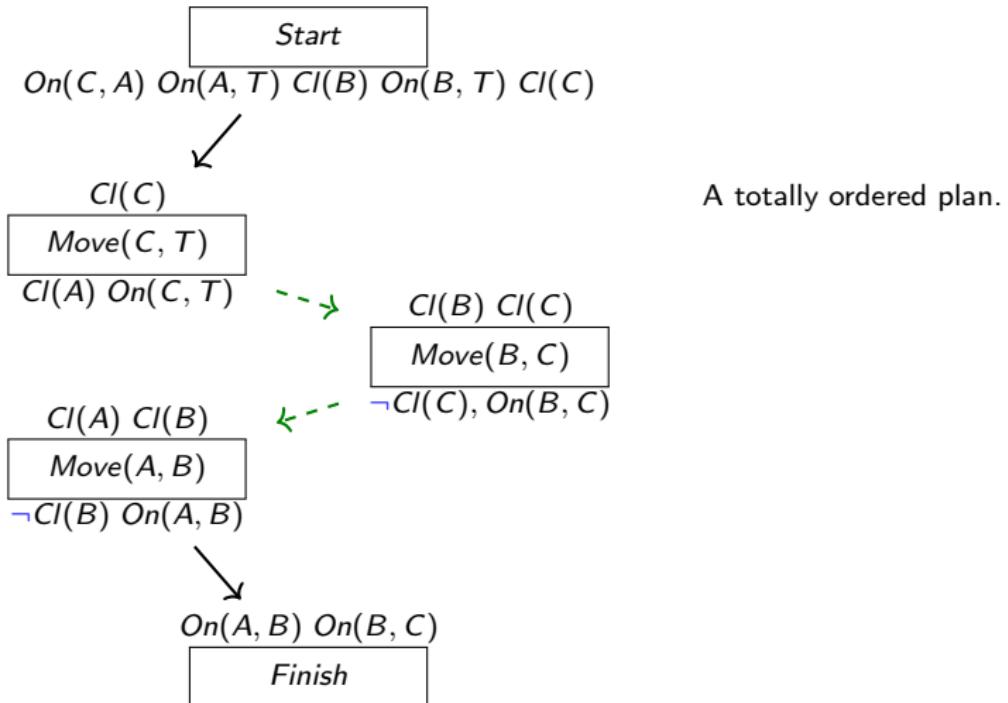
Example: Solving the Sussman Anomaly (contd.)

► Example 4.16. Solving the Sussman anomaly



Example: Solving the Sussman Anomaly (contd.)

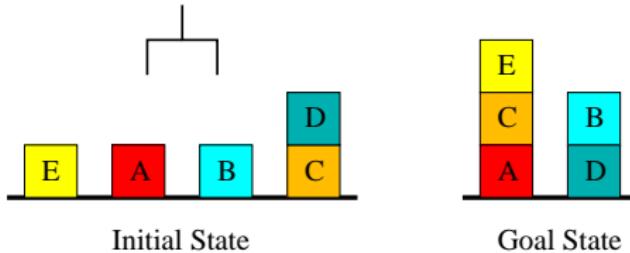
► Example 4.16. Solving the Sussman anomaly



5 The PDDL Language

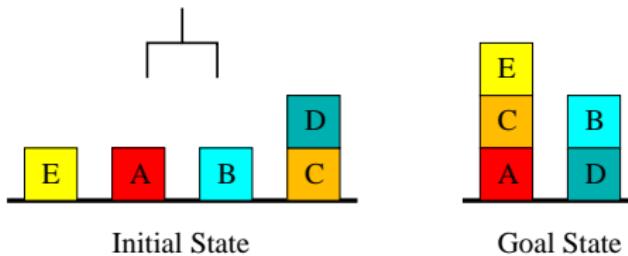
- ▶ **Definition 5.1.** The **Planning Domain Description Language (PDDL)** is a standardized representation language for planning **benchmarks** in various extensions of the **STRIPS** formalism.
- ▶ **Definition 5.2.** PDDL is not a propositional language
 - ▶ Representation is lifted, using **object 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 5.3.** A **PDDL planning task** comes in two pieces
 - ▶ The **problem file** gives the objects, the initial state, and the goal state.
 - ▶ The **domain file** gives the predicates and the actions.

The Blocksworld in PDDL: Domain File



```
(define (domain blocksworld)
  (:predicates (clear ?x) (holding ?x) (on ?x ?y)
               (on-table ?x) (arm-empty))
  (:action stack
    :parameters (?x ?y)
    :precondition (and (clear ?y) (holding ?x))
    :effect (and (arm-empty) (on ?x ?y)
             (not (clear ?y)) (not (holding ?x))))
    ...)
```

The Blocksworld in PDDL: Problem File



```
(define (problem bw-abcde)
  (:domain blocksworld)
  (:objects a b c d e)
  (:init (on-table a) (clear a)
        (on-table b) (clear b)
        (on-table e) (clear e)
        (on-table c) (on d c) (clear d)
        (arm-empty))
  (:goal (and (on e c) (on c a) (on b d))))
```

Miconic-ADL "Stop" Action Schema in PDDL

```
(:action stop
:parameters (?f - floor)
:precondition (and (lift-at ?f)
(imply
(exists
(?p - conflict-A)
(or (and (not (served ?p))
(origin ?p ?f))
(and (boarded ?p)
(not (destin ?p ?f)))))

(forall
(?q - conflict-B)
(and (or (destin ?q ?f)
(not (boarded ?q)))
(or (served ?q)
(not (origin ?q ?f))))))

(imply (exists
(?p - conflict-B)
(or (and (not (served ?p))
(origin ?p ?f))
(and (boarded ?p)
(not (destin ?p ?f)))))

(forall
(?q - conflict-A)
(and (or (destin ?q ?f)
(not (boarded ?q)))
(or (served ?q)
(not (origin ?q ?f))))))
```

```
(imply
(exists
(?p - never-alone)
(or (and (origin ?p ?f)
(not (served ?p)))
(and (boarded ?p)
(not (destin ?p ?f)))))

(exists
(?q - attendant)
(or (and (boarded ?q)
(not (destin ?q ?f)))
(and (not (served ?q))
(origin ?q ?f)))))

(forall
(?p - going-nonstop)
(imply (boarded ?p) (destin ?p ?f)))

(or (forall
(?p - vip) (served ?p))
(exists
(?p - vip)
(or (origin ?p ?f) (destin ?p ?f)))))

(forall
(?p - passenger)
(imply
(no-access ?p ?f) (not (boarded ?p)))))

)
```

6 Planning Complexity

- ▶ **Definition 6.1.** 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 effective in practice.
- ▶ **Definition 6.2.** Programs solving these problems are called (optimal) **planner**, **planning system**, or **planning tool**.

Decision Problems in (STRIPS) Planning

- ▶ **Definition 6.3.** By **PlanEx**, we denote the problem of deciding, given a STRIPS task Π , whether or not there exists a plan for Π .
- ▶ **PlanEx** corresponds to satisfying planning.
- ▶ **Definition 6.4.** By **PlanLen**, we denote the problem of deciding, given a STRIPS task Π and an integer B , whether or not there exists a plan for Π of length at most B .
- ▶ **PlanLen** corresponds to optimal planning.
- ▶ **Definition 6.5.** By **PolyPlanLen**, we denote the problem of deciding, given a STRIPS task Π and an integer B **bounded by a polynomial in the size of Π** , whether or not there exists a plan for Π of length at most B .
- ▶ **PolyPlanLen** corresponds to optimal planning with “small” **plans**.
- ▶ Example of a planning domain with exponentially long **plans**?
- ▶ Towers of Hanoi.

Complexity of PlanEx (after [Byl94])

- ▶ **Lemma 6.6.** PlanEx is **PSPACE**-hard. (At least as hard as any other problem contained in **PSPACE**)
- ▶ **Proof:** Given a Turing machine with space bounded by polynomial $p(|w|)$, we can in polynomial time (in the size of the machine) generate an equivalent **STRIPS tasks**. Say the possible symbols in tape cells are x_1, \dots, x_m and the internal states are s_1, \dots, s_n , accepting state s_{acc} .
 1. The contents of the tape cells:
 $in(1, x_1), \dots, in(p(|w|), x_1), \dots, in(1, x_m), \dots, in(p(|w|), x_m)$.
 2. The position of the R/W head: $at(1), \dots, at(p(|w|))$.
 3. The internal state of the machine: $state(s_1), \dots, state(s_n)$.
 4. Transitions rules \mapsto **STRIPS actions**; accepting state \mapsto **STRIPS goal** $\{state(s_{acc})\}$; initial state obvious.
 5. This reduction to **STRIPS** runs in polynomial-time because we need only polynomially many facts.



- ▶ **Lemma 6.7.** PlanEx is in **PSPACE** (At most as hard as any other problem contained in PSPACE)
- ▶ Proof: As **PSPACE = NPSPACE** we show that PlanEx is in **NPSPACE**

1. 1. $s := I; l := 0;$
2. Guess an applicable action a , compute the outcome state s' , set $I := l + 1$;
3. If s' contains the goal then succeed;
4. If $l \geq 2^{|P|}$ then fail else goto 2;

□

- ▶ Remembering the actual action *sequence* would take exponential space in case of exponentially long **plans** (cf. slide 573). But, to decide PlanEx, we only need to remember its length.
- ▶ **Corollary 6.8 (Complexity of PlanEx).** PlanEx is **PSPACE**-complete. (Immediate from previous two lemmata)

Complexity of PlanLen (after [Byl94])

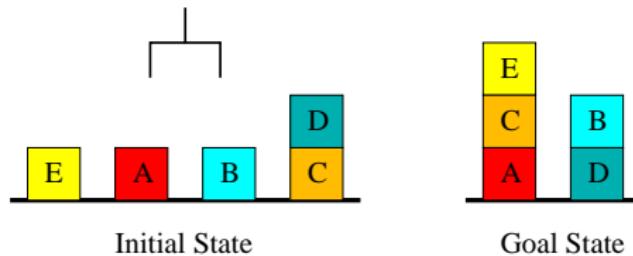
- ▶ PlanLen isn't any easier than PlanEx
- ▶ **Corollary 6.9.** PlanLen is **PSPACE**-complete.
- ▶ Proof:
 1. (**Membership**) Same as before but failing at $l \geq B$.
 2. (**Hardness**) Setting $B := 2^{|P|}$, PlanLen answers PlanEx: If a **plan** exists, then there exists a **plan** that traverses each possible state at most once.

□

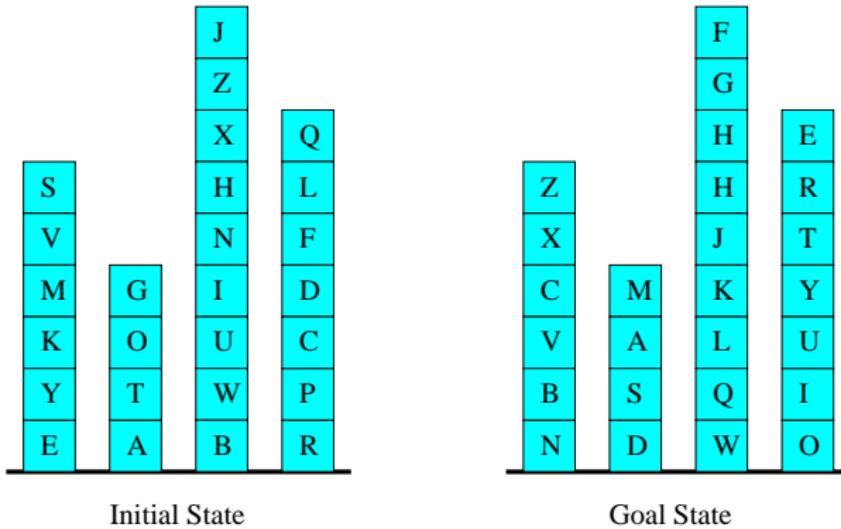
- ▶ PolyPlanLen is easier than PlanEx:
- ▶ **Theorem 6.10.** PolyPlanLen is **NP**-complete.
- ▶ Proof:
 1. (**Membership**) Guess B actions and check whether they form a **plan**. This runs in polynomial time because B is polynomially bounded.
 2. (**Hardness**) E.g., by reduction from SAT.
- ▶ Bounding **plan** length does not help in the general case as we can set the bound to a trivial (exponential) upper bound on **plan** length. If we restrict **plan** length to be "short" (polynomial), planning becomes easier.

□

The Blocksworld is Hard?



The Blocksworld is Hard!

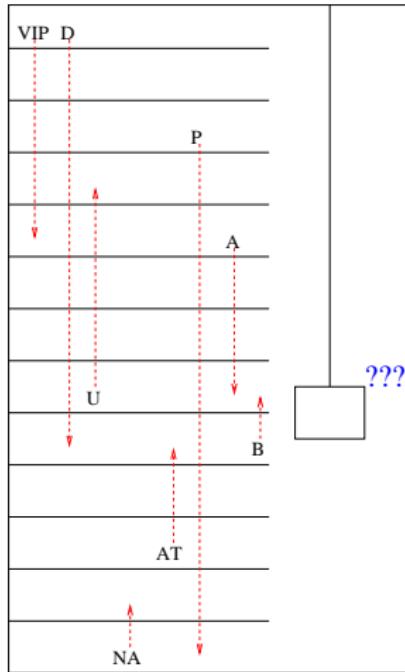


Miconic-10: A Real-World Example

- ▶ **Example 6.11.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP:
- ▶ D:
- ▶ NA:
- ▶ AT:
- ▶ A, B:
- ▶ P:

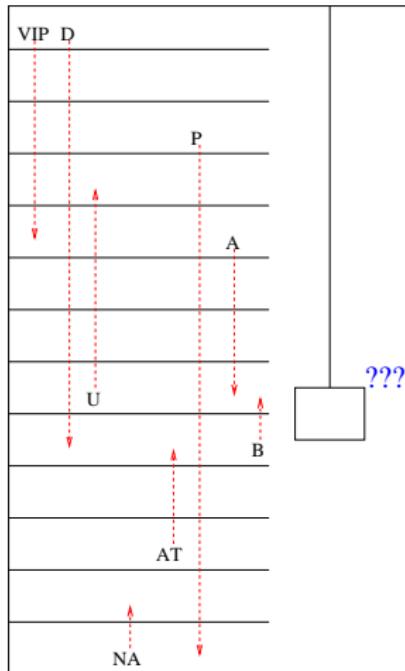


Miconic-10: A Real-World Example

- ▶ **Example 6.11.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D:
- ▶ NA:
- ▶ AT:
- ▶ A, B:
- ▶ P:

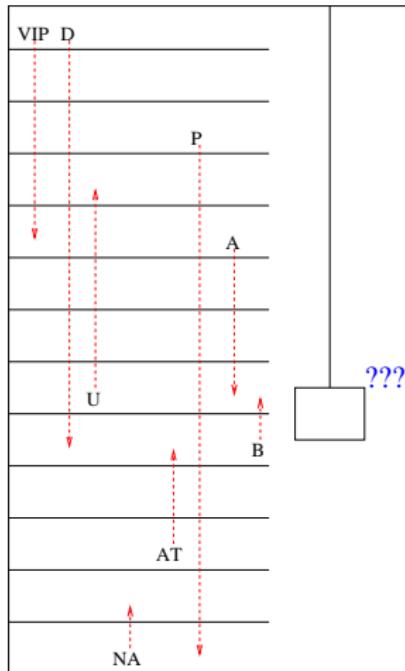


Miconic-10: A Real-World Example

- ▶ **Example 6.11.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D: Lift may only go *down* when inside; similar for U.
- ▶ NA:
- ▶ AT:
- ▶ A, B:
- ▶ P:

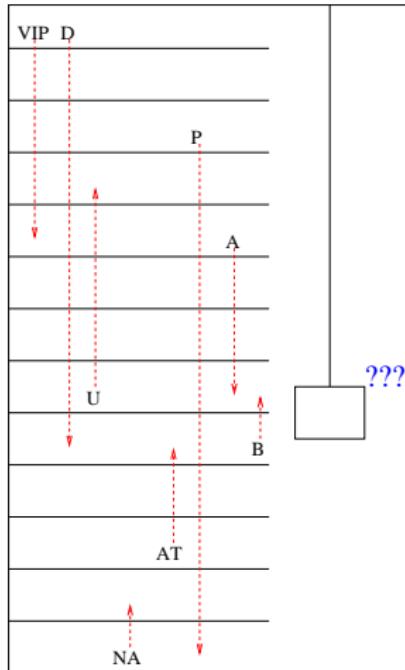


Miconic-10: A Real-World Example

- ▶ **Example 6.11.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D: Lift may only go *down* when inside; similar for U.
- ▶ NA: Never-alone
- ▶ AT:
- ▶ A, B:
- ▶ P:

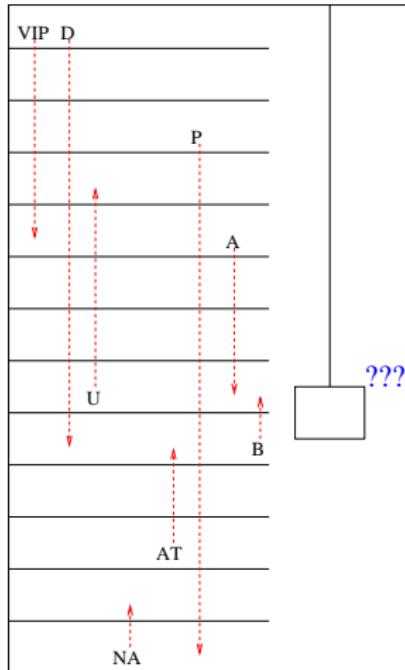


Miconic-10: A Real-World Example

- ▶ **Example 6.11.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D: Lift may only go *down* when inside; similar for U.
- ▶ NA: Never-alone
- ▶ AT: Attendant.
- ▶ A, B:
- ▶ P:

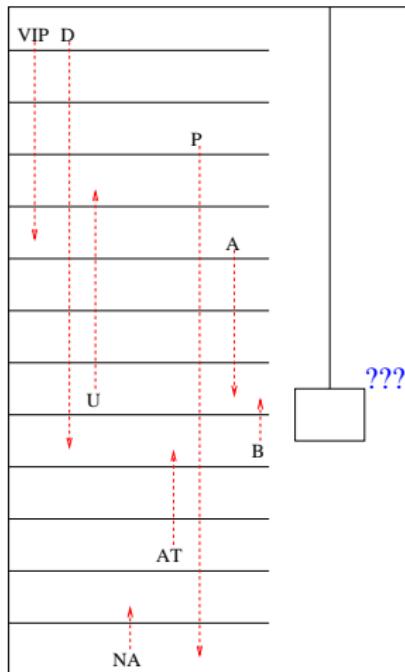


Miconic-10: A Real-World Example

- ▶ **Example 6.11.** Elevator control as a planning problem; details at [KS00]
Specify mobility needs before boarding, let a planner schedule/otimize trips



- ▶ VIP: Served first.
- ▶ D: Lift may only go *down* when inside; similar for U.
- ▶ NA: Never-alone
- ▶ AT: Attendant.
- ▶ A, B: Never together in the same elevator
- ▶ P:

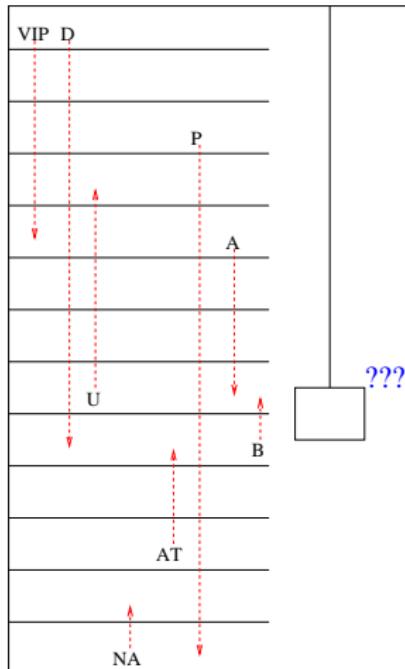


Miconic-10: A Real-World Example

- ▶ **Example 6.11.** 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



7 Conclusion

- ▶ General problem solving attempts to develop solvers that perform well across a large class of problems.
- ▶ Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems. (Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.)
- ▶ Heuristic search planning has dominated the [International Planning Competition \(IPC\)](#). We focus on it here.
- ▶ **STRIPS** is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines actions in terms of precondition, add list, and delete list.
- ▶ PDDL is the de-facto standard language for describing planning problems.
- ▶ Plan existence (bounded or not) is **PSPACE**-complete to decide for **STRIPS**. If we bound **plans** polynomially, we get down to **NP**-completeness.

Chapter 17 Planning II: Algorithms

1 Introduction

- ▶ : 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).
- ▶ **This Chapter:** How to automatically generate a heuristic function, given planning language input?
 - ▶ Focussing on heuristic search as the solution method, this is the main question that needs to be answered.

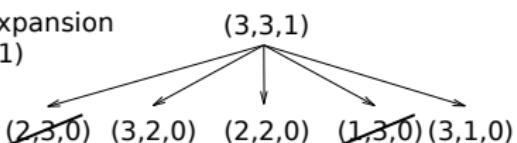
Reminder: Search

- ▶ Starting at initial state, produce all successor states step by step:

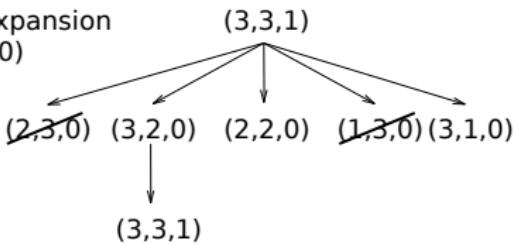
(a) initial state

(3,3,1)

(b) after expansion
of (3,3,1)

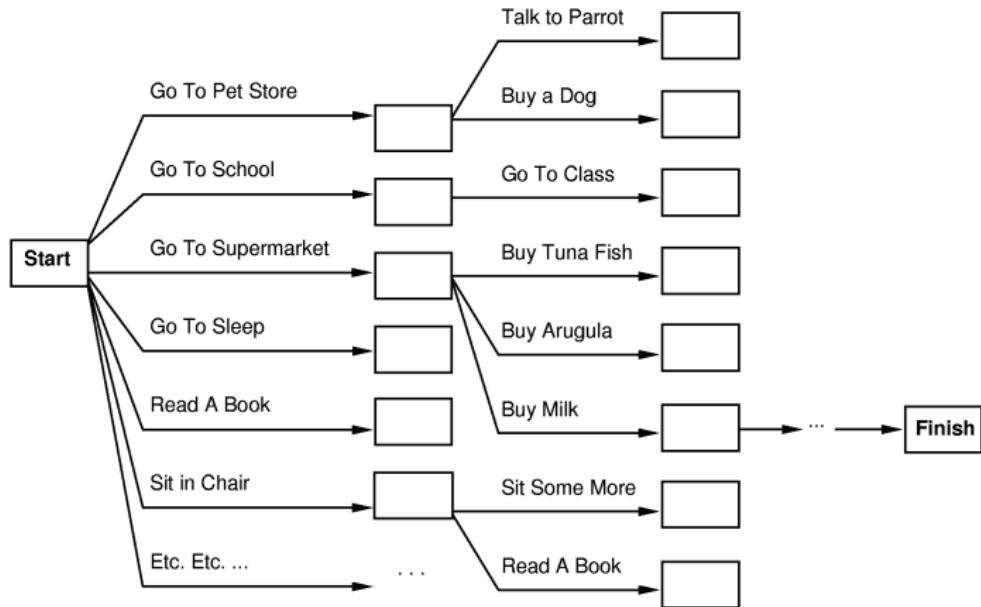


(c) after expansion
of (3,2,0)



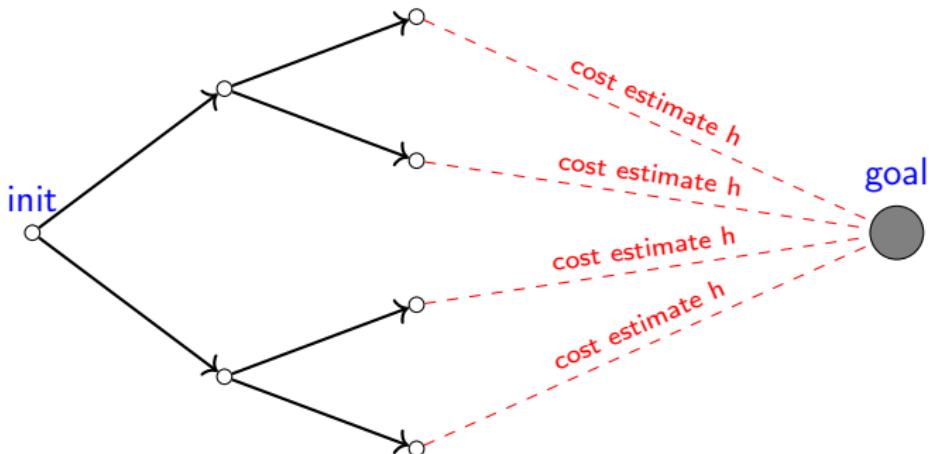
In planning, this is referred to as **forward search**, or **forward state-space search**.

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; search prefers to expand states s with small $h(s)$.
- ▶ Live Demo vs. Breadth-First Search:

<http://qiao.github.io/PathFinding.js/visual/>

- ▶ **Definition 1.1.** Let Π be a STRIPS task with states S . A **heuristic function**, short **heuristic**, for Π is a function $h: S \rightarrow \mathbb{N} \cup \{\infty\}$ so that $h(s) = 0$ whenever s is a goal state.
- ▶ Exactly like our definition from . Except, because we assume unit costs here, we use \mathbb{N} instead of \mathbb{R}^+ .
- ▶ **Definition 1.2.** Let Π be a STRIPS task with states S . The **perfect heuristic** h^* assigns every $s \in S$ the length of a shortest path from s to a goal state, or ∞ if no such path exists. A heuristic function h for Π is **admissible** if, for all $s \in S$, we have $h(s) \leq h^*(s)$.
- ▶ Exactly like our definition from , except for path *length* instead of path *cost* (cf. above).
- ▶ In all cases, we attempt to approximate $h^*(s)$, the length of an optimal plan for s . Some algorithms guarantee to lower-bound $h^*(s)$.

Our (Refined) Agenda for This Chapter

- ▶ **How to Relax:** How to relax a problem?
 - ▶ Basic principle for generating heuristic functions.
- ▶ **The Delete Relaxation:** How to relax a planning problem?
 - ▶ The delete relaxation is the most successful method for the *automatic* generation of heuristic functions. It is a key ingredient to almost all **IPC** winners of the last decade. It relaxes **STRIPS tasks** by ignoring the delete lists.
- ▶ **The h^+ Heuristic:** What is the resulting heuristic function?
 - ▶ h^+ is the “ideal” delete relaxation heuristic.
- ▶ **Approximating h^+ :** How to actually compute a heuristic?
 - ▶ Turns out that, in practice, we must approximate h^+ .

2 How to Relax in Planning

Reminder: Heuristic Functions from Relaxed Problems



- ▶ Problem Π: Find a route from Saarbrücken to Edinburgh.

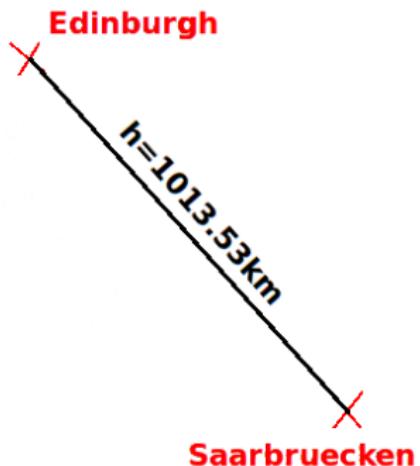
Reminder: Heuristic Functions from Relaxed Problems

Edinburgh
X

X
Saarbruecken

- Relaxed Problem Π' : Throw away the map.

Reminder: Heuristic Functions from Relaxed Problems

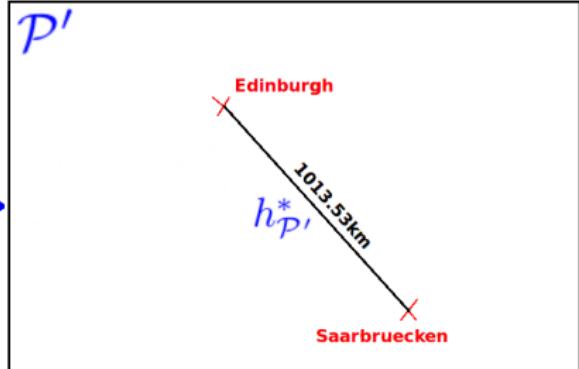


- ▶ Heuristic function h : Straight line distance.

Relaxation in Route-Finding



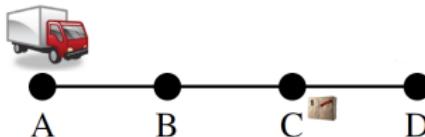
\mathcal{R}



- ▶ Problem class \mathcal{P} : Route finding.
- ▶ Perfect heuristic $h^*_{\mathcal{P}}$ for \mathcal{P} : Length of a shortest route.
- ▶ Simpler problem class \mathcal{P}' : Route finding on an empty map.
- ▶ Perfect heuristic $h^*_{\mathcal{P}'}$ for \mathcal{P}' : Straight-line distance.
- ▶ Transformation \mathcal{R} : Throw away the map.

How to Relax in Planning? (A Reminder!)

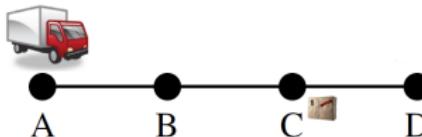
► Example 2.1 (Logistics).



- ▶ 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 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: “ $\text{truck}(x) \Rightarrow \text{truck}(y), \neg\text{truck}(x)$ ”.
 - ▶ $\text{load}(x)$: “ $\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T), \neg\text{pack}(x)$ ”.
 - ▶ $\text{unload}(x)$: “ $\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x), \neg\text{pack}(T)$ ”.

How to Relax in Planning? (A Reminder!)

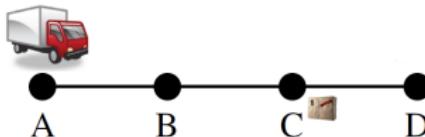
► Example 2.1 (Logistics).



- ▶ 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 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 2.2 (“Only-Adds” Relaxation). Drop the preconditions and deletes.
 - ▶ “ $\text{drive}(x, y)$: $\Rightarrow \text{truck}(y)$ ”;
 - ▶ “ $\text{load}(x)$: $\Rightarrow \text{pack}(T)$ ”;
 - ▶ “ $\text{unload}(x)$: $\Rightarrow \text{pack}(x)$ ”.
- ▶ Heuristic value for I is?

How to Relax in Planning? (A Reminder!)

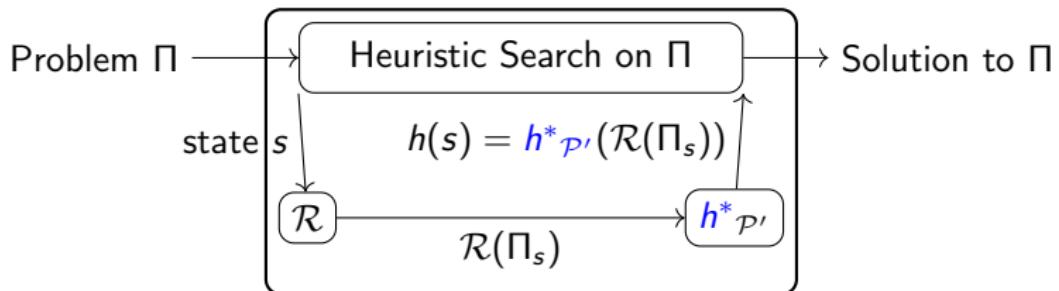
► Example 2.1 (Logistics).



- ▶ 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 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 2.2 (“Only-Adds” Relaxation). Drop the preconditions and deletes.
 - ▶ “ $\text{drive}(x, y)$: $\Rightarrow \text{truck}(y)$ ”;
 - ▶ “ $\text{load}(x)$: $\Rightarrow \text{pack}(T)$ ”;
 - ▶ “ $\text{unload}(x)$: $\Rightarrow \text{pack}(x)$ ”.
- ▶ Heuristic value for I is?
- ▶ $h^R(I) = 1$: A plan for the relaxed task is $\langle \text{unload}(D) \rangle$.

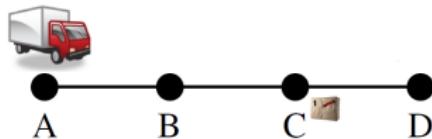
How to Relax During Search: Overview

- ▶ **Attention:** Search uses the real (un-relaxed) Π . The relaxation is applied (e.g., in Only-Adds, the simplified actions are used) **only within the call to $h(s)$!!!**



- ▶ Here, Π_s is Π with initial state replaced by s , i.e., $\Pi = \langle P, A, I, G \rangle$ changed to $\Pi^s := \langle P, A, s, G \rangle$: The task of finding a plan for search state s .
- ▶ A common student mistake is to instead apply the relaxation once to the whole problem, then doing the whole search “within the relaxation”.
- ▶ The next slide illustrates the correct search process in detail.

How to Relax During Search: Only-Adds



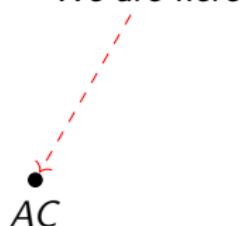
Real problem:

- ▶ Initial state I : AC ; goal G : AD .
- ▶ Actions A : `pre`, `add`, `del`.
- ▶ $drXY$, loX , ulX .

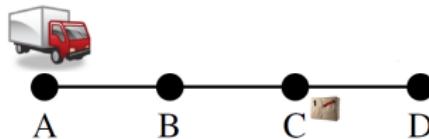
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



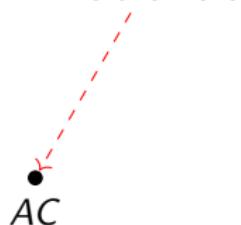
Relaxed problem:

- ▶ State $s: AC$; goal $G: AD$.
- ▶ Actions A : add.
- ▶ $h^R(s) =$

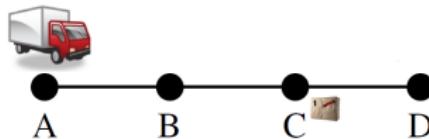
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



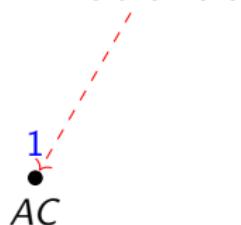
Relaxed problem:

- ▶ State s : AC ; goal G : AD .
- ▶ Actions A : **add**.
- ▶ $h^R(s) = 1$: $\langle uID \rangle$.

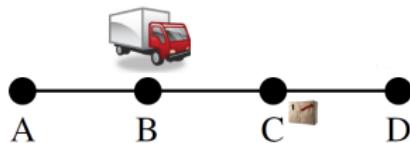
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



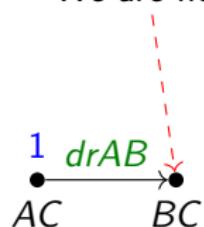
Real problem:

- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions A : `pre`, `add`, `del`.
- ▶ $AC \xrightarrow{drAB} BC$.

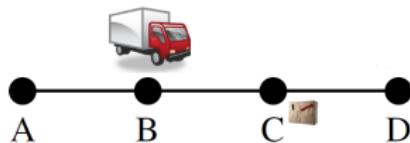
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



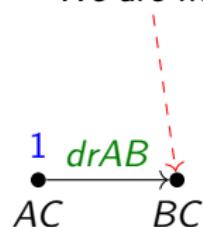
Relaxed problem:

- ▶ State s : BC ; goal G : AD .
- ▶ Actions A : add.
- ▶ $h^R(s) =$

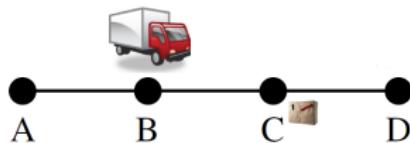
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



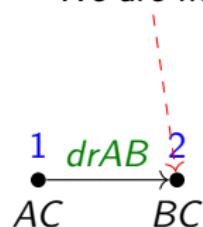
Relaxed problem:

- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions $A: \text{add}$.
- ▶ $h^R(s) = 2: \langle drBA, uID \rangle$.

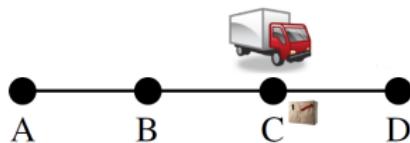
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



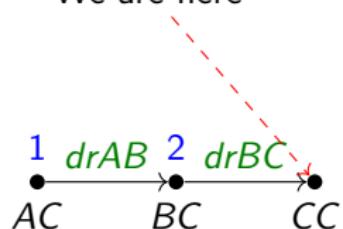
Real problem:

- ▶ State $s: CC$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ $BC \xrightarrow{drBC} CC$.

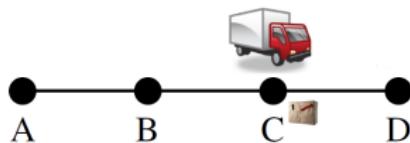
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



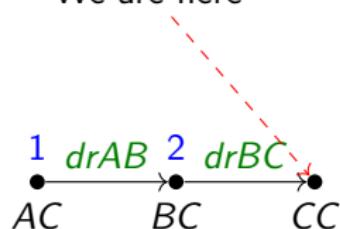
Relaxed problem:

- ▶ State s : CC ; goal G : AD .
- ▶ Actions A : **add**.
- ▶ $h^R(s) =$

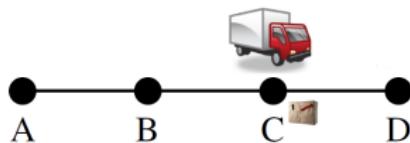
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



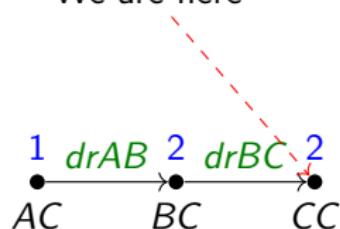
Relaxed problem:

- ▶ State s : CC; goal G : AD.
- ▶ Actions A : add.
- ▶ $h^R(s) = 2$: $\langle drBA, uID \rangle$.

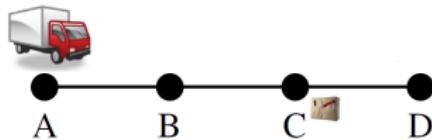
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



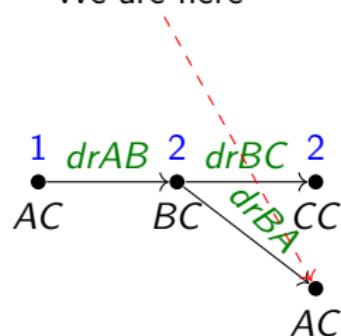
Real problem:

- ▶ State $s: AC$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ $BC \xrightarrow{drBA} AC$.

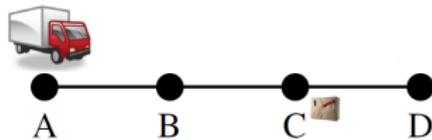
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



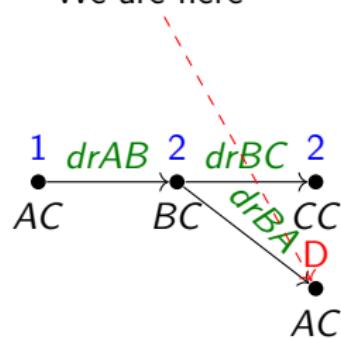
Real problem:

- ▶ State $s: AC$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ Duplicate state, prune.

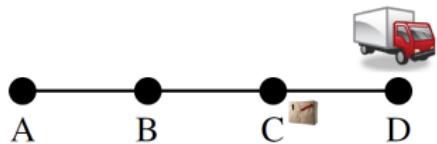
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



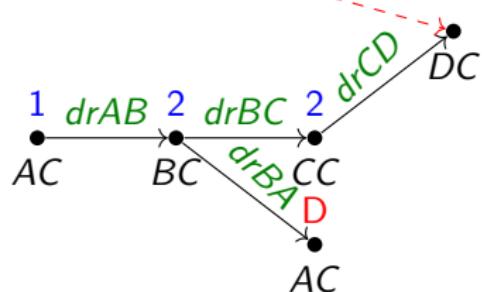
Real problem:

- ▶ State $s: DC$; goal $G: AD$.
- ▶ Actions A : pre, add, del.
- ▶ $CC \xrightarrow{drCD} DC$.

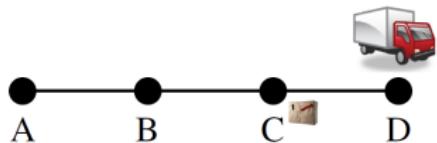
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



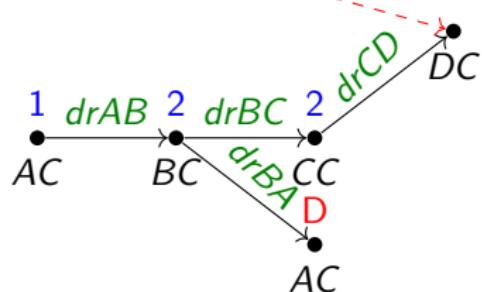
Relaxed problem:

- ▶ State $s: DC$; goal $G: AD$.
- ▶ Actions $A: \text{add}$.
- ▶ $h^R(s) =$

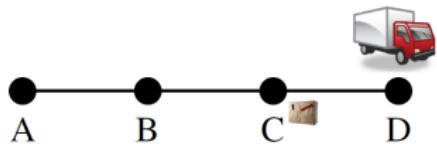
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



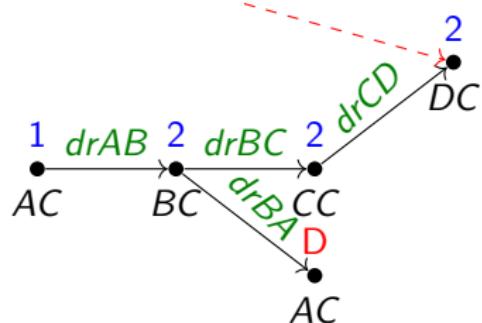
Relaxed problem:

- ▶ State $s: DC$; goal $G: AD$.
- ▶ Actions $A: \text{add}$.
- ▶ $h^R(s) = 2: \langle drBA, uID \rangle$.

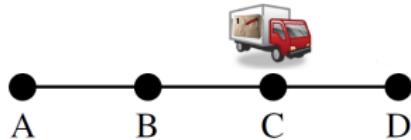
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



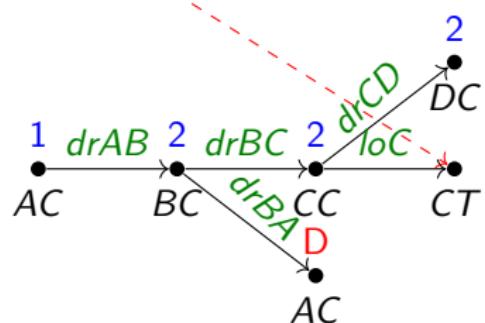
Real problem:

- ▶ State $s: CT$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ $CC \xrightarrow{\text{loC}} CT$.

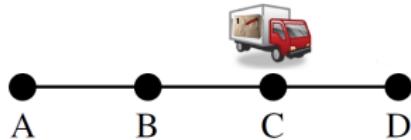
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



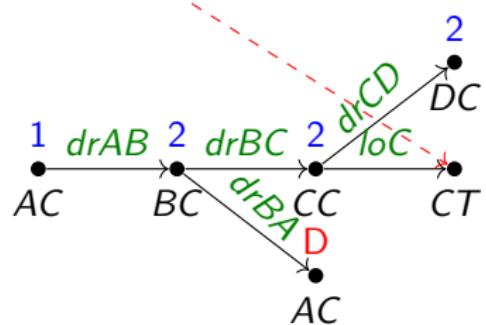
Relaxed problem:

- ▶ State $s: CT$; goal $G: AD$.
- ▶ Actions $A: \text{add}$.
- ▶ $h^R(s) =$

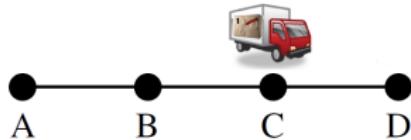
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



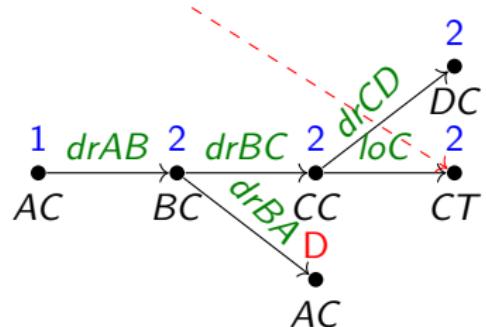
Relaxed problem:

- ▶ State $s: CT$; goal $G: AD$.
- ▶ Actions $A: \text{add}$.
- ▶ $h^R(s) = 2: \langle drBA, ulD \rangle$.

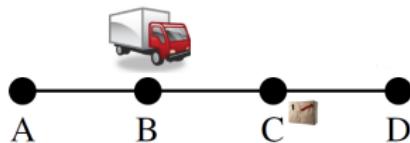
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



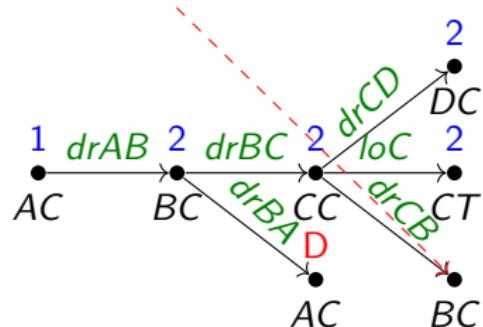
Real problem:

- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ $CC \xrightarrow{drCB} BC$.

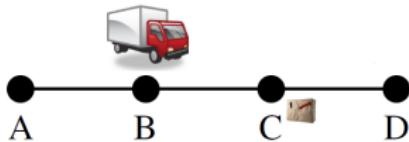
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



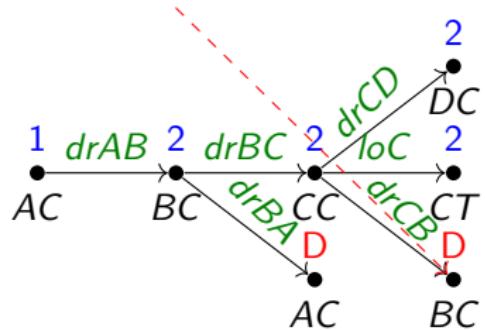
Real problem:

- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▶ Duplicate state, prune.

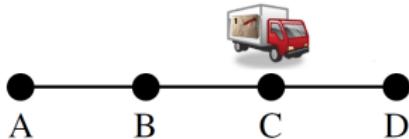
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds



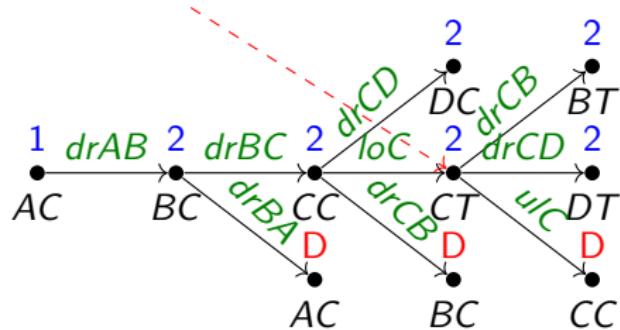
Real problem:

- ▶ State $s: CT$; goal $G: AD$.
- ▶ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▶ Successors: BT, DT, CC .

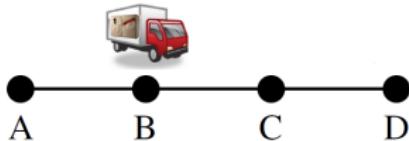
▶ Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Only-Adds

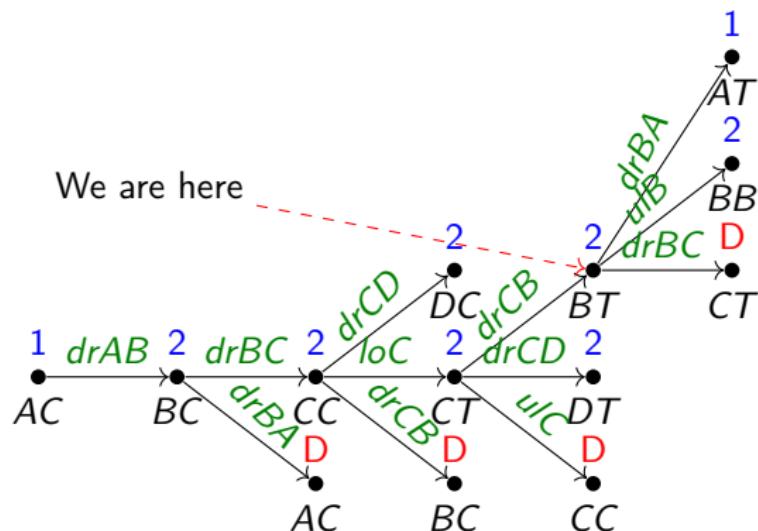


Real problem:

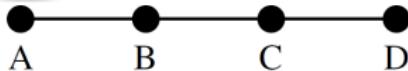
- ▶ State $s: BT$; goal $G: AD$.
- ▶ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▶ Successors: AT, BB, CT .

▶ Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

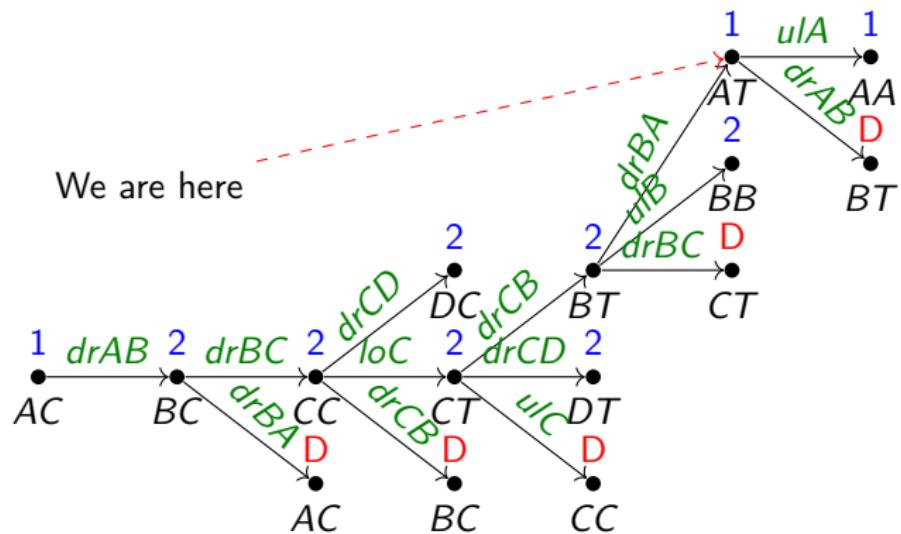


Real problem:

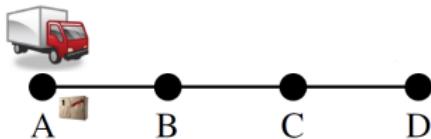
- ▶ State s : AT ; goal G : AD .
- ▶ Actions A : `pre`, `add`, `del`.
- ▶ Successors: AA , BT .

▶ Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

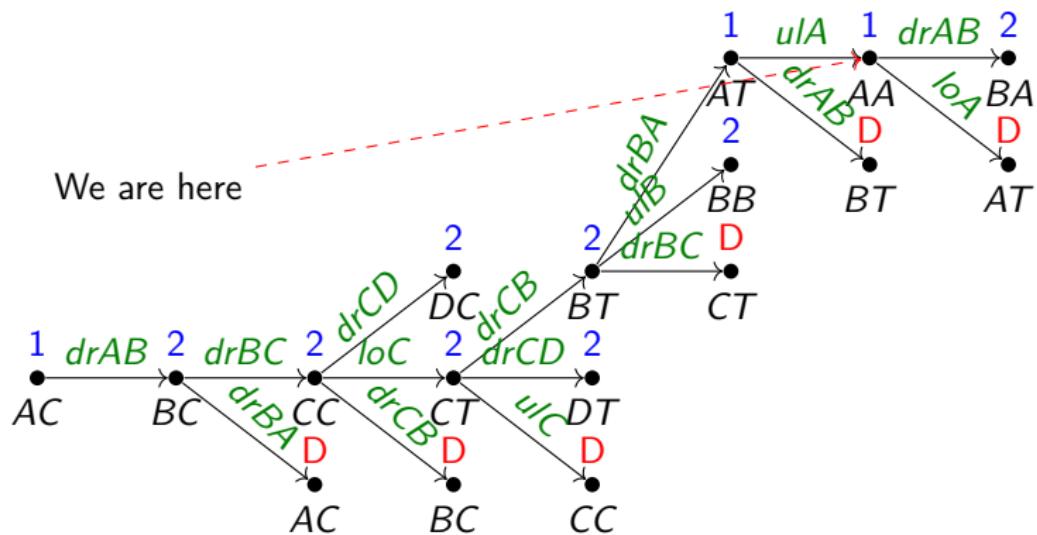


Real problem:

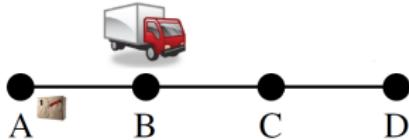
- ▶ State s : AA ; goal G : AD .
- ▶ Actions A : `pre`, `add`, `del`.
- ▶ Successors: BA , AT .

▶ Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

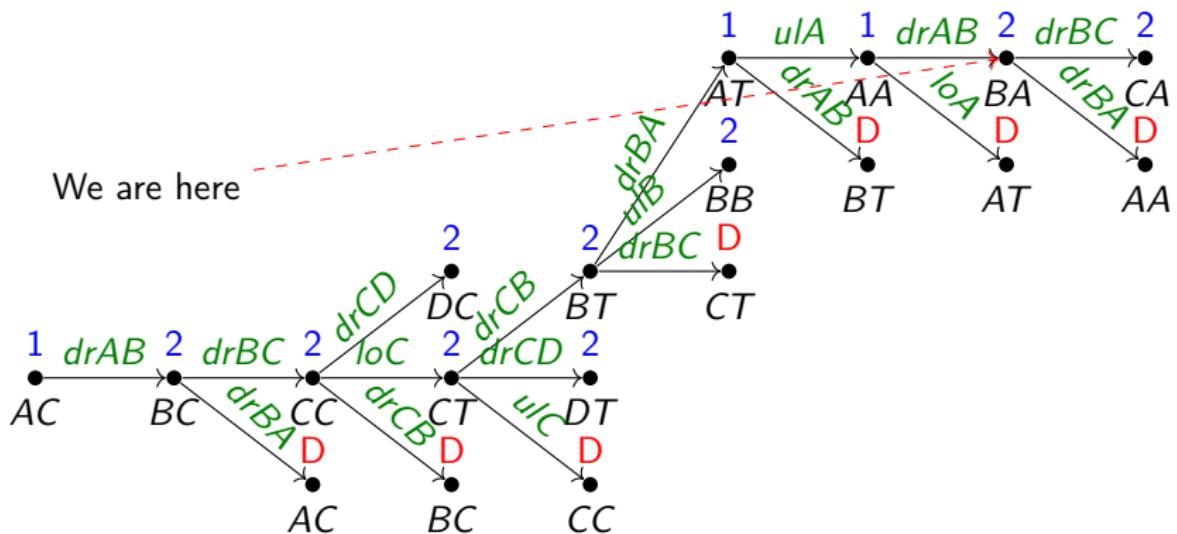


Real problem:

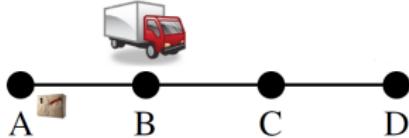
- ▶ State $s: BA$; goal $G: AD$.
- ▶ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▶ Successors: CA, AA .

▶ Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

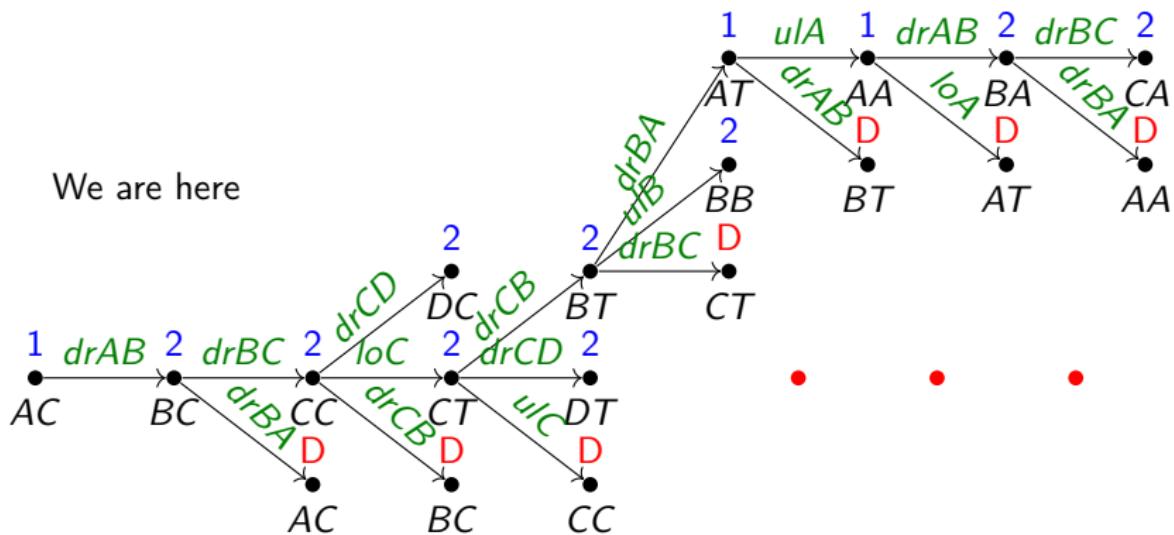


- ▶ State s : BA ; goal G : AD .
 - ▶ Actions A : [pre](#), [add](#), [del](#).
 - ▶ Successors: CA , AA .

► Greedy best-first search:

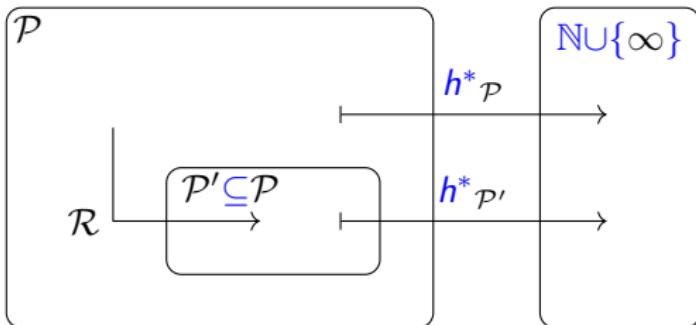
(tie-breaking: alphabetic)

We are here



Only-Adds is a “Native” Relaxation

- ▶ **Definition 2.3 (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.

3 The Delete Relaxation

How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:
“When the world changes, its previous state remains true as well.”

How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

“When the world changes, its previous state remains true as well.”

Real world: (before)



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

“When the world changes, its previous state remains true as well.”

Real world: (after)



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

“When the world changes, its previous state remains true as well.”

Relaxed world: (before)



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

“When the world changes, its previous state remains true as well.”

Relaxed world: (after)



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

"When the world changes, its previous state remains true as well."

Real world: (before)



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

"When the world changes, its previous state remains true as well."

Real world: (after)



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

“When the world changes, its previous state remains true as well.”
Relaxed world: (before)



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

"When the world changes, its previous state remains true as well."

Relaxed world: (after)



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

“When the world changes, its previous state remains true as well.”

Real world:



How the Delete Relaxation Changes the World

- ▶ Relaxation mapping \mathcal{R} saying that:

"When the world changes, its previous state remains true as well."
Relaxed world:



- ▶ **Definition 3.1 (Delete Relaxation).** Let $\Pi = \langle P, A, I, G \rangle$ be a STRIPS task. The **delete relaxation** of Π is the task $\Pi^+ = \langle P, A^+, I, G \rangle$ where $A^+ := \{a^+ | a \in A\}$ with $\text{pre}_{a^+} = \text{pre}_a$, $\text{add}_{a^+} = \text{add}_a$, and $\text{del}_{a^+} = \emptyset$.
- ▶ In other words, the class of simpler problems \mathcal{P}' is the set of all STRIPS tasks with empty delete lists, and the relaxation mapping \mathcal{R} drops the delete lists.
- ▶ **Definition 3.2 (Relaxed Plan).** Let $\Pi = \langle P, A, I, G \rangle$ be a STRIPS task, and let s be a state. A **relaxed plan** for s is a **plan** for $\langle P, A, s, G \rangle^+$. A relaxed plan for I is called a relaxed plan for Π .
- ▶ A relaxed plan for s is an action sequence that solves s when pretending that all delete lists are empty.
- ▶ Also called “delete-relaxed plan”: “relaxation” is often used to mean “delete-relaxation” by default.

A Relaxed Plan for “TSP” in Australia



1. **Initial state:** $\{\text{at}(Sy), \text{vis}(Sy)\}$.

A Relaxed Plan for “TSP” in Australia



1. **Initial state:** $\{\text{at}(Sy), \text{vis}(Sy)\}$.
2. **drv(Sy, Br)⁺:** $\{\text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.

A Relaxed Plan for “TSP” in Australia



1. **Initial state:** $\{\text{at}(Sy), \text{vis}(Sy)\}$.
2. **drv(Sy, Br)⁺:** $\{\text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
3. **drv(Sy, Ad)⁺:** $\{\text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.

A Relaxed Plan for “TSP” in Australia



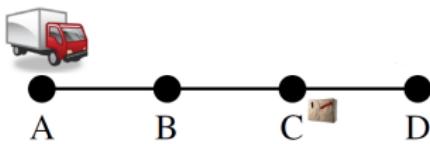
1. **Initial state:** $\{\text{at}(Sy), \text{vis}(Sy)\}$.
2. **$\text{drv}(Sy, Br)^+$:** $\{\text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
3. **$\text{drv}(Sy, Ad)^+$:** $\{\text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
4. **$\text{drv}(Ad, Pe)^+$:** $\{\text{at}(Pe), \text{vis}(Pe), \text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.

A Relaxed Plan for “TSP” in Australia



1. **Initial state:** $\{\text{at}(Sy), \text{vis}(Sy)\}$.
2. **drv(Sy, Br)⁺:** $\{\text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
3. **drv(Sy, Ad)⁺:** $\{\text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
4. **drv(Ad, Pe)⁺:** $\{\text{at}(Pe), \text{vis}(Pe), \text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
5. **drv(Ad, Da)⁺:**
 $\{\text{at}(Da), \text{vis}(Da), \text{at}(Pe), \text{vis}(Pe), \text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.

A Relaxed Plan for “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 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 ”.

- ▶ **Definition 3.3 (Relaxed Plan Existence Problem).** By PlanEx⁺, we denote the problem of deciding, given a STRIPS task $\Pi = \langle P, A, I, G \rangle$, whether or not there exists a **relaxed plan** for Π .
- ▶ This is easier than PlanEx for general STRIPS!
- ▶ **Assertion (PlanEx⁺ is Easy)** PlanEx⁺ is a member of **P**.
- ▶ **Proof:** The following algorithm decides PlanEx⁺
 - 1.

```
var  $F := I$ 
while  $G \not\subseteq F$  do
     $F' := F \cup \bigcup_{a \in A: \text{pre}_a \subseteq F} \text{add}_a$ 
    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 602



Deciding PlanEx⁺ in “TSP” in Australia

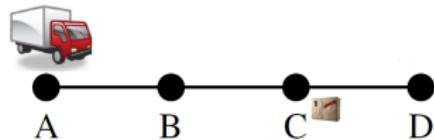


Iterations on F :

1. $\{ \text{at}(Sy), \text{vis}(Sy) \}$
2. $\cup \{ \text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br) \}$
3. $\cup \{ \text{at}(Da), \text{vis}(Da), \text{at}(Pe), \text{vis}(Pe) \}$

Deciding PlanEx⁺ in “Logistics”

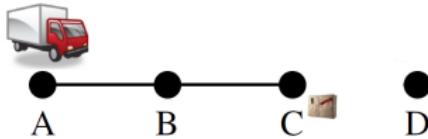
► Example 3.4 (The solvable Case).



Iterations on F :

1. {truck(A), pack(C)}
2. $\cup\{\text{truck}(B)\}$
3. $\cup\{\text{truck}(C)\}$
4. $\cup\{\text{truck}(D), \text{pack}(T)\}$
5. $\cup\{\text{pack}(A), \text{pack}(B), \text{pack}(D)\}$

► Example 3.5 (The unsolvable Case).



Iterations on F :

1. {truck(A), pack(C)}
2. $\cup\{\text{truck}(B)\}$
3. $\cup\{\text{truck}(C)\}$
4. $\cup\{\text{pack}(T)\}$
5. $\cup\{\text{pack}(A), \text{pack}(B)\}$
6. $\cup\emptyset$

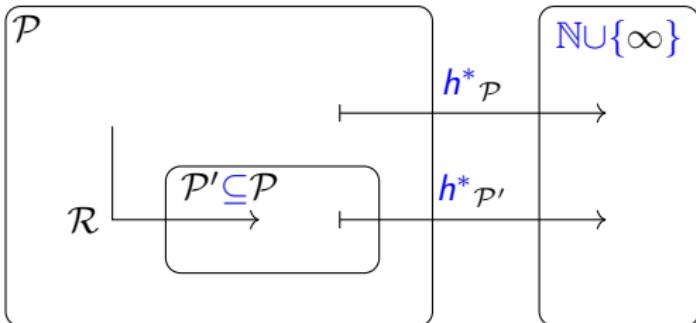
$\mathbb{P} \triangleleft \mathcal{D} \times \mathbb{E} \curvearrowright^+$ Algorithm: Proof

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 i th iteration of the while-loop,
2. All $a \in A_0$ are applicable in I , all $a \in A_1$ are applicable in $\text{apply}(I, A_0^+)$, and so forth.
3. Thus $F_i = \text{apply}(I, \langle A_0^+, \dots, A_{i-1}^+ \rangle)$. (Within each A_j^+ , we can sequence the actions in any order.)
4.
 - 4.1. **Direction “ \Rightarrow ”:** If “solvable” is returned after iteration n then $G \subseteq F_n = \text{apply}(I, \langle A_0^+, \dots, A_{n-1}^+ \rangle)$ so $\langle A_0^+, \dots, A_{n-1}^+ \rangle$ can be sequenced to a relaxed plan which shows the claim. \square
 - 4.2. **Direction “ \Leftarrow ”:**
 - 4.2.1. Let $\langle a_0^+, \dots, a_{n-1}^+ \rangle$ be a relaxed plan, hence $G \subseteq \text{apply}(I, \langle a_0^+, \dots, a_{n-1}^+ \rangle)$.
 - 4.2.2. Assume, for the moment, that we drop line (*) from the algorithm. It is then easy to see that $a_i \in A_i$ and $\text{apply}(I, \langle a_0^+, \dots, a_{i-1}^+ \rangle) \subseteq F_i$, for all i .
 - 4.2.3. We get $G \subseteq \text{apply}(I, \langle a_0^+, \dots, a_{n-1}^+ \rangle) \subseteq F_n$, and the algorithm returns “solvable” as desired.
 - 4.2.4. 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. \square

4 The h^+ Heuristic

Hold on a Sec – Where are we?



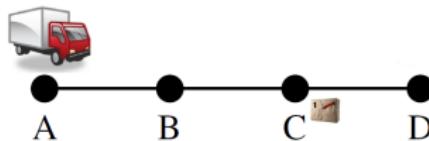
- ▶ \mathcal{P} : STRIPS tasks; $h^*_{\mathcal{P}}$: Length h^* of a shortest plan.
- ▶ $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS tasks with empty delete lists.
- ▶ \mathcal{R} : Drop the delete lists.
- ▶ Heuristic function: Length of a shortest *relaxed* plan ($h^* \circ \mathcal{R}$).
- ▶ PlanEx^+ is not actually what we're looking for. $\text{PlanEx}^+ \cong$ relaxed plan *existence*; we want relaxed plan *length* $h^* \circ \mathcal{R}$.

- ▶ **Definition 4.1 (Optimal Relaxed Plan).** Let $\Pi = \langle P, A, I, G \rangle$ 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 596, just adding the word “optimal”.
- ▶ **Here's what we're looking for:**
- ▶ **Definition 4.2.** Let $\Pi = \langle P, A, I, G \rangle$ be a STRIPS task with states S . The ideal delete relaxation heuristic h^+ for Π is the function $h^+ : S \rightarrow \mathbb{N} \cup \{\infty\}$ where $h^+(s)$ is the length of an optimal relaxed plan for s if a relaxed plan for s exists, and $h^+(s) = \infty$ otherwise.
- ▶ In other words, $h^+ = h^* \circ \mathcal{R}$, cf. previous slide.

- ▶ **Lemma 4.3.** Let $\Pi = \langle P, A, I, G \rangle$ be a STRIPS task, and let s be a state. If $\langle a_1, \dots, a_n \rangle$ is a plan for $\langle P, A, s, G \rangle$, then $\langle a_1^+, \dots, a_n^+ \rangle$ is a plan for $\langle P, A, s, G \rangle^+$.
- ▶ **Proof Sketch:** Show by induction over $0 \leq i \leq n$ that $\text{apply}(s, \langle a_1, \dots, a_i \rangle) \subseteq \text{apply}(s, \langle a_1^+, \dots, a_i^+ \rangle)$. □
- ▶ If we ignore deletes, the states along the plan can only get bigger.
- ▶ **Theorem 4.4.** h^+ is Admissible.
- ▶ **Proof:**
 1. Let $\Pi = \langle P, A, I, G \rangle$ be a STRIPS task with states S , and let $s \in S$.
 2. $h^+(s)$ is defined as optimal plan length in $\langle P, A, s, G \rangle^+$.
 3. With the lemma above, any plan for $\langle P, A, s, G \rangle$ also constitutes a plan for $\langle P, A, s, G \rangle^+$.
 4. Thus optimal plan length in $\langle P, A, s, G \rangle^+$ can only be shorter than that in $\langle P, A, s, G \rangle$, and the claim follows.□

How to Relax During Search: Ignoring Deletes

Real problem:



- ▶ Initial state I : AC ; goal G : AD .
- ▶ Actions A : [pre](#), [add](#), [del](#).
- ▶ $drXY$, loX , ulX .

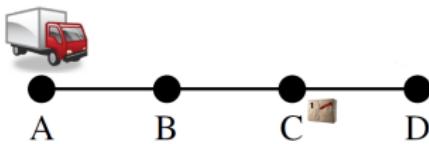
Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

Relaxed problem:



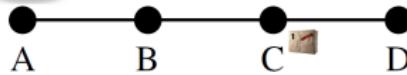
- ▶ State s : AC ; goal G : AD .
- ▶ Actions A : [pre](#), [add](#).
- ▶ $h^+(s) =$

Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes



Relaxed problem:

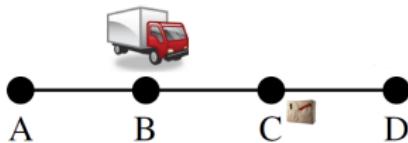
- ▶ State s : AC ; goal G : AD .
- ▶ Actions A : [pre](#), [add](#).
- ▶ $h^+(s) = 5$: e.g.
 $\langle drAB, drBC, drCD, loC, uID \rangle$.
(tie-breaking: alphabetic)

Greedy best-first search:



How to Relax During Search: Ignoring Deletes

Real problem:

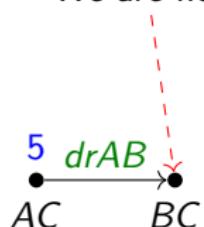


- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions A : pre, add, del.
- ▶ $AC \xrightarrow{drAB} BC$.

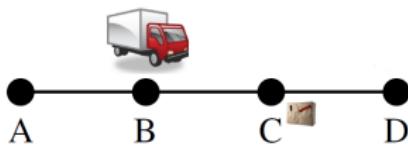
Greedy best-first search:

(tie-breaking: alphabetic)

We are here



How to Relax During Search: Ignoring Deletes



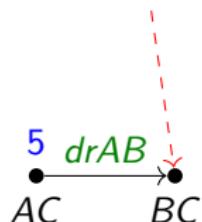
Relaxed problem:

- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions A : pre, add.
- ▶ $h^+(s) =$

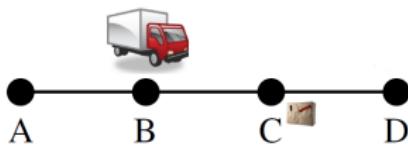
Greedy best-first search:

(tie-breaking: alphabetic)

We are here



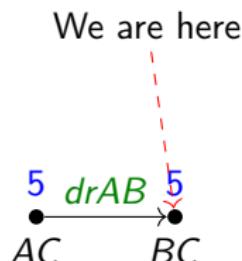
How to Relax During Search: Ignoring Deletes



Relaxed problem:

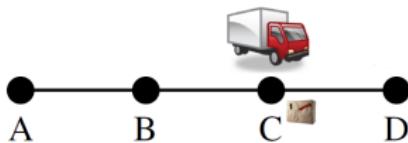
- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions A : `pre, add.`
- ▶ $h^+(s) = 5$: e.g.
 $\langle drBA, drBC, drCD, loC, uID \rangle$.
(tie-breaking: alphabetic)

Greedy best-first search:



How to Relax During Search: Ignoring Deletes

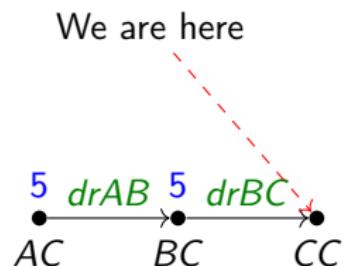
Real problem:



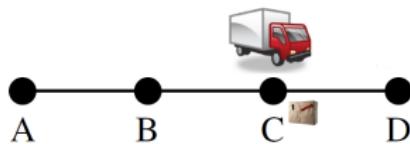
- ▶ State s : CC; goal G : AD.
- ▶ Actions A : pre, add, del.
- ▶ $BC \xrightarrow{drBC} CC$.

Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

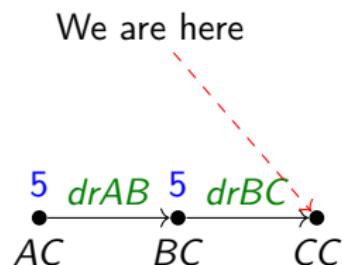


Relaxed problem:

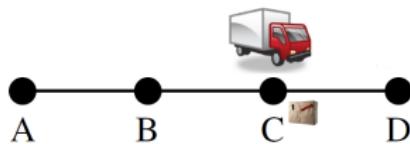
- ▶ State s : CC; goal G : AD.
- ▶ Actions A : pre, add.
- ▶ $h^+(s) =$

Greedy best-first search:

(tie-breaking: alphabetic)



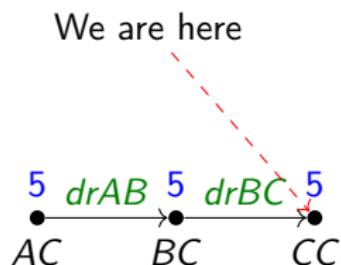
How to Relax During Search: Ignoring Deletes



Relaxed problem:

- ▶ State s : CC; goal G : AD.
- ▶ Actions A : pre, add.
- ▶ $h^+(s) = 5$: e.g.
 $\langle drCB, drBA, drCD, loC, ulD \rangle$.
(tie-breaking: alphabetic)

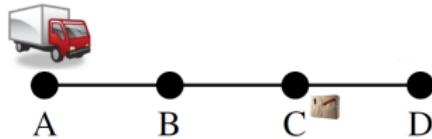
Greedy best-first search:



How to Relax During Search: Ignoring Deletes

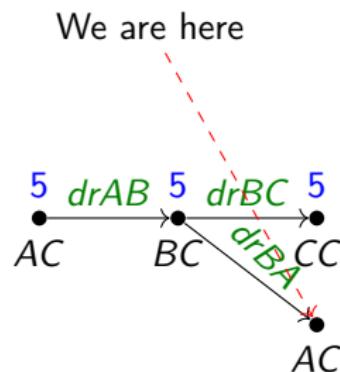
Real problem:

- ▶ State $s: AC$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del.}$
- ▶ $BC \xrightarrow{drBA} AC$.



Greedy best-first search:

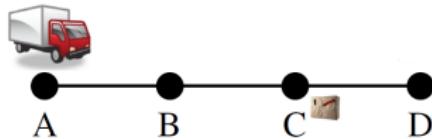
(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

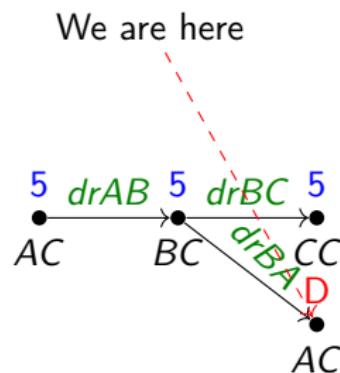
Real problem:

- ▶ State $s: AC$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ Duplicate state, prune.



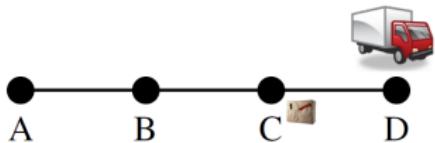
Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

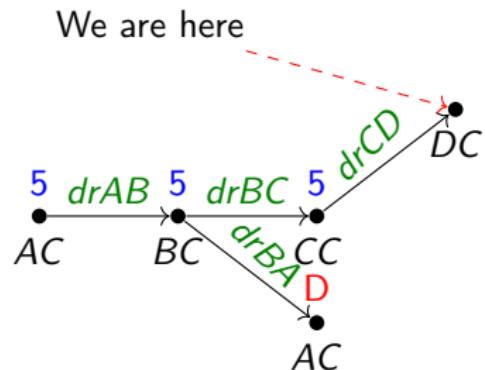
Real problem:



- ▶ State $s: DC$; goal $G: AD$.
- ▶ Actions A : pre, add, del.
- ▶ $CC \xrightarrow{drCD} DC$.

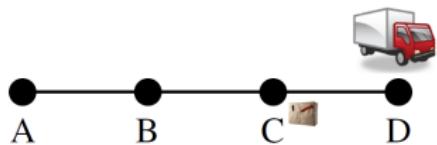
Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

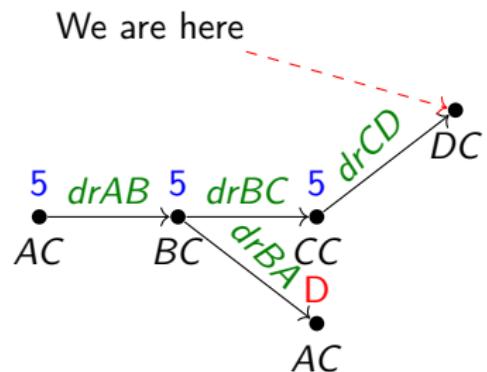
Relaxed problem:



- ▶ State s : DC ; goal G : AD .
- ▶ Actions A : **pre, add.**
- ▶ $h^+(s) =$

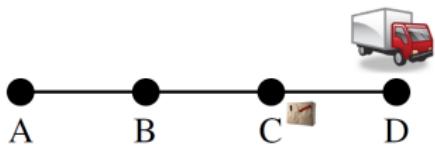
Greedy best-first search:

(tie-breaking: alphabetic)



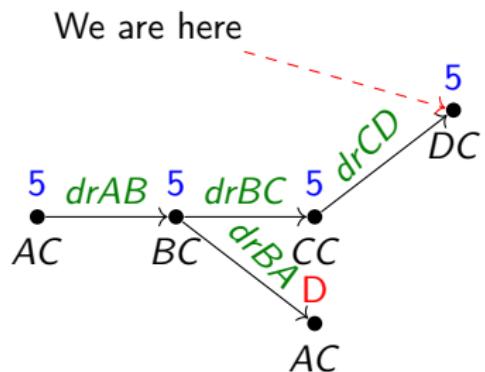
How to Relax During Search: Ignoring Deletes

Relaxed problem:



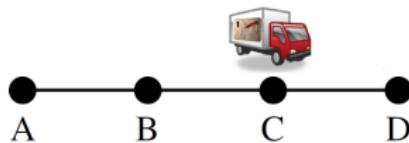
- ▶ State s : DC ; goal G : AD .
- ▶ Actions A : [pre](#), [add](#).
- ▶ $h^+(s) = 5$: e.g.
 $\langle drDC, drCB, drBA, loC, uID \rangle$.
(tie-breaking: alphabetic)

Greedy best-first search:



How to Relax During Search: Ignoring Deletes

Real problem:

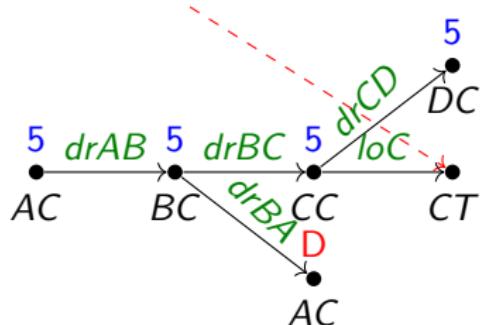


- ▶ State s : CT ; goal G : AD .
 - ▶ Actions A : [pre](#), [add](#), [del](#).
 - ▶ $CC \xrightarrow{loC} CT$.

Greedy best-first search:

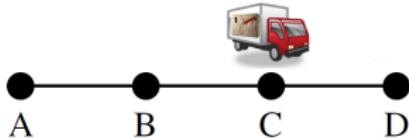
(tie-breaking: alphabetic)

We are here



How to Relax During Search: Ignoring Deletes

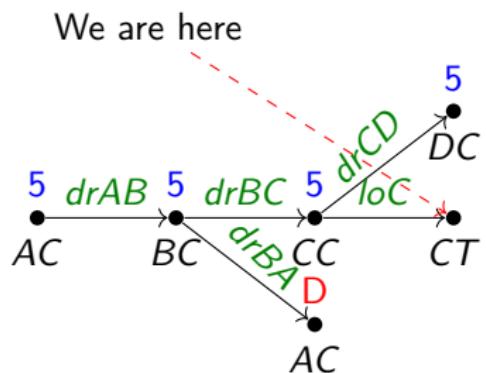
Relaxed problem:



- ▶ State s : CT ; goal G : AD .
- ▶ Actions A : `pre, add.`
- ▶ $h^+(s) =$

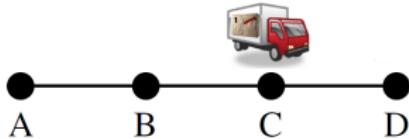
Greedy best-first search:

(tie-breaking: alphabetic)



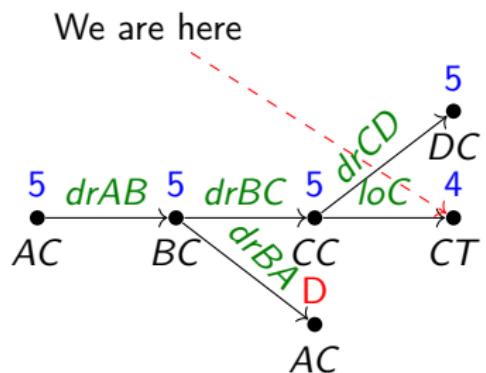
How to Relax During Search: Ignoring Deletes

Relaxed problem:



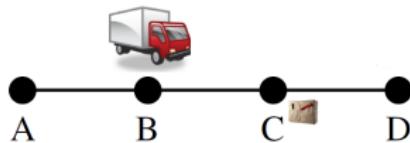
Greedy best-first search:

- ▶ State s : CT ; goal G : AD .
- ▶ Actions A : [pre](#), [add](#).
- ▶ $h^+(s) = 4$: e.g.
 $\langle drCB, drBA, drCD, uID \rangle$.
(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

Real problem:

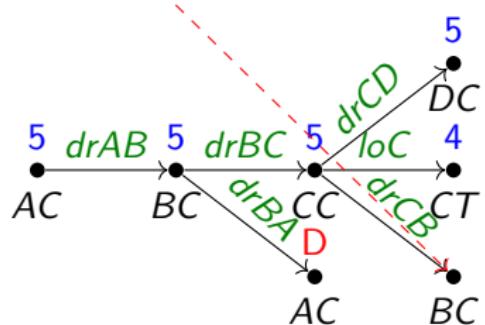


- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions A : pre, add, del.
- ▶ $CC \xrightarrow{drCB} BC$.

Greedy best-first search:

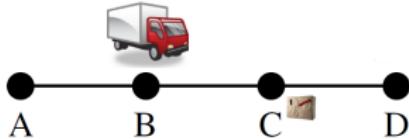
(tie-breaking: alphabetic)

We are here



How to Relax During Search: Ignoring Deletes

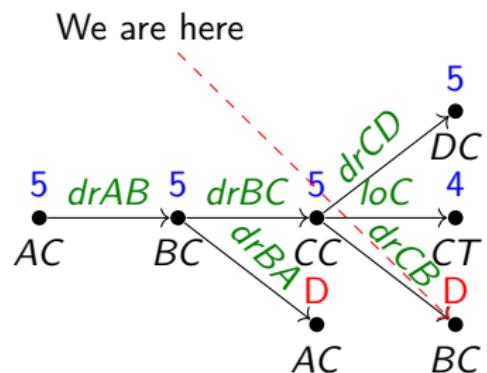
Real problem:



- ▶ State $s: BC$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ Duplicate state, prune.

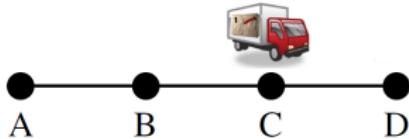
Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

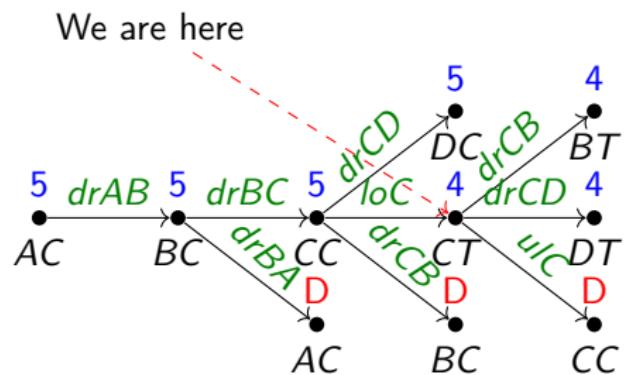
Real problem:



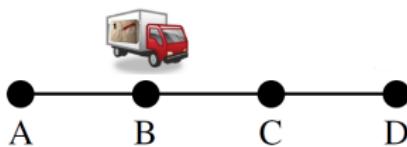
- ▶ State s : CT ; goal G : AD .
- ▶ Actions A : [pre](#), [add](#), [del](#).
- ▶ Successors: BT , DT , CC .

Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes



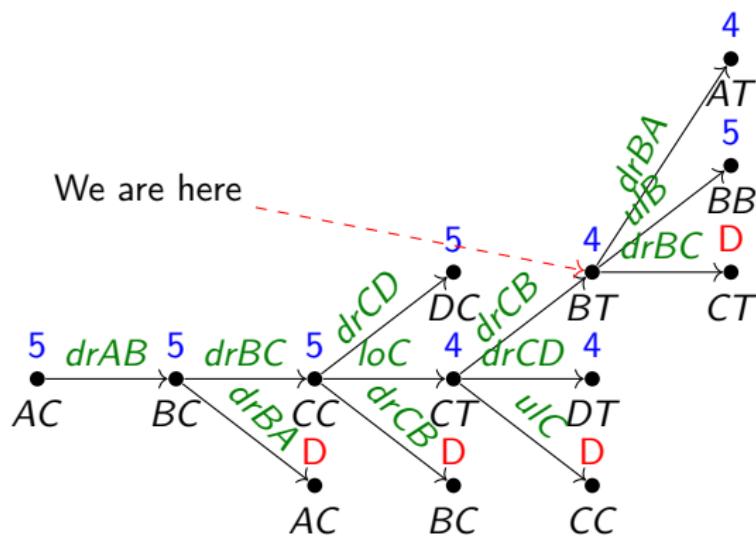
Real problem:

- ▶ State s : BT ; goal G : AD .
 - ▶ Actions A : [pre](#), [add](#), [del](#).
 - ▶ Successors: AT , BB , CT .

Greedy best-first search:

(tie-breaking: alphabetic)

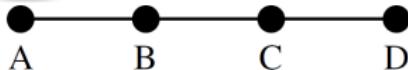
We are here



How to Relax During Search: Ignoring Deletes

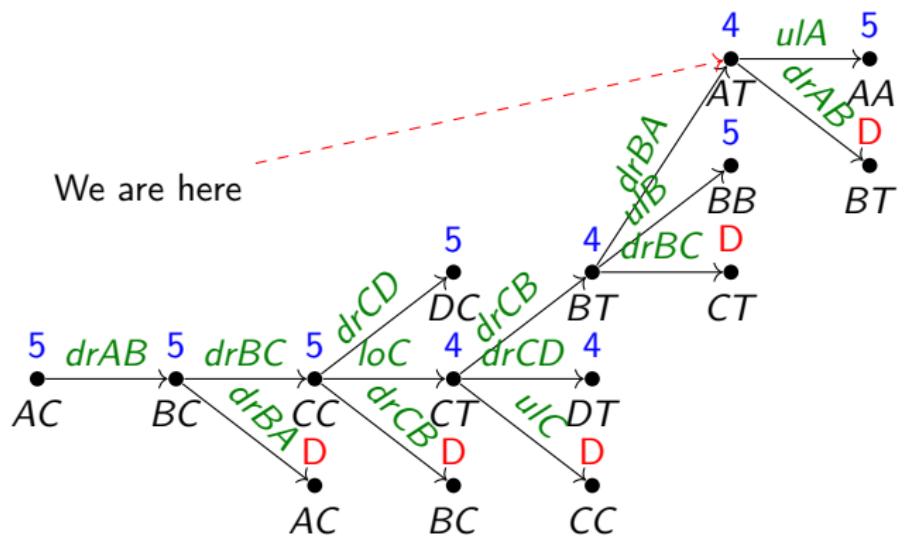
Real problem:

- ▶ State s : AT ; goal G : AD .
- ▶ Actions A : [pre](#), [add](#), [del](#).
- ▶ Successors: AA , BT .



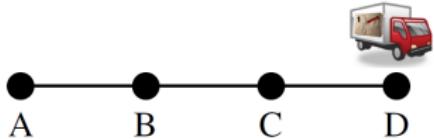
Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

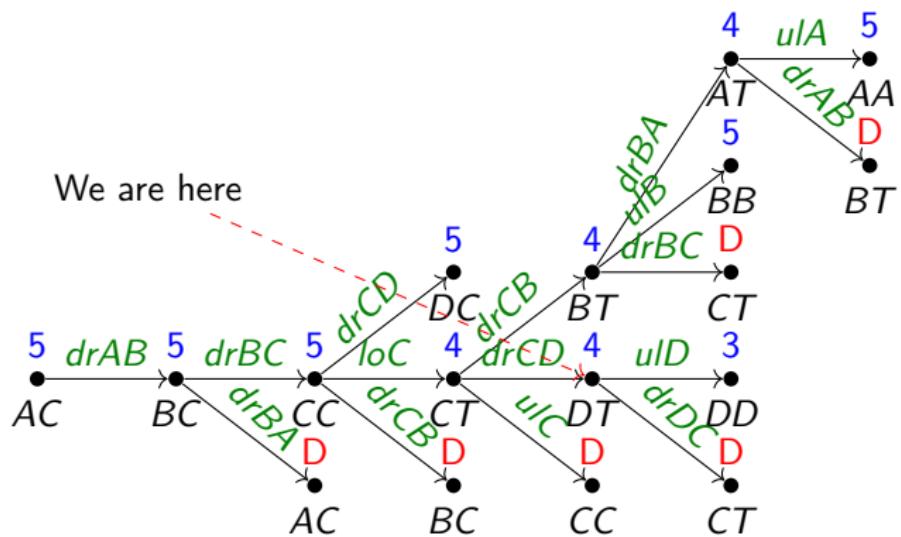
Real problem:



- ▶ State s : DT ; goal G : AD .
- ▶ Actions A : [pre](#), [add](#), [del](#).
- ▶ Successors: DD , CT .

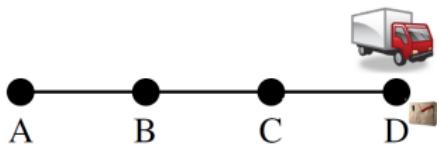
Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

Real problem:

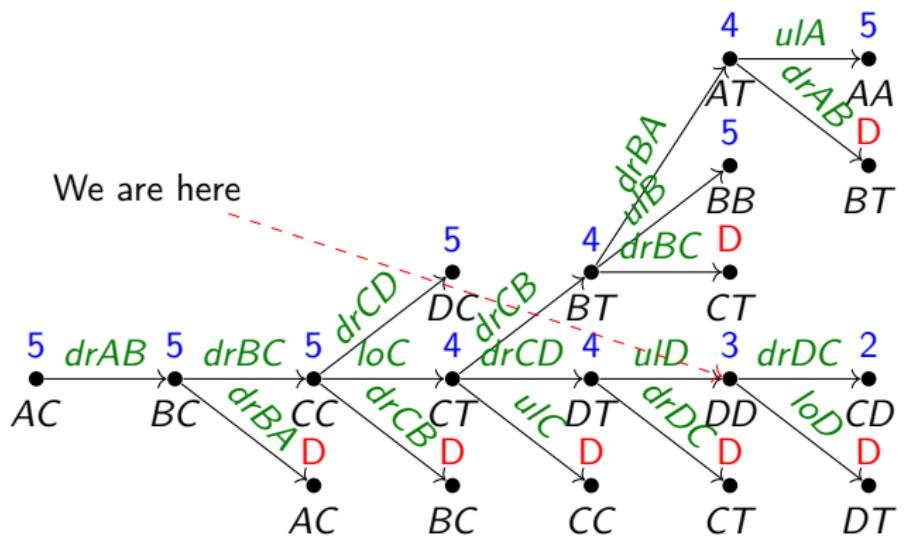


- ▶ State s : DD ; goal G : AD .
 - ▶ Actions A : [pre](#), [add](#), [del](#).
 - ▶ Successors: CD , DT .

Greedy best-first search:

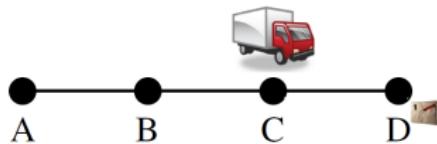
(tie-breaking: alphabetic)

We are here



How to Relax During Search: Ignoring Deletes

Real problem:

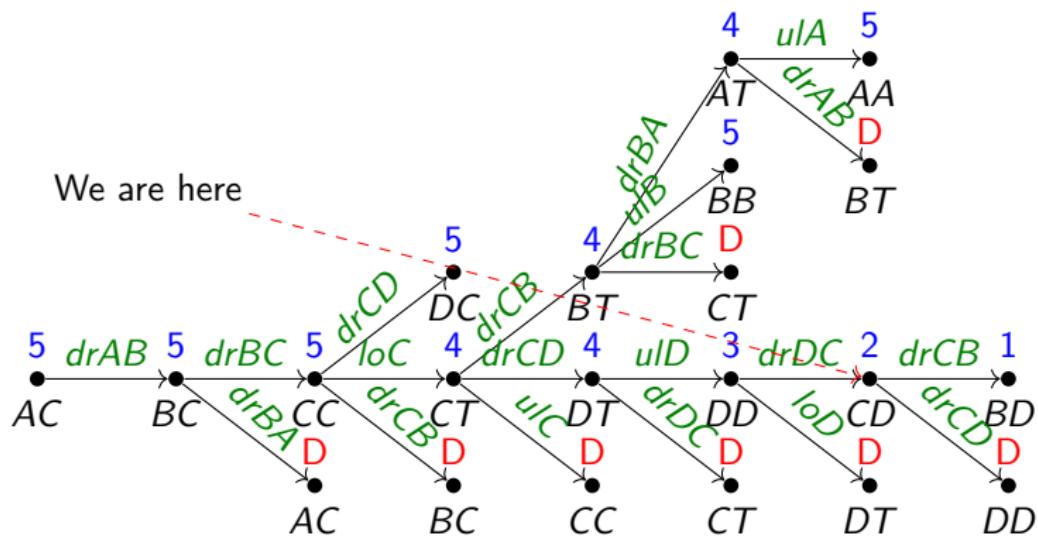


- ▶ State s : CD ; goal G : AD .
 - ▶ Actions A : [pre](#), [add](#), [del](#).
 - ▶ Successors: BD , DD .

Greedy best-first search:

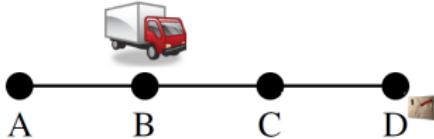
(tie-breaking: alphabetic)

We are here



How to Relax During Search: Ignoring Deletes

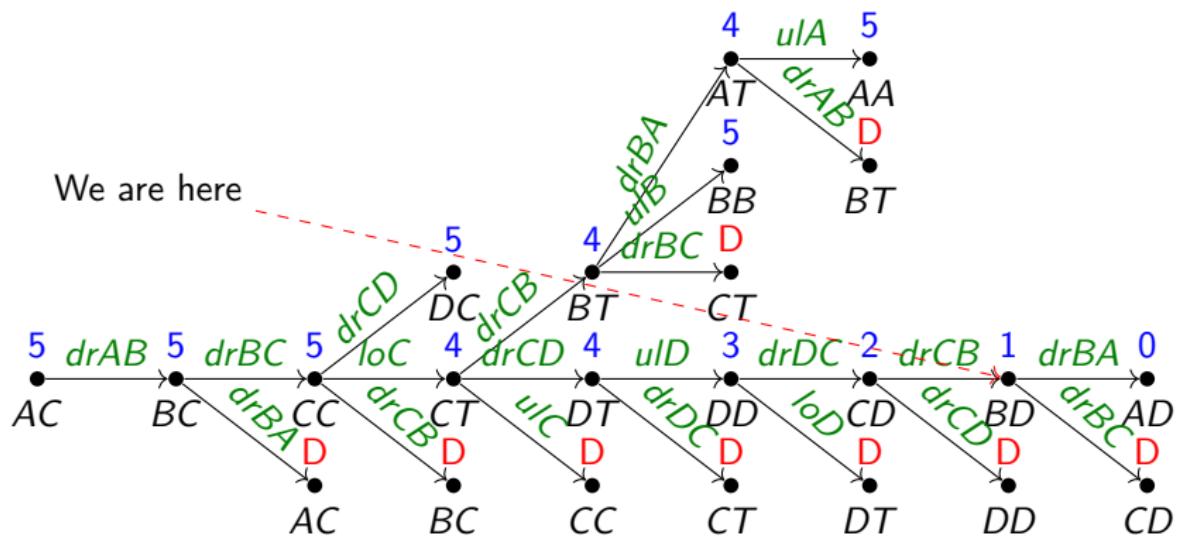
Real problem:



- ▶ State $s: BD$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ Successors: AD, CD .

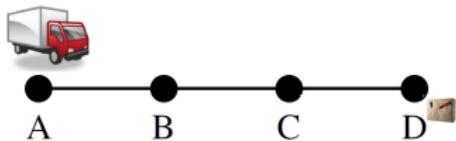
Greedy best-first search:

(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

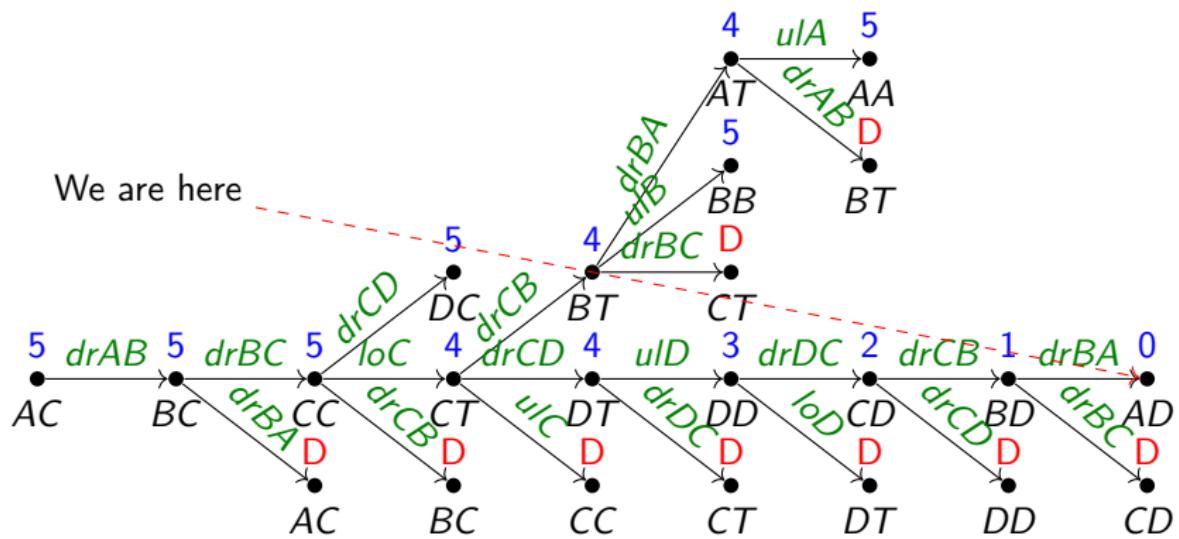
Real problem:



- ▶ State $s: AD$; goal $G: AD$.
- ▶ Actions $A: \text{pre, add, del}$.
- ▶ Goal state!

Greedy best-first search:

(tie-breaking: alphabetic)

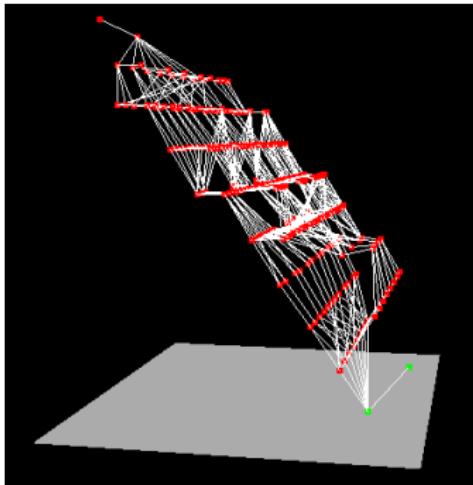


On the “Accuracy” of h^+

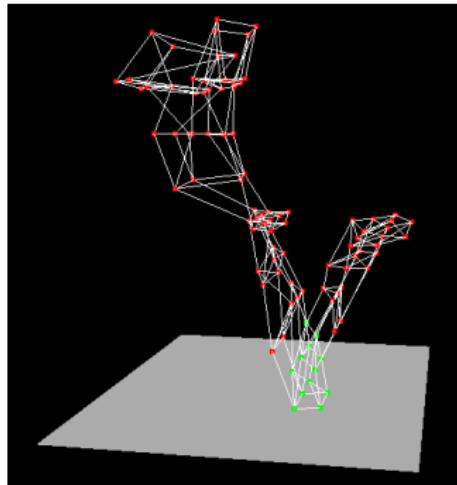
- ▶ **Reminder:** Heuristics based on ignoring deletes are the key ingredient to almost all IPC winners of the last decade.
- ▶ **Why?**: A heuristic function is useful if its estimates are “accurate”.
- ▶ **How to measure this?**:
 - ▶ Known method 1: Error relative to h^* , i.e., bounds on $|h^*(s) - h(s)|$.
 - ▶ Known method 2: Properties of the search space surface: Local minima etc.
- ▶ For h^+ , method 2 is the road to success:
- ▶ In many benchmarks, under h^+ , local minima *provably* do not exist! [Hof05]

A Brief Glimpse of h^+ Search Space Surfaces

- ▶ Graphs $\hat{=}$ state spaces, vertical height $\hat{=}$ h^+ :



“Gripper”



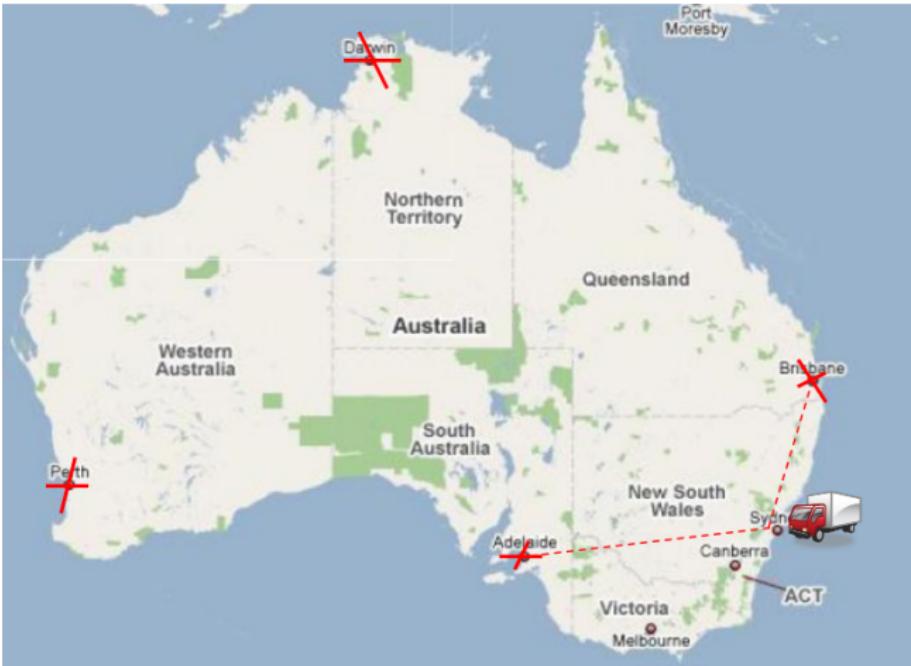
“Logistics”

- ▶ On the side: In Russel/Norvig the text reads as if these illustrations referred to computing the heuristic, rather than to finding a plan.

h^+ in (the Real) TSP



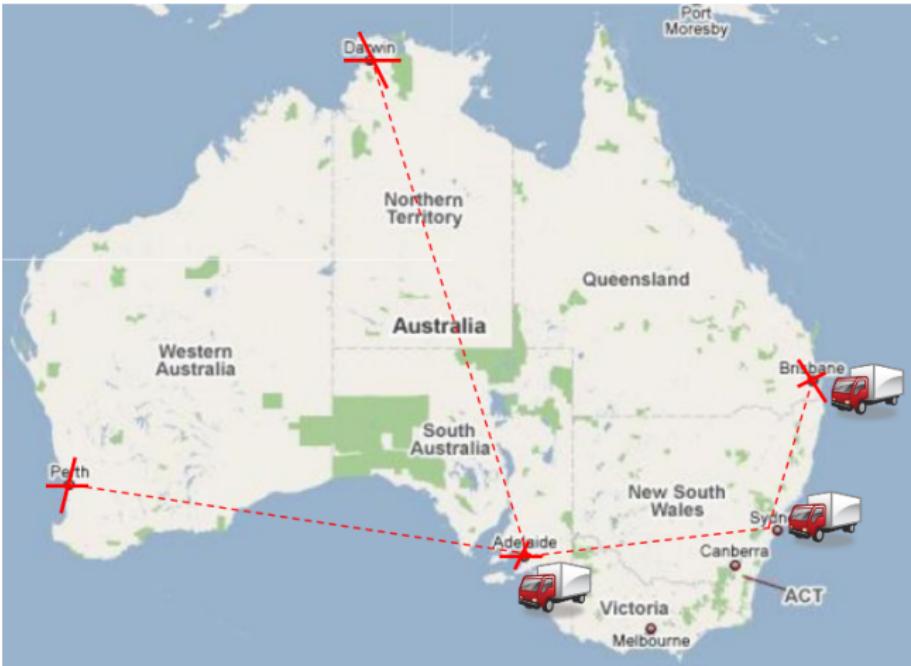
h^+ in (the Real) TSP



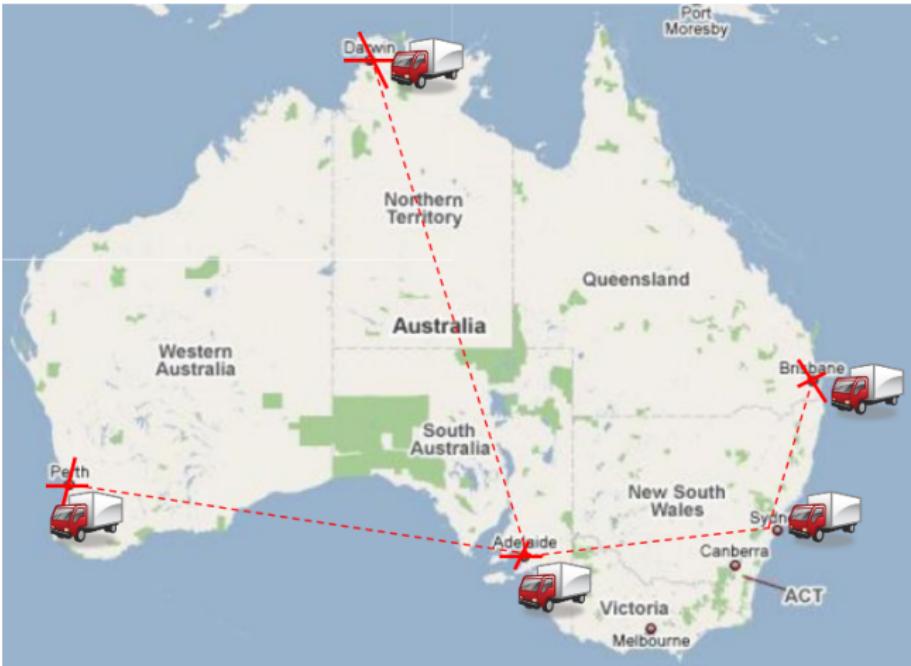
h^+ in (the Real) TSP



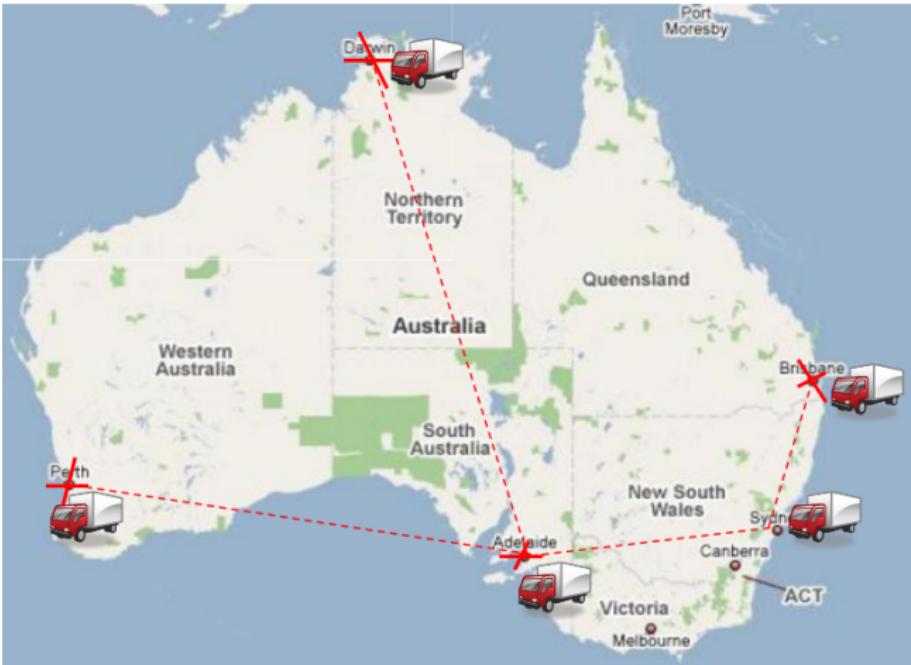
h^+ in (the Real) TSP



h^+ in (the Real) TSP

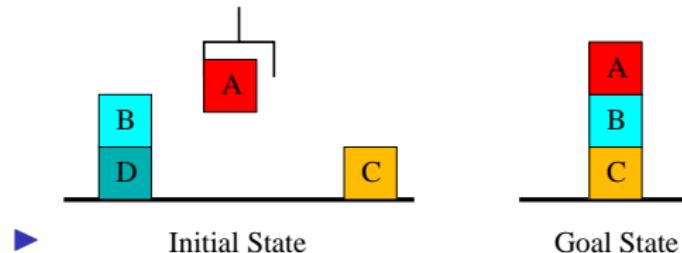


h^+ in (the Real) TSP



$h^+ \triangleq \text{Minimum Spanning Tree}$

h^+ in the Blocksworld



- ▶ Optimal plan:
 $\langle \text{putdown}(A), \text{unstack}(B, D), \text{stack}(B, C), \text{pickup}(A), \text{stack}(A, B) \rangle.$
- ▶ Optimal relaxed plan: $\langle \text{stack}(A, B), \text{unstack}(B, D), \text{stack}(B, C) \rangle.$
- ▶ 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 neighbours of these two states have a strictly higher h^+ value.

5 Approximating h^+

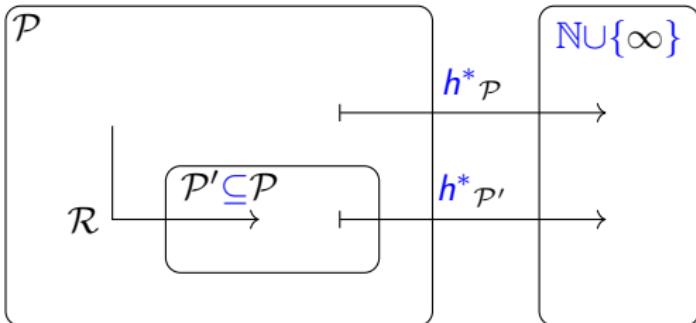
How to Compute h^+ ?

- ▶ **Definition 5.1.** By PlanLen^+ , we denote the problem of deciding, given a STRIPS task Π and an integer B , whether there exists a relaxed plan for Π of length at most B .
- ▶ By computing h^+ , we solve PlanLen^+ . (and to compute h^+ we have to solve PlanLen^+)

PlanLen⁺ is NP-complete

- ▶ **Theorem 5.2.** PlanLen⁺ is NP-complete.
- ▶ Proof:
 1. (Membership) Easy: Guess action sequences of length $\min(B, |A|)$ – in a relaxed plan, each action is applied at most once!
 2. (Hardness) Trivial from our prior results, because this generalizes optimal planning under the Only-Add relaxation, .
- ▶ However, hardness of optimal Only-Adds comes from hardness of Minimum Cover, which is easy for sets (add lists) of size ≤ 2 .
- ▶ One can prove by reduction from SAT that PlanLen⁺ remains hard even with small add lists.
- ▶ Construction outline, example $C_1 = A^T$, $C_2 = A^F$:
 - ▶ Action “set X_i ;true” for every variable X_i : pre empty, add {Atrue,Aset}.
 - ▶ Action “set X_i ;false” for every variable X_i : pre empty, add {Afalse,Aset}.
 - ▶ Action “satisfy C_j ” for every clause C_j : pre Atrue, add $C_{1\text{sat}}$; pre Afalse, add $C_{2\text{sat}}$.
 - ▶ Goal “ X_i ;set” for all variables X_i , “ C_j ;sat” for all clauses C_j : Aset, $C_{1\text{sat}}$, $C_{2\text{sat}}$.
 - ▶ Length bound $B :=$ number of variables + number of clauses. (= 3 here)

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: $h^+ = h^* \circ \mathcal{R}$, which is hard to compute.
- ▶ Problem: We can't compute our heuristic h^+ efficiently.
- ▶ Idea: So we approximate it instead. (and hope that this still gives a good heuristic)

Approximating h^+ : h^{FF} (the Base Heuristic of the FF Planner)

- ▶ **Definition 5.3.** Let $\Pi = \langle P, A, I, G \rangle$ be a STRIPS task with states S . A **relaxed plan heuristic** h^{FF} for Π is a function $h^{\text{FF}}: S \rightarrow \mathbb{N} \cup \{\infty\}$ returning the length of some, not necessarily optimal, relaxed plan for s if a relaxed plan for s exists, and returning $h^{\text{FF}}(s) = \infty$ otherwise.
- ▶ **Notes:**
 - ▶ $h^{\text{FF}} \geq h^+$, i.e., h^{FF} never under-estimates h^+ .
 - ▶ We may have $h^{\text{FF}} > h^*$, i.e., h^{FF} is not admissible! Thus h^{FF} can be used for satisficing planning only, not for optimal planning.
- ▶ **Observe:** h^{FF} as per this definition is not unique.
- ▶ How do we find *some, not necessarily optimal, relaxed plan for $\langle P, A, s, G \rangle$?*
- ▶ In what follows, we consider the following algorithm computing relaxed **plans**, and therewith (one variant of) h^{FF} :
 1. Chain **forward** to build a **relaxed planning graph (RPG)**.
 2. Chain **backward** to extract a relaxed plan from the RPG.

Computing h^F : Relaxed Planning Graphs (RPG)

► Definition 5.4 (Level Saturation for RPG).

```
let  $F_0 := s, t := 0$ 
while  $G \not\subseteq F_t$  do
     $A_t := \{a \in A \mid \text{pre}_a \subseteq F_t\}$ 
     $F_{t+1} := F_t \cup \bigcup_{a \in A_t} \text{add}_a$ 
    if  $F_{t+1} = F_t$  then stop end if
     $t := t + 1$ 
end while
```

- Does this look familiar to you?
- It's the same algorithm we used to decide PlanEx⁺ (slide 599).
- “Logistics” example: Blackboard (similar to slide 601)
- Are we done? c.f. slide 602: “ $\langle A_0^+, \dots, A_{n-1}^+ \rangle$ can be sequenced to a relaxed plan”. Could use this as the basis of h^F .
- But this would overestimate vastly!
- In “Logistics”, $\sum_{i=0}^{n-1} \#(A_i) = 11 > 8 = h^*$.
- And now imagine there are 100 packages only one of which needs to be transported ...

Computing h^F : Extracting a Relaxed Plan

- **Definition 5.5 (Information from the RPG).** Once we ran the algorithm from 5.4 we define

- $\text{level}(p) := \min \{t \mid p \in F_t\}$ for $p \in P$. ($\min \emptyset := \infty$)
- $\text{level}(a) := \min \{t \mid a \in A_t\}$ for $a \in A$.

- **Definition 5.6 (RPG-Informed Level Saturation).**

```
let  $M := \max \{\text{level}(p) \mid p \in G\}$ 
If  $M = \infty$  then  $h^F(s) := \infty$ ; stop endif
for  $t := 0, \dots, M$  do
     $G_t := \{g \in G \mid \text{level}(g) = t\}$ 
endfor
for  $t := M, \dots, 1$  do
    for all  $g \in G_t$  do
        select  $a$ ,  $\text{level}(a) = t - 1$ ,  $g \in \text{add}_a$ 
        for all  $p \in \text{pre}_a$  do  $G_{\text{level}(p)} := G_{\text{level}(p)} \cup \{p\}$  endfor
    endfor
endfor
 $h^F(s) := \text{number of selected actions}$ 
```

Computing h^F in “TSP” in Australia



RPG:

Computing h^F in “TSP” in Australia



RPG:

- ▶ $F_0 = \{\text{at}(Sy), \text{vis}(Sy)\}.$

Computing h^F in “TSP” in Australia



RPG:

- ▶ $F_0 = \{\text{at}(Sy), \text{vis}(Sy)\}.$
- ▶ $A_0 = \{\text{drv}(Sy, Ad), \text{drv}(Sy, Br)\}.$

Computing h^F in “TSP” in Australia



RPG:

- ▶ $F_0 = \{\text{at}(Sy), \text{vis}(Sy)\}.$
- ▶ $A_0 = \{\text{drv}(Sy, Ad), \text{drv}(Sy, Br)\}.$
- ▶ $F_1 = F_0 \cup \{\text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br)\}.$

Computing h^F in “TSP” in Australia



RPG:

- ▶ $F_0 = \{\text{at}(Sy), \text{vis}(Sy)\}.$
- ▶ $A_0 = \{\text{drv}(Sy, Ad), \text{drv}(Sy, Br)\}.$
- ▶ $F_1 = F_0 \cup \{\text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br)\}.$
- ▶ $A_1 = A_0 \cup \{\text{drv}(Ad, Da), \text{drv}(Ad, Pe), \text{drv}(Ad, Sy), \text{drv}(Br, Sy)\}.$

Computing h^F in “TSP” in Australia



RPG:

- ▶ $F_0 = \{\text{at}(Sy), \text{vis}(Sy)\}.$
- ▶ $A_0 = \{\text{drv}(Sy, Ad), \text{drv}(Sy, Br)\}.$
- ▶ $F_1 = F_0 \cup \{\text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br)\}.$
- ▶ $A_1 = A_0 \cup \{\text{drv}(Ad, Da), \text{drv}(Ad, Pe), \text{drv}(Ad, Sy), \text{drv}(Br, Sy)\}.$
- ▶ $F_2 = F_1 \cup \{\text{at}(Da), \text{vis}(Da), \text{at}(Pe), \text{vis}(Pe)\}.$

Computing h^F in “TSP” in Australia



RPG:

- ▶ $F_0 = \{\text{at}(Sy), \text{vis}(Sy)\}.$
- ▶ $A_0 = \{\text{drv}(Sy, Ad), \text{drv}(Sy, Br)\}.$
- ▶ $F_1 = F_0 \cup \{\text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br)\}.$
- ▶ $A_1 = A_0 \cup \{\text{drv}(Ad, Da), \text{drv}(Ad, Pe), \text{drv}(Ad, Sy), \text{drv}(Br, Sy)\}.$
- ▶ $F_2 = F_1 \cup \{\text{at}(Da), \text{vis}(Da), \text{at}(Pe), \text{vis}(Pe)\}.$

Computing h^F in “TSP” in Australia



Inserting the goals:

- ▶ F_0 : $\text{at}(\text{Sy}), \text{vis}(\text{Sy})$.
- ▶ A_0 : $\text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Sy}, \text{Br})$.
- ▶ F_1 : $\text{at}(\text{Ad}), \text{vis}(\text{Ad}), \text{at}(\text{Br}), \text{vis}(\text{Br})$.
- ▶ A_1 : $\text{drv}(\text{Ad}, \text{Da}), \text{drv}(\text{Ad}, \text{Pe}), \text{drv}(\text{Ad}, \text{Sy}), \text{drv}(\text{Br}, \text{Sy})$.
- ▶ F_2 : $\text{at}(\text{Da}), \text{vis}(\text{Da}), \text{at}(\text{Pe}), \text{vis}(\text{Pe})$.

Computing h^F in “TSP” in Australia



Inserting the goals:

- ▶ F_0 : $\text{at}(\text{Sy}), \text{vis}(\text{Sy})$.
- ▶ A_0 : $\text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Sy}, \text{Br})$.
- ▶ F_1 : $\text{at}(\text{Ad}), \text{vis}(\text{Ad}), \text{at}(\text{Br}), \text{vis}(\text{Br})$.
- ▶ A_1 : $\text{drv}(\text{Ad}, \text{Da}), \text{drv}(\text{Ad}, \text{Pe}), \text{drv}(\text{Ad}, \text{Sy}), \text{drv}(\text{Br}, \text{Sy})$.
- ▶ F_2 : $\text{at}(\text{Da}), \text{vis}(\text{Da}), \text{at}(\text{Pe}), \text{vis}(\text{Pe})$.

Computing h^F in “TSP” in Australia



Supporting the goals at $t = 2$:

- ▶ F_0 : $\text{at}(\text{Sy}), \text{vis}(\text{Sy})$.
- ▶ A_0 : $\text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Sy}, \text{Br})$.
- ▶ F_1 : $\text{at}(\text{Ad}), \text{vis}(\text{Ad}), \text{at}(\text{Br}), \text{vis}(\text{Br})$.
- ▶ A_1 : $\text{drv}(\text{Ad}, \text{Da}), \text{drv}(\text{Ad}, \text{Pe}), \text{drv}(\text{Ad}, \text{Sy}), \text{drv}(\text{Br}, \text{Sy})$.
- ▶ F_2 : $\text{at}(\text{Da}), \text{vis}(\text{Da}), \text{at}(\text{Pe}), \text{vis}(\text{Pe})$.

Computing h^F in “TSP” in Australia



Supporting the goals at $t = 2$:

- ▶ F_0 : $\text{at}(\text{Sy}), \text{vis}(\text{Sy})$.
- ▶ A_0 : $\text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Sy}, \text{Br})$.
- ▶ F_1 : $\text{at}(\text{Ad}), \text{vis}(\text{Ad}), \text{at}(\text{Br}), \text{vis}(\text{Br})$.
- ▶ A_1 : $\text{drv}(\text{Ad}, \text{Da}), \text{drv}(\text{Ad}, \text{Pe}), \text{drv}(\text{Ad}, \text{Sy}), \text{drv}(\text{Br}, \text{Sy})$.
- ▶ F_2 : $\text{at}(\text{Da}), \text{vis}(\text{Da}), \text{at}(\text{Pe}), \text{vis}(\text{Pe})$.

Computing h^F in “TSP” in Australia



Supporting the goals at $t = 1$:

- ▶ F_0 : $\text{at}(\text{Sy})$, $\text{vis}(\text{Sy})$.
- ▶ A_0 : $\text{drv}(\text{Sy}, \text{Ad})$, $\text{drv}(\text{Sy}, \text{Br})$.
- ▶ F_1 : $\text{at}(\text{Ad})$, $\text{vis}(\text{Ad})$, $\text{at}(\text{Br})$, $\text{vis}(\text{Br})$.
- ▶ A_1 : $\text{drv}(\text{Ad}, \text{Da})$, $\text{drv}(\text{Ad}, \text{Pe})$, $\text{drv}(\text{Ad}, \text{Sy})$, $\text{drv}(\text{Br}, \text{Sy})$.
- ▶ F_2 : $\text{at}(\text{Da})$, $\text{vis}(\text{Da})$, $\text{at}(\text{Pe})$, $\text{vis}(\text{Pe})$.

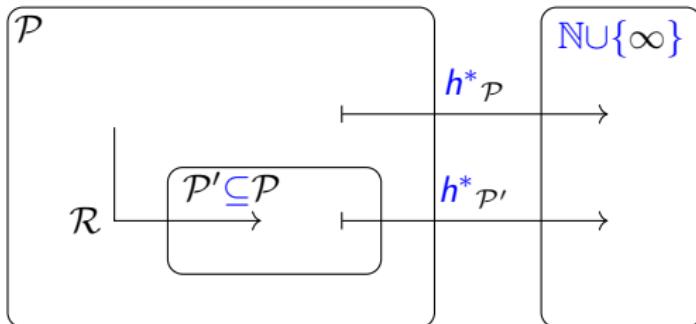
Computing h^F in “TSP” in Australia



Supporting the goals at $t = 1$:

- ▶ F_0 : $\text{at}(\text{Sy})$, $\text{vis}(\text{Sy})$.
- ▶ A_0 : $\text{drv}(\text{Sy}, \text{Ad})$, $\text{drv}(\text{Sy}, \text{Br})$.
- ▶ F_1 : $\text{at}(\text{Ad})$, $\text{vis}(\text{Ad})$, $\text{at}(\text{Br})$, $\text{vis}(\text{Br})$.
- ▶ A_1 : $\text{drv}(\text{Ad}, \text{Da})$, $\text{drv}(\text{Ad}, \text{Pe})$, $\text{drv}(\text{Ad}, \text{Sy})$, $\text{drv}(\text{Br}, \text{Sy})$.
- ▶ F_2 : $\text{at}(\text{Da})$, $\text{vis}(\text{Da})$, $\text{at}(\text{Pe})$, $\text{vis}(\text{Pe})$.

How Does it All Fit Together?



- ▶ \mathcal{P} : STRIPS tasks.
- ▶ $h^*_{\mathcal{P}}$: Length h^* of a shortest plan.
- ▶ \mathcal{P}' : STRIPS tasks with empty delete lists.
- ▶ \mathcal{R} : Drop the delete lists.
- ▶ $h^* \circ \mathcal{R}$: Length h^+ of a shortest relaxed plan.
- ▶ Use h^F to approximate h^+ which itself is hard to compute.
- ▶ h^+ is admissible; h^F is not.

Other Approximations of h^+

- ▶ **Definition 5.7.** h^{max} : Approximate the cost of fact set g by the most costly single fact $p \in g$

$$h^{max}(s, g) := \begin{cases} 0 & \text{if } g \subseteq s \\ \min_{a \in A, p \in add_a} 1 + h^{max}(s, \text{pre}_a) & \text{if } g = \{p\} \\ \max_{p \in g} h^{max}(s, \{p\}) & \text{if } \#(g) > 1 \end{cases}$$

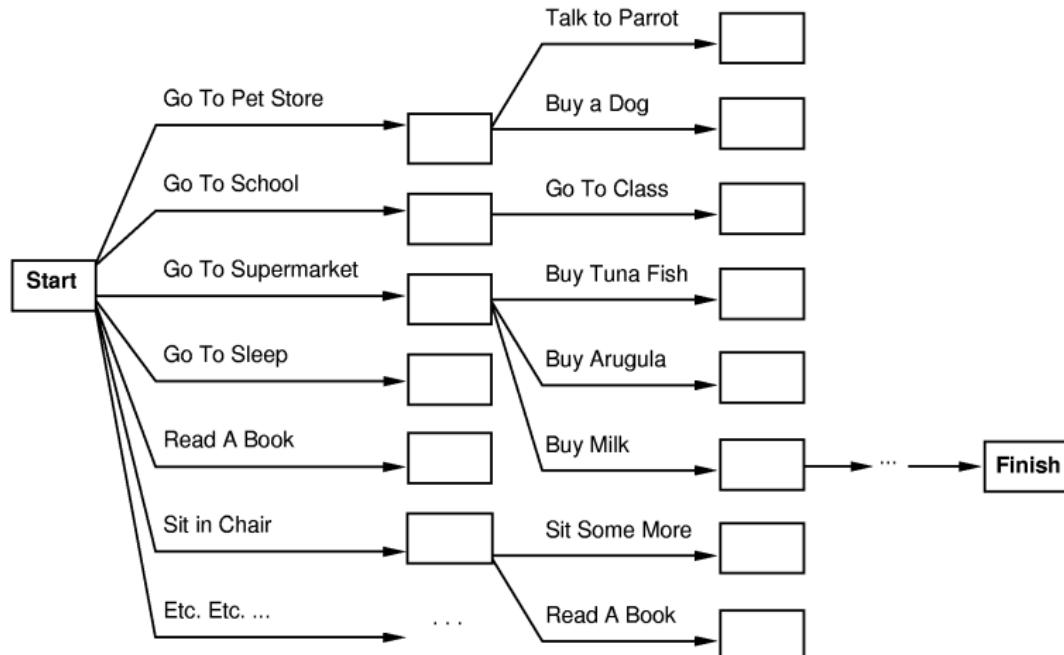
- ▶ Admissible, but very uninformative (under-estimates vastly).
- ▶ **Definition 5.8.** h^{add} : Use instead the *sum* of the costs of the single facts $p \in g$

$$h^{add}(s, g) := \begin{cases} 0 & \text{if } g \subseteq s \\ \min_{a \in A, p \in add_a} 1 + h^{add}(s, \text{pre}_a) & \text{if } g = \{p\} \\ \sum_{p \in g} h^{add}(s, \{p\}) & \text{if } \#(g) > 1 \end{cases}$$

- ▶ Not admissible, and prone to over-estimation; h^{HF} works better.
- ▶ **Good lower bounds on h^+ :**
 - ▶ **Admissible landmarks** [KD09]
 - ▶ **LM-cut** [HD09].

Helpful Actions Pruning

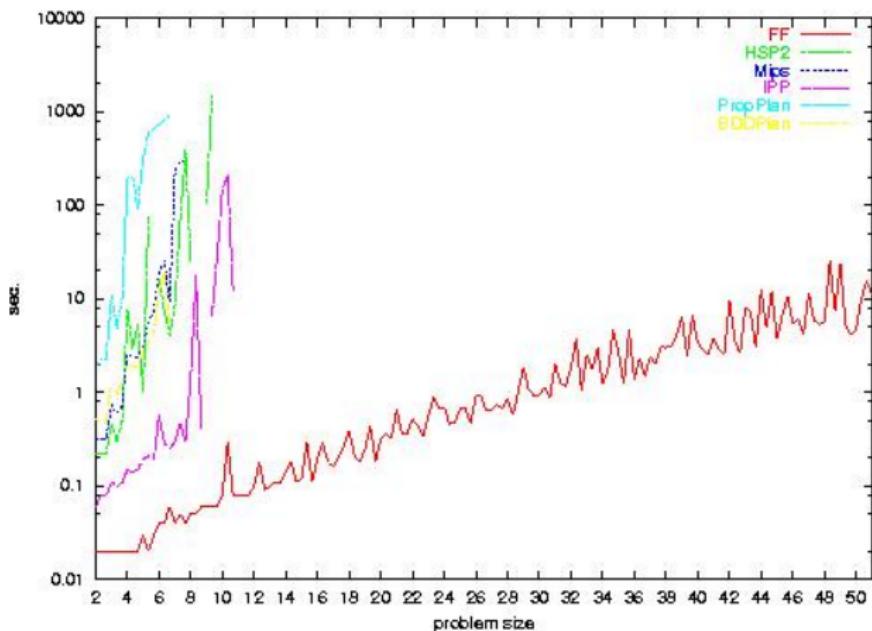
- **Idea:** In search, expand only those actions contained in the relaxed plan.
- **Example 5.9.**



Relaxed plan $\hat{=}$ “Go To Supermarket, Buy Milk, ...”

Helpful Actions Pruning

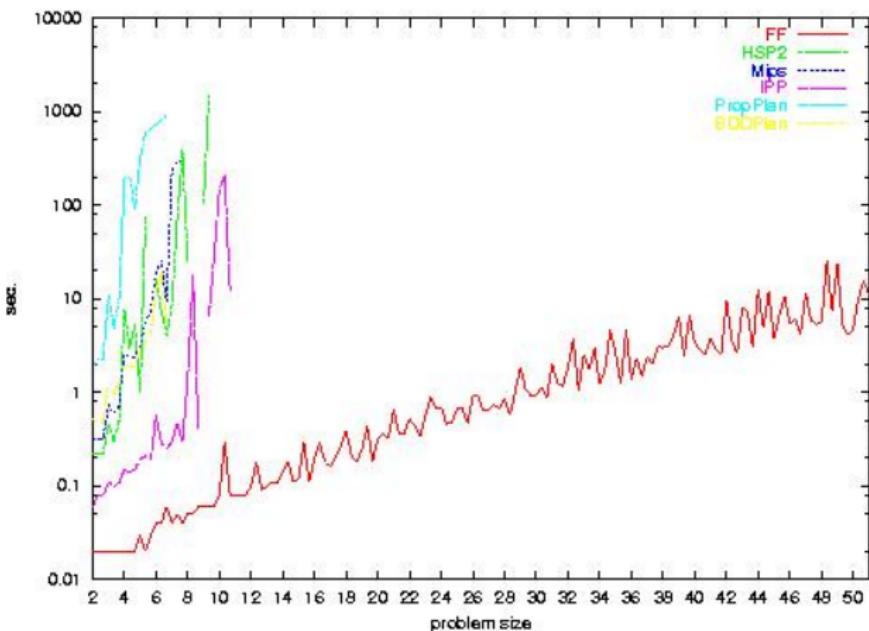
- Absolutely essential, used in all state-of-the-art satisficing planners.



domain:
tool scheduling
(many tools,
many objects)

Helpful Actions Pruning

- Absolutely essential, used in all state-of-the-art satisficing planners.



domain:
tool scheduling
(many tools,
many objects)

- Intuition: Relaxed plan does *not* drill holes into objects that need to be painted.

6 Conclusion

- ▶ Heuristic search on classical search problems relies on a function h mapping states s to an estimate $h(s)$ of their goal distance. Such functions h are derived by solving **relaxed problems**.
- ▶ In planning, the relaxed problems are generated and solved automatically. There are four known families of suitable relaxation methods: **abstractions**, **landmarks**, **critical paths**, and **ignoring deletes** (aka **delete relaxation**).
- ▶ The delete relaxation consists in dropping the deletes from **STRIPS tasks**. A **relaxed plan** is a plan for such a relaxed task. $h^+(s)$ is the length of an optimal relaxed plan for state s . h^+ is **NP-hard** to compute.
- ▶ h^{RF} approximates h^+ by computing some, not necessarily optimal, relaxed plan. That is done by a forward pass (building a **relaxed planning graph**), followed by a backward pass (**extracting a relaxed plan**).

Topics We Didn't Cover Here

- ▶ Abstractions, Landmarks, Critical-Path Heuristics, Cost Partitions, Compilability between Heuristic Functions, Planning Competitions:
- ▶ Tractable fragments: Planning sub-classes that can be solved in polynomial time. Often identified by properties of the “causal graph” and “domain transition graphs”.
- ▶ Planning as SAT: Compile length- k bounded plan existence into satisfiability of a CNF formula φ . Extensive literature on how to obtain small φ , how to schedule different values of k , how to modify the underlying SAT solver.
- ▶ Compilations: Formal framework for determining whether planning formalism X is (or is not) at least as expressive as planning formalism Y .
- ▶ Admissible pruning/decomposition methods: Partial-order reduction, symmetry reduction, simulation-based dominance pruning, factored planning, decoupled search.
- ▶ Hand-tailored planning: Automatic planning is the extreme case where the computer is given no domain knowledge other than “physics”. We can instead allow the user to provide search control knowledge, trading off modeling effort against search performance.
- ▶ Numeric planning, temporal planning, planning under uncertainty ...

Chapter 18 Searching, Planning, and Acting in the Real World

- ▶ **So Far:** we made idealizing/simplifying assumptions:
The environment is **fully observable** and **deterministic**.
- ▶ **Outline:** In this document 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 document apply to both search and planning.

1 Introduction

The real world

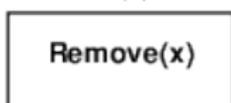
- ▶ Example 1.1. We have a flat tire – what to do?



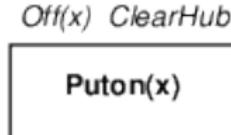
$\neg \text{Flat}(\text{Spare}) \ Intact(\text{Spare}) \ \text{Off}(\text{Spare})$
 $\text{On}(\text{Tire}1) \ \text{Flat}(\text{Tire}1)$

$\text{On}(x) \ \neg \text{Flat}(x)$

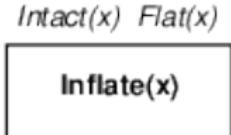
FINISH



$\text{Off}(x) \ \text{ClearHub}$



$\text{On}(x) \ \neg \text{ClearHub}$



$\neg \text{Flat}(x)$

Generally: Things go wrong (in the real world)

- ▶ **Example 1.2 (Incomplete Information).**
 - ▶ Unknown **preconditions**, e.g., *Intact(Spare)?*
 - ▶ Disjunctive **effects**, e.g., *Inflate(x)* causes
Inflated(x) ∨ SlowHiss(x) ∨ Burst(x) ∨ BrokenPump ∨ ...
- ▶ **Example 1.3 (Incorrect Information).**
 - ▶ Current **state** incorrect, e.g., spare NOT intact
 - ▶ Missing/incorrect **effects** in actions.
- ▶ **Definition 1.4.** The **qualification problem** in planning is that we can never finish listing all the required **preconditions** and possible conditional **effects** of actions.
- ▶ **Root Cause:** The **environment** is **partially observable** and/or non-deterministic.
- ▶ **Technical Problem:** We cannot know the “current state of the world”, but search/planning algorithms are based on this assumption.
- ▶ **Idea:** Adapt search/planning algorithms to work with “sets of possible states”.

What can we do if things (can) go wrong?

- ▶ One Solution: Sensorless planning: plans that work regardless of state/outcome.
- ▶ Problem: Such plans may not exist! (but they often do in practice)
- ▶ Another Solution: Conditional plans:
 - ▶ Plan to obtain information, (observation actions)
 - ▶ Subplan for each contingency.
- ▶ Example 1.5 (A conditional Plan). (AAA $\hat{=}$ ADAC)
[Check(T_1), if $Intact(T_1)$ then $Inflate(T_1)$ else $CallAAA$ fi]
- ▶ Problem: Expensive because it plans for many unlikely cases.
- ▶ Still another Solution: Execution monitoring/replanning
 - ▶ Assume normal states/outcomes, check progress during execution, replan if necessary.
- ▶ Problem: Unanticipated outcomes may lead to failure. (e.g., no AAA card)
- ▶ Observation 1.6. We really need a combination; plan for likely/serious eventualities, deal with others when they arise, as they must eventually.

2 The Furniture Coloring Example

The Furniture-Coloring Example: Specification

► Example 2.1 (Coloring Furniture).

Paint a chair and a table in matching colors.

► The initial state is:

- we have two cans of paint of unknown color,
- the color of the furniture is unknown as well,
- only the table is in the agent's field of view.

► Actions:

- remove lid from can
- paint object with paint from open can.



The Furniture-Coloring Example: PDDL

► Example 2.2 (Formalization in PDDL).

- The PDDL domain file is as expected (actions below)

```
(define (domain furniture-coloring)
  (:predicates (object ?x) (can ?x) (inview ?x) (color ?x ?y))
  ...)
```

The Furniture-Coloring Example: PDDL

► Example 2.2 (Formalization in PDDL).

- The PDDL domain file is as expected (actions below)
- The PDDL problem file has a “free” variable ?c for the (undetermined) joint color.

```
(define (problem tc-coloring)
  (:domain furniture-objects)
  (:objects table chair c1 c2)
  (:init (object table) (object chair) (can c1) (can c2) (inview table))
  (:goal (color chair ?c) (color table ?c)))
```

The Furniture-Coloring Example: PDDL

► Example 2.2 (Formalization in PDDL).

- The PDDL domain file is as expected (actions below)
- The PDDL problem file has a “free” variable $?c$ for the (undetermined) joint color.
- Two action schemata: *remove can lid to open* and *paint with open can*

```
(:action remove-lid
    :parameters (?x)
    :precondition (can ?x)
    :effect (open can))

(:action paint
    :parameters (?x ?y)
    :precondition (and (object ?x) (can ?y) (color ?y ?c) (open ?y))
    :effect (color ?x ?c))
```

has a universal variable $?c$ for the paint action ⇔ we cannot just give paint a color argument in a partially observable environment.

- Sensorless Plan: Open one can, paint chair and table in its color.

The Furniture-Coloring Example: PDDL

► Example 2.2 (Formalization in PDDL).

- The PDDL domain file is as expected (actions below)
- The PDDL problem file has a “free” variable $?c$ for the (undetermined) joint color.
- Two action schemata: *remove can lid to open* and *paint with open can* has a universal variable $?c$ for the paint action ↵ we cannot just give paint a color argument in a partially observable environment.
- **Sensorless Plan:** Open one can, paint chair and table in its color.
- **Note:** Contingent planning can create better plans, but needs perception
- Two percept schemata: *color of an object* and *color in a can*

```
(:percept color
  :parameters (?x ?c)
  :precondition (and (object ?x) (inview ?x)))

(:percept can-color
  :parameters (?x ?c)
  :precondition (and (can ?x) (inview ?x) (open ?x)))
```

To perceive the color of an object, it must be in view, a can must also be open.

Note: In a fully observable world, the percepts would not have preconditions.

The Furniture-Coloring Example: PDDL

► Example 2.2 (Formalization in PDDL).

- The PDDL domain file is as expected (actions below)
- The PDDL problem file has a “free” variable $?c$ for the (undetermined) joint color.
- Two action schemata: *remove can lid to open* and *paint with open can* has a universal variable $?c$ for the paint action ↵ we cannot just give paint a color argument in a partially observable environment.
- Sensorless Plan: Open one can, paint chair and table in its color.
- Note: Contingent planning can create better plans, but needs perception
- Two percept schemata: *color of an object* and *color in a can*
- An action schema: *look at an object* that causes it to come into view.

```
(:action lookat
  :parameters (?x)
  :precond: (and (inview ?y) and (notequal ?x ?y))
  :effect (and (inview ?x) (not (inview ?y))))
```

The Furniture-Coloring Example: PDDL

► Example 2.2 (Formalization in PDDL).

- The PDDL domain file is as expected (actions below)
- The PDDL problem file has a “free” variable ?c for the (undetermined) joint color.
- Two action schemata: *remove can lid to open* and *paint with open can* has a universal variable ?c for the paint action ↵ we cannot just give paint a color argument in a partially observable environment.
- Sensorless Plan: Open one can, paint chair and table in its color.
- Note: Contingent planning can create better plans, but needs perception
- Two percept schemata: *color of an object* and *color in a can*
- An action schema: *look at an object* that causes it to come into view.
- 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

3 Searching/Planning with Non-Deterministic Actions

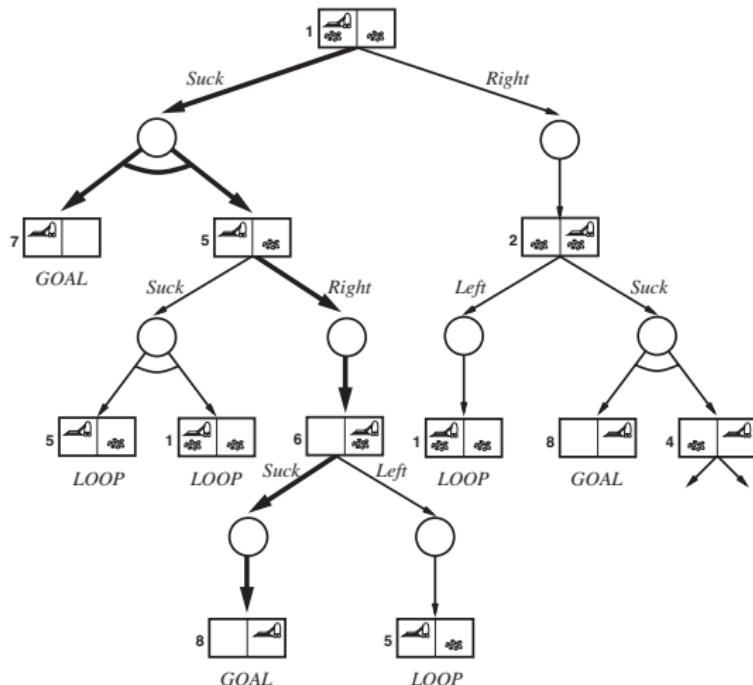
- ▶ **Definition 3.1.** **Conditional plans** extend the possible actions in **plans** by **conditional steps** that execute **sub plans** conditionally whether $K + P \models C$, where $K + P$ is the current knowledge base + the percepts.
- ▶ **Example 3.2.** $[\dots, \text{if } C \text{ then } Plan_A \text{ else } Plan_B \text{ fi}, \dots]$
- ▶ **Definition 3.3.** If the possible percepts are limited to determining the current state in a **conditional plan**, then we speak of a **contingency plan**.
- ▶ **Note:** Need *some plan* for *every possible percept!* Compare to
 - game playing:** *some response* for *every opponent move*.
 - backchaining:** *some rule* such that *every premise satisfied*.
- ▶ **Idea:** Use an AND–OR tree search (**very similar to backward chaining algorithm**)

Contingency Planning: The Erratic Vacuum Cleaner

► Example 3.4 (Erratic vacuum world).

A variant *suck* action:
if square is

- **dirty**: clean the square,
sometimes remove dirt in
adjacent square.
- **clean**: sometimes deposits dirt
on the carpet.



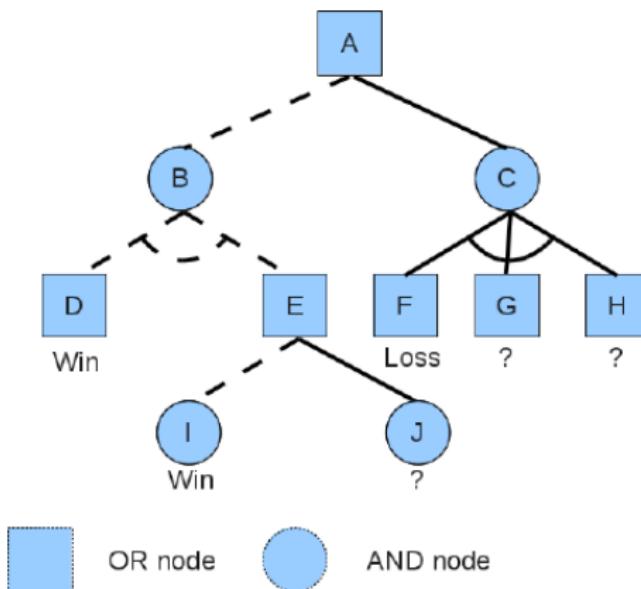
Solution: [suck, if $State = 5$ then [right, suck] else [] fi]

Conditional AND/OR Search (Data Structure)

- ▶ Idea: Use AND/OR trees as data structures for representing problems (or goals) that can be reduced to conjunctions and disjunctions of subproblems (or subgoals).
- ▶ Definition 3.5. An AND/OR graph 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 an AND/OR graph T . A solution that assigns T to a terminal node, iff it is a goal node. Corresponds to a conditional plan.

Conditional AND/OR Search (Example)

- ▶ **Definition 3.6.** An **AND/OR tree** is a **AND/OR graph** that is also a **tree**.
Notation: **AND nodes** are written with arcs connecting the child edges.
- ▶ **Example 3.7 (An AND/OR-tree).**



Conditional AND/OR Search (Algorithm)

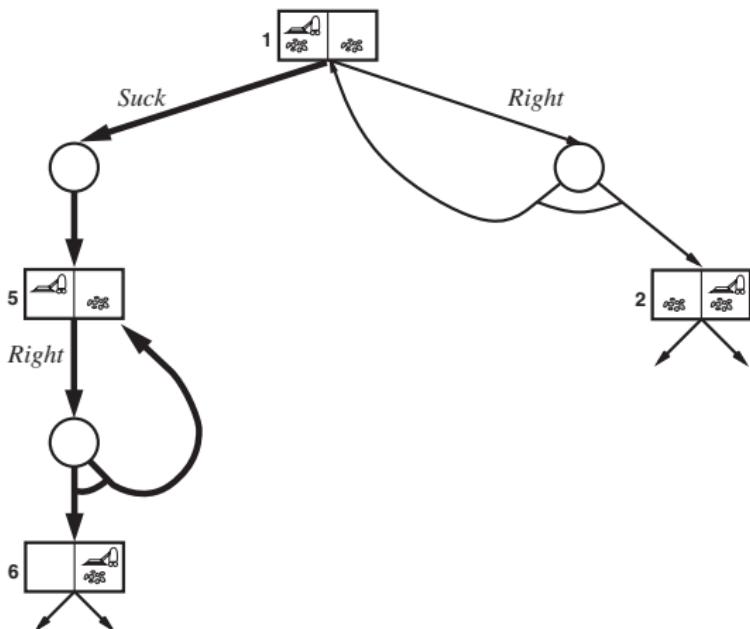
- ▶ **Definition 3.8.** **AND/OR search** is an algorithm for searching AND-OR graphs generated by nondeterministic environments.

```
function AND/OR-GRAPH-SEARCH(prob) returns a conditional plan, or fail
    OR-SEARCH(prob.INITIAL-STATE, prob, [])
function OR-SEARCH(state, prob, path) returns a conditional plan, or fail
    if prob.GOAL-TEST(state) then return the empty plan
    if state is on path then return fail
    for each action in prob.ACTIONS(state) do
        plan := AND-SEARCH(RESULTS(state, action), prob, [state | path])
        if plan ≠ fail then return [action | plan]
    return fail
function AND-SEARCH(states, prob, path) returns a conditional plan, or fail
    for each si in states do
        pi := OR-SEARCH(si, prob, path)
        if pi = fail then return fail
    return [if s1 then p1 else if s2 then p2 else ... if sn-1 then pn-1 else pn]
```

- ▶ **Cycle Handling:** If a state has been seen before \rightsquigarrow **fail**
 - ▶ **fail** does not mean *there is no solution*, but
 - ▶ *if there is a non-cyclic solution, then it is reachable by an earlier incarnation!*

The Slippery Vacuum Cleaner (try, try, try, ... try again)

► Example 3.9 (Slippery Vacuum World).



Moving sometimes fails
~ AND/OR graph

Two possible solutions

(depending on what our plan language allows)

- `[L1 : left, if AtR then L1 else [if CleanL then Ø else suck fi] fi]` or
- `[while AtR do [left] done, if CleanL then Ø else suck fi]`
- We have an “Infinite loop” but `plan` eventually works unless action always fails.

4 Agent Architectures based on Belief States

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!

- ▶ **Problem:** We do not know with certainty what state the world is in!
- ▶ **Idea:** Just keep track of all the possible **states** it could be in.
- ▶ **Definition 4.1.** 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 **transition model** that updates the **belief state** based on **sensor** information and **actions**.

World Models for Uncertainty

- ▶ **Problem:** We do not know with certainty what state the world is in!
- ▶ **Idea:** Just keep track of all the possible **states** it could be in.
- ▶ **Definition 4.1.** 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 **transition model** that updates the **belief state** based on **sensor** information and **actions**.
- ▶ **Idea:** The **agent environment** determines what the world model can be.

- ▶ **Problem:** We do not know with certainty what state the world is in!
- ▶ **Idea:** Just keep track of all the possible **states** it could be in.
- ▶ **Definition 4.1.** 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 **transition model** that updates the **belief state** based on **sensor** information and **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 set (we call its member the **world state**) and the **transition model** is a function from **states** and **actions** to **states**: a **transition function**.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model. What we can do with it depends on representation and inference.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model.
What we can do with it depends on representation and inference.
- ▶ **Search-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ "current state"
 - ▶ no inference.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model.
What we can do with it depends on representation and inference.
- ▶ **Search-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ "current state"
 - ▶ no inference.
- ▶ **CSP-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ constraint network
 - ▶ inference $\hat{=}$ constraint propagation.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model.
What we can do with it depends on representation and inference.
- ▶ **Search-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ "current state"
 - ▶ no inference.
- ▶ **CSP-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ constraint network
 - ▶ inference $\hat{=}$ constraint propagation.
- ▶ **Logic-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ logical formula
 - ▶ inference $\hat{=}$ e.g. DPLL or resolution.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model.
What we can do with it depends on representation and inference.
- ▶ **Search-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ "current state"
 - ▶ no inference.
- ▶ **CSP-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ constraint network
 - ▶ inference $\hat{=}$ constraint propagation.
- ▶ **Logic-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ logical formula
 - ▶ inference $\hat{=}$ e.g. DPLL or resolution.
- ▶ **Planning Agents:** In a **fully observable, deterministic, environment**
 - ▶ **world state** $\hat{=}$ PL0, transition model $\hat{=}$ STRIPS,
 - ▶ inference $\hat{=}$ state/plan space search.

World Models for Complex Environments

- ▶ In a **fully observable**, but **stochastic** environment,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ \leadsto generalize the **transition function** to a **transition relation**.

- ▶ In a **fully observable**, but **stochastic** environment,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ \leadsto generalize the **transition function** to a **transition relation**.
- ▶ Note: This even applies to **online problem solving**, where we can just perceive the **state**.
(e.g. when we want to optimize utility)

World Models for Complex Environments

- ▶ In a **fully observable**, but **stochastic** environment,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ \leadsto generalize the **transition function** to a **transition relation**.
- ▶ Note: This even applies to **online problem solving**, where we can just perceive the **state**.
(e.g. when we want to optimize utility)
- ▶ In a **deterministic**, but **partially observable environment**,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ we can use **transition functions**.
 - ▶ We need a **sensor model**, which predicts the influence of **percepts** on the **belief state**
 - during update.

World Models for Complex Environments

- ▶ In a **fully observable**, but **stochastic** environment,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ \leadsto generalize the **transition function** to a **transition relation**.
- ▶ Note: This even applies to **online problem solving**, where we can just perceive the **state**.
(e.g. when we want to optimize utility)
- ▶ In a **deterministic**, but **partially observable environment**,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ we can use **transition functions**.
 - ▶ We need a **sensor model**, which predicts the influence of **percepts** on the **belief state**
 - during update.
- ▶ In a **stochastic**, **partially observable environment**,
 - ▶ mix the ideas from the last two.
(sensor model + transition relation)

Preview: New World Models (Belief) \leadsto new Agent Types

- ▶ Probabilistic Agents: In a partially observable environment
 - ▶ belief state $\hat{=}$ Bayesian networks,
 - ▶ inference $\hat{=}$ probabilistic inference.

- ▶ Probabilistic Agents: In a partially observable environment
 - ▶ belief state $\hat{=}$ Bayesian networks,
 - ▶ inference $\hat{=}$ probabilistic inference.
- ▶ Decision-Theoretic Agents: In a partially observable, stochastic environment
 - ▶ belief state + transition model $\hat{=}$ decision networks,
 - ▶ inference $\hat{=}$ maximizing expected utility.
- ▶ We will study them in detail in the coming semester.

5 Searching/Planning without Observations

Conformant/Sensorless Planning

- ▶ **Definition 5.1.** Conformant or sensorless planning tries to find plans that work without any sensing. (not even the initial state)
- ▶ **Example 5.2 (Sensorless Vacuum Cleaner World).**



States	integer dirt and robot locations
Actions	<i>left, right, suck, noOp</i>
Goal test	<i>notdirty?</i>

- ▶ **Observation 5.3.** In a sensorless world we do not know the initial state.(or any state after)
- ▶ **Observation 5.4.** Sensorless planning must search in the space of belief states (sets of possible actual states).
- ▶ **Example 5.5 (Searching the Belief State Space).**

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

Search in the Belief State Space: Let's Do the Math

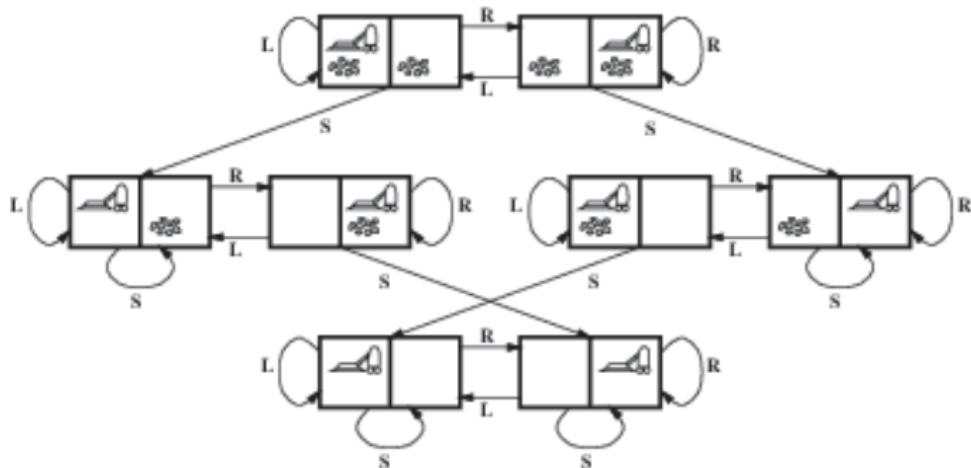
- ▶ **Recap:** We describe an agent problem $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ via its **states** \mathcal{S} , actions \mathcal{A} , and **transition model** $\mathcal{T}: \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, **goal states** \mathcal{G} , and **initial state** \mathcal{I} .
- ▶ **Problem:** What is the corresponding sensorless problem?
- ▶ **Let's think:** Let $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ be a (physical) problem
 - ▶ **States** \mathcal{S}^b : The **belief states** are the $2^{|\mathcal{S}|}$ subsets of \mathcal{S} .
 - ▶ The **initial state** \mathcal{I}^b is just \mathcal{S} (no information)
 - ▶ **Goal states** $\mathcal{G}^b := \{S \in \mathcal{S}^b \mid S \subseteq \mathcal{G}\}$ (all possible states must be physical goal states)
 - ▶ **Actions** \mathcal{A}^b : we just take \mathcal{A} . (that's the point!)
 - ▶ **Transition model** $\mathcal{T}^b: \mathcal{A}^b \times \mathcal{S}^b \rightarrow \mathcal{P}(\mathcal{A}^b)$: 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 5.6.** In belief-state space the problem is always fully observable!

State Space vs. Belief State Space

► Example 5.7 (State/Belief State Space in the Vacuum World).

In the vacuum world all actions are always applicable

(1./2. equal)

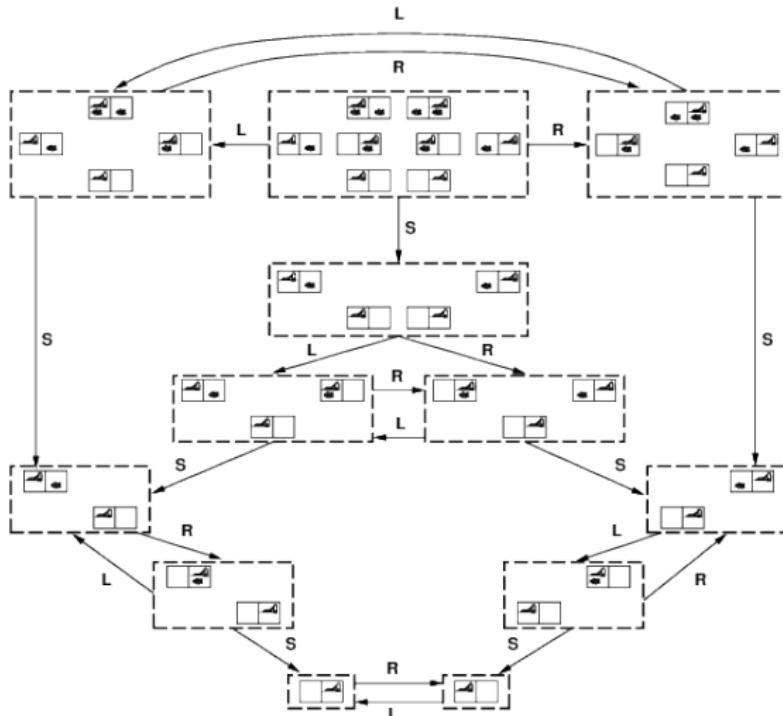


State Space vs. Belief State Space

► Example 5.7 (State/Belief State Space in the Vacuum World).

In the vacuum world all actions are always applicable

(1./2. equal)

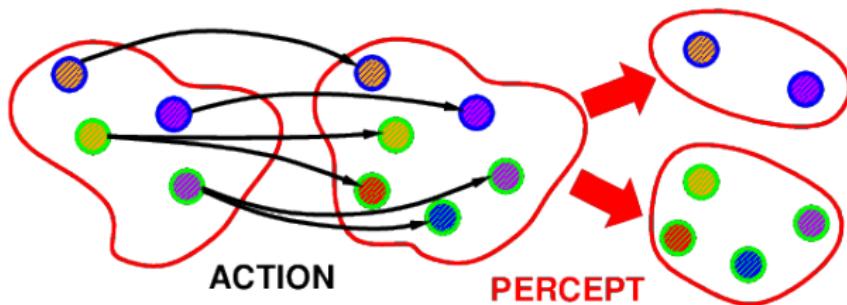


- ▶ **Upshot:** We can build belief-space problem formulations automatically,
 - ▶ but they are exponentially bigger in theory, in practice they are often similar;
 - ▶ e.g. 12 reachable **belief states** out of $2^8 = 256$ for vacuum example.
- ▶ **Problem:** **Belief states** are HUGE; e.g. initial **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!**

6 Searching/Planning with Observation

Conditional planning (Motivation)

- ▶ Note: So far, we have never used the agent's sensors.
 - ▶ In *General Problem Solving via Search*, since the environment was observable and deterministic we could just use *offline* planning.
 - ▶ In *Conformant Planning* 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

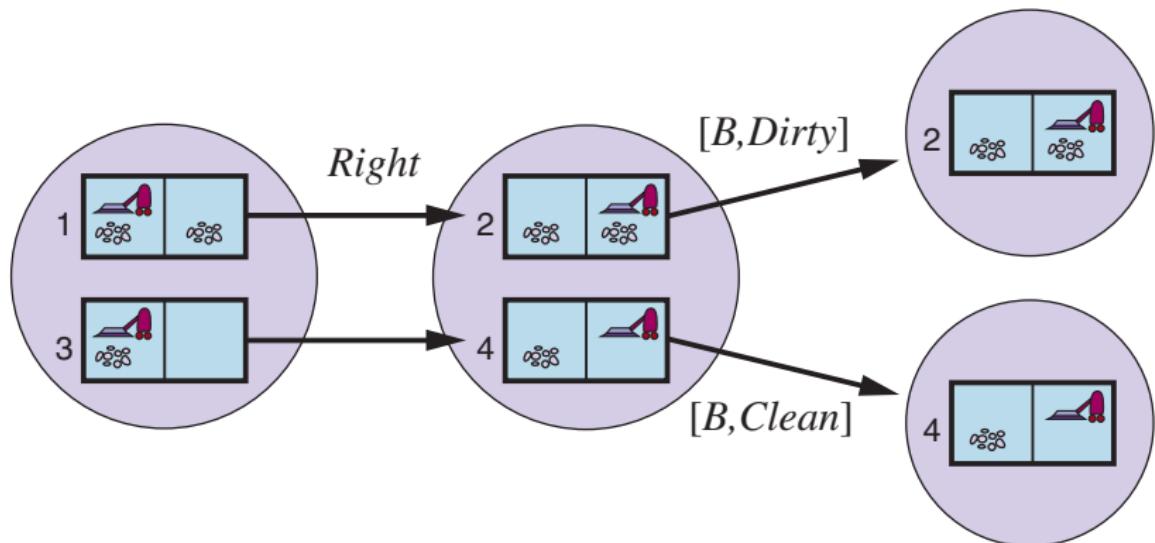
- ▶ We extend the ideas from slide 647 to include partial observability.
- ▶ **Definition 6.1.** Given a (physical) sproblem $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, we define the **belief state search problem** induced by \mathcal{P} to be $\langle \mathcal{P}(\mathcal{S}), \mathcal{A}, \mathcal{T}^b, \mathcal{S}, \{S \in \mathcal{S}^b \mid S \subseteq \mathcal{G}\} \rangle$, 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 $\hat{b} := \text{PRED}(b, a)$ for some function $\text{PRED} : \mathcal{P}(\mathcal{S}) \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$.
 - ▶ The **observation prediction** stage determines the set of possible percepts that could be observed in the predicted belief state: $\text{PossPERC}(\hat{b}) = \{\text{PERC}(s) \mid s \in \hat{b}\}$.
 - ▶ The **update** stage determines, for each possible percept, the resulting belief state: $\text{UPDATE}(\hat{b}, o) := \{s \mid o = \text{PERC}(s) \text{ and } s \in \hat{b}\}$

The functions **PRED** and **PERC** are the main parameters of this model. We define $\text{RESULT}(b, a) := \{\text{UPDATE}(\text{PRED}(b, a), o) \mid \text{PossPERC}(\text{PRED}(b, a))\}$

- ▶ **Observation 6.2.** We always have $\text{UPDATE}(\hat{b}, o) \subseteq \hat{b}$.
- ▶ **Observation 6.3.** If sensing is deterministic, belief states for different possible percepts are disjoint, forming a partition of the original predicted belief state.

Example: Local Sensing Vacuum Worlds

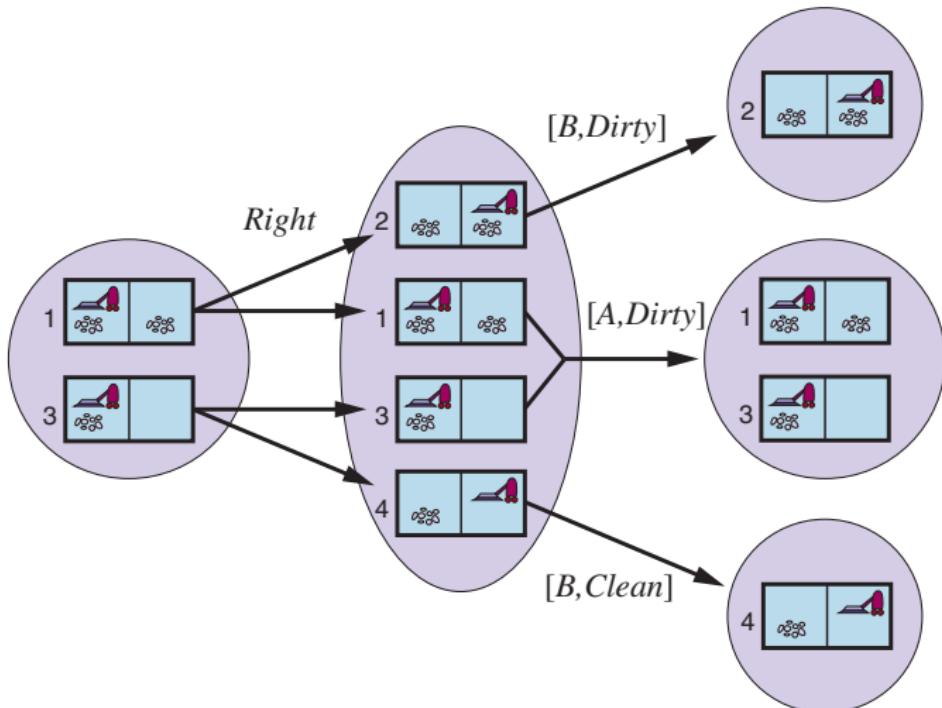
► Example 6.4 (Transitions in the Vacuum World). Deterministic World:



The action *Right* is deterministic, sensing disambiguates to singletons

Example: Local Sensing Vacuum Worlds

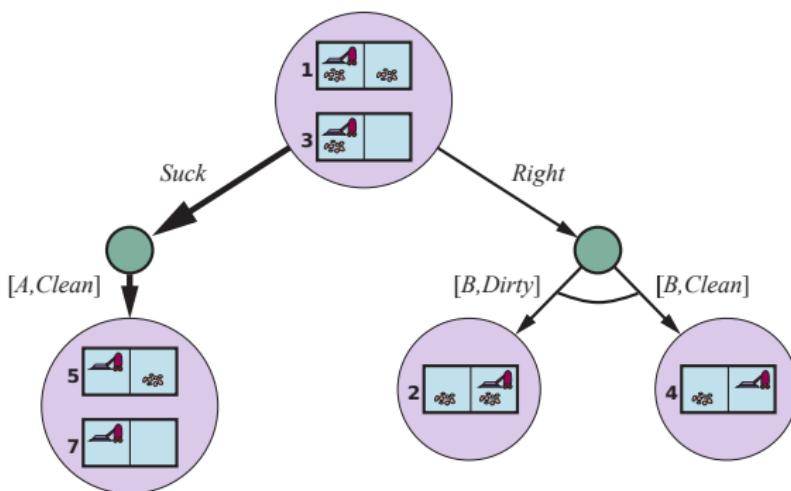
► Example 6.4 (Transitions in the Vacuum World). Slippery World:



The action *Right* is non-deterministic, sensing disambiguates somewhat

Belief-State Search with Percepts

- ▶ **Observation:** The belief-state transition model induces an AND/OR graph.
- ▶ **Idea:** Use AND/OR search in non-deterministic environments.
- ▶ **Example 6.5.** AND/OR graph for initial percept $[A, Dirty]$.



Solution: $[Suck, Right, \text{if } B\text{state} = \{6\} \text{ then } Suck \text{ else } [] \text{ fi}]$

- ▶ **Note:** Belief-state-problem \leadsto conditional step tests on belief-state percept (plan would not be executable in a partially observable environment otherwise)

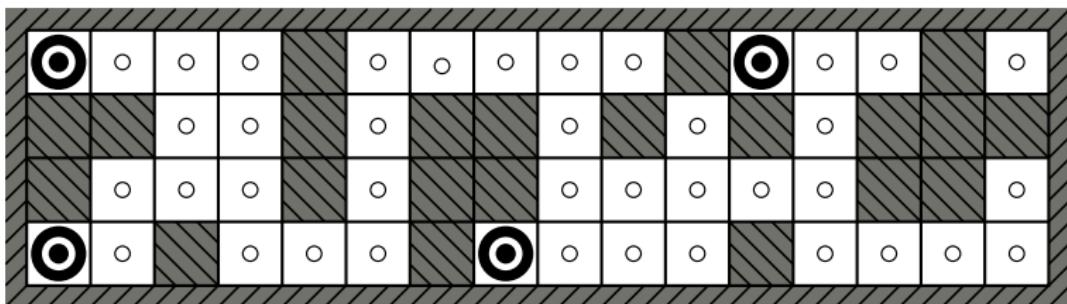
Example: Agent Localization

- ▶ **Example 6.6.** 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 $\sim \hat{b}_1$ all possible locations.

Example: Agent Localization

► **Example 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 $\sim \hat{b}_1$ all possible locations.
2. Initial percept: NWS (walls north, west, and south) $\sim \hat{b}_2 = \text{UPDATE}(\hat{b}_1, \text{NWS})$



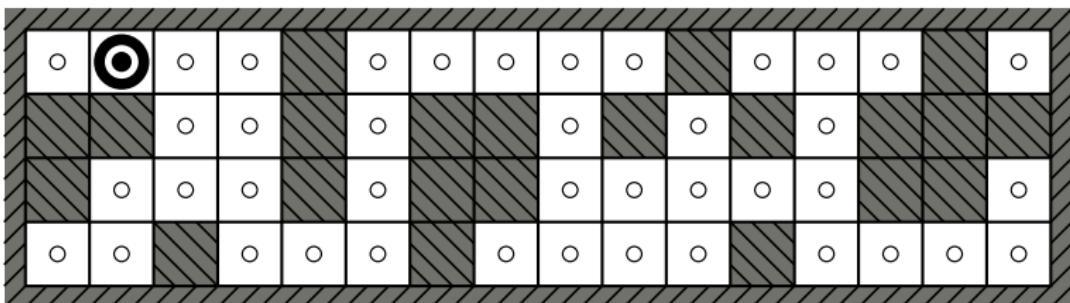
Example: Agent Localization

- ▶ **Example 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 $\sim \hat{b}_1$ all possible locations.
 2. Initial percept: *NWS* (walls north, west, and south) $\sim \hat{b}_2 = \text{UPDATE}(\hat{b}_1, \text{NWS})$
 3. Agent executes *Move* $\sim \hat{b}_3 = \text{PRED}(\hat{b}_2, \text{Move}) = \text{one step away from these.}$

Example: Agent Localization

► **Example 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 $\sim \hat{b}_1$ all possible locations.
2. Initial percept: NWS (walls north, west, and south) $\sim \hat{b}_2 = \text{UPDATE}(\hat{b}_1, \text{NWS})$
3. Agent executes *Move* $\sim \hat{b}_3 = \text{PRED}(\hat{b}_2, \text{Move}) = \text{one step away from these.}$
4. Next percept: NS $\sim \hat{b}_4 = \text{UPDATE}(\hat{b}_3, \text{NS})$



Example: Agent Localization

- ▶ **Example 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 $\sim \hat{b}_1$ all possible locations.
2. Initial percept: NWS (walls north, west, and south) $\sim \hat{b}_2 = \text{UPDATE}(\hat{b}_1, \text{NWS})$
3. Agent executes *Move* $\sim \hat{b}_3 = \text{PRED}(\hat{b}_2, \text{Move}) = \text{one step away from these.}$
4. Next percept: NS $\sim \hat{b}_4 = \text{UPDATE}(\hat{b}_3, \text{NS})$

All in all, $\hat{b}_4 = \text{UPDATE}(\text{PRED}(\text{UPDATE}(\hat{b}_1, \text{NWS}), \text{Move}), \text{NS})$ localizes the agent.

- ▶ **Observation:** **PRED** enlarges the belief state, while **UPDATE** shrinks it again.

Contingent Planning

- ▶ **Definition 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 6.8.** Continuing 2.1.

One of the possible **contingent plan** is

((lookat table) (lookat chair))

```
(if (and (color table c) (color chair c)) (noop)
    ((removelid c1) (lookat c1) (removelid c2) (lookat c2)
     (if (and (color table c) (color can c)) ((paint chair can))
         (if (and (color chair c) (color can c)) ((paint table can))
             ((paint chair c1) (paint table c1)))))))
```

- ▶ **Note:** Variables in this plan are existential; e.g. in
 - ▶ line 2: If there is some joint color c of the table and chair \rightsquigarrow done.
 - ▶ line 4/5: Condition can be satisfied by $[c_1/can]$ or $[c_2/can] \rightsquigarrow$ instantiate accordingly.
- ▶ **Definition 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 \hat{b} is computed based on prediction with the action a and the refinement with the percept p .
- ▶ **Here:** Given an action a and percepts $p = p_1 \wedge \dots \wedge p_n$, we have
 - ▶ $\hat{b} = (b \setminus \text{del}_a) \cup \text{add}_a$ (as for the sensorless agent)
 - ▶ If $n = 1$ and $(\text{:percept } p_1 : \text{precondition } c)$ is the only percept axiom, also add p and c to \hat{b} . (add c as otherwise p impossible)
 - ▶ If $n > 1$ and $(\text{:percept } p_i : \text{precondition } c_i)$ are the percept axioms, also add p and $c_1 \vee \dots \vee c_n$ to \hat{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.

7 Online Search

- ▶ 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** ↪ exploration problem.

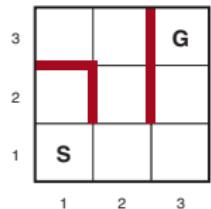
- ▶ Observation: Online problem solving even makes sense in deterministic, fully observable environments.
- ▶ **Definition 7.1.** A **online search problem** consists of a set S of states, and
 - ▶ a function $\text{Actions}(s)$ that returns a list of actions allowed in state s .
 - ▶ the step cost function 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 7.2.** The **competitive ratio** of an **online problem solving agent** is the quotient of
 - ▶ **offline performance**, i.e. cost of optimal solutions with full information and
 - ▶ **online performance**, i.e. the actual cost induced by **online problem solving**.

Online Search Problems (Example)

► Example 7.3 (A simple maze problem).

The agent starts at S and must reach G but knows nothing of the environment. In particular not that

- Up(1,1) results in (1,2) and
- Down(1,1) results in (1,1) (i.e. back)



Online Search Obstacles (Dead Ends)

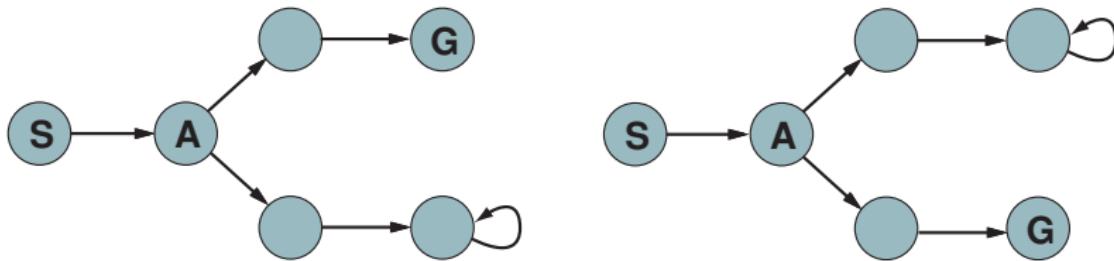
- ▶ **Definition 7.4.** We call a state a **dead end**, iff no state is reachable from it by an action. An action that leads to a **dead end** is called **irreversible**.
- ▶ **Note:** With **irreversible** actions the **competitive ratio** can be infinite.

Online Search Obstacles (Dead Ends)

- ▶ **Definition 7.4.** 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 7.5.** No **online** algorithm can avoid **dead ends** in all state spaces.

Online Search Obstacles (Dead Ends)

- ▶ **Definition 7.4.** We call a state a **dead end**, iff no state is reachable from it by an action. An action that leads to a dead end is called **irreversible**.
- ▶ **Note:** With **irreversible** actions the **competitive ratio** can be infinite.
- ▶ **Observation 7.5.** No **online** algorithm can avoid **dead ends** in all state spaces.
- ▶ **Example 7.6.** Two state spaces that lead an online agent into **dead ends**:

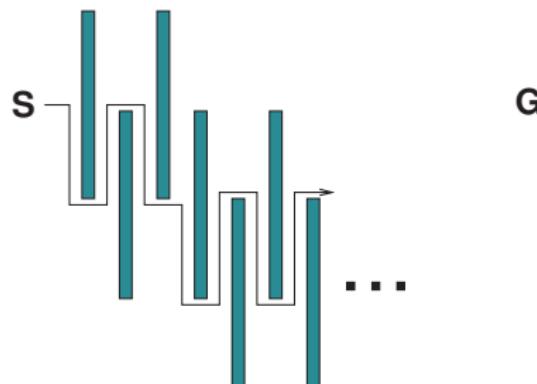


Any agent will fail in at least one of the spaces.

- ▶ **Definition 7.7.** We call 7.6 an **adversary argument**.

Online Search Obstacles (Dead Ends)

- ▶ **Definition 7.4.** We call a state a **dead end**, iff no state is reachable from it by an action. An action that leads to a dead end is called **irreversible**.
- ▶ **Note:** With **irreversible** actions the **competitive ratio** can be infinite.
- ▶ **Observation 7.5.** No **online** algorithm can avoid **dead ends** in all state spaces.
- ▶ **Example 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 7.7.** We call 7.6 an **adversary argument**.
- ▶ **Example 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

- ▶ **Definition 7.4.** We call a state a **dead end**, iff no state is reachable from it by an action. An action that leads to a **dead end** is called **irreversible**.
- ▶ **Note:** With **irreversible** actions the **competitive ratio** can be infinite.
- ▶ **Observation 7.5.** No **online** algorithm can avoid **dead ends** in all state spaces.
- ▶ **Example 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 7.7.** We call 7.6 an **adversary argument**.
- ▶ **Example 7.8.** Forcing an online agent into an arbitrarily inefficient route:
- ▶ **Observation:** **Dead ends** are a real problem for robots: ramps, stairs, cliffs, ...
- ▶ **Definition 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.

- ▶ **Observation:** Online and offline search algorithms differ considerably:
 - ▶ For an offline agent, the environment is visible a priori.
 - ▶ An online agent builds a “map” of the environment from percepts in visited states.
Therefore, e.g. A^* can expand any node in the fringe, but an online agent must go there to explore it.
- ▶ **Intuition:** It seems best to expand nodes in “local order” to avoid spurious travel.
- ▶ **Idea:** Depth-first search seems a good fit. (must only travel for backtracking)

Online DFS Search Agent

► Definition 7.10. The :

```
function ONLINE-DFS-AGENT( $s'$ ) returns an action
  inputs:  $s'$ , a percept that identifies the current state
  persistent: result, a table mapping  $(s, a)$  to  $s'$ , initially empty
              untried, a table mapping  $s$  to a list of untried actions
              unbacktracked, a table mapping  $s$  to a list backtracks not tried
               $s, a$ , the previous state and action, initially null
  if Goal-Test( $s'$ ) then return stop
  if  $s' \notin \text{untried}$  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.

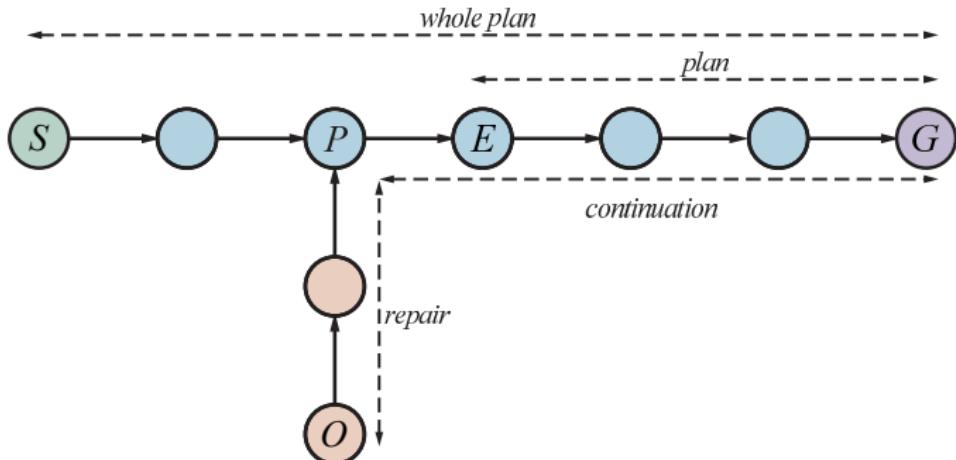
8 Replanning and Execution Monitoring

Replanning (Ideas)

- ▶ **Idea:** We can turn a **planner** P into an **online problem solver** by adding an action $\text{RePlan}(g)$ without preconditions that re-starts P in the current state with goal g .
- ▶ **Observation:** **Replanning** induces a tradeoff between pre-planning and re-planning.
- ▶ **Example 8.1.** The plan $[\text{RePlan}(g)]$ is a (trivially) complete plan for any goal g . **(not helpful)**
- ▶ **Example 8.2.** A plan with sub-plans for every contingency (e.g. what to do if a meteor strikes) may be too costly/large. **(wasted effort)**
- ▶ **Example 8.3.** But when a tire blows while driving into the desert, we want to have water pre-planned. **(due diligence against catastrophes)**
- ▶ **Observation:** In **stochastic or partially observable environments** we also need some form of execution monitoring to determine the need for replanning (plan repair).

Replanning for Plan Repair

- Generally: Replanning when the agent's model of the world is incorrect.
- Example 8.4 (Plan Repair by Replanning). Given a plan from S to G .



- The agent executes *wholeplan step by step*, monitoring the rest (*plan*). (green checkmark)
- After a few *steps* the agent expects to be in E , but observes state O . (green checkmark)
- **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*

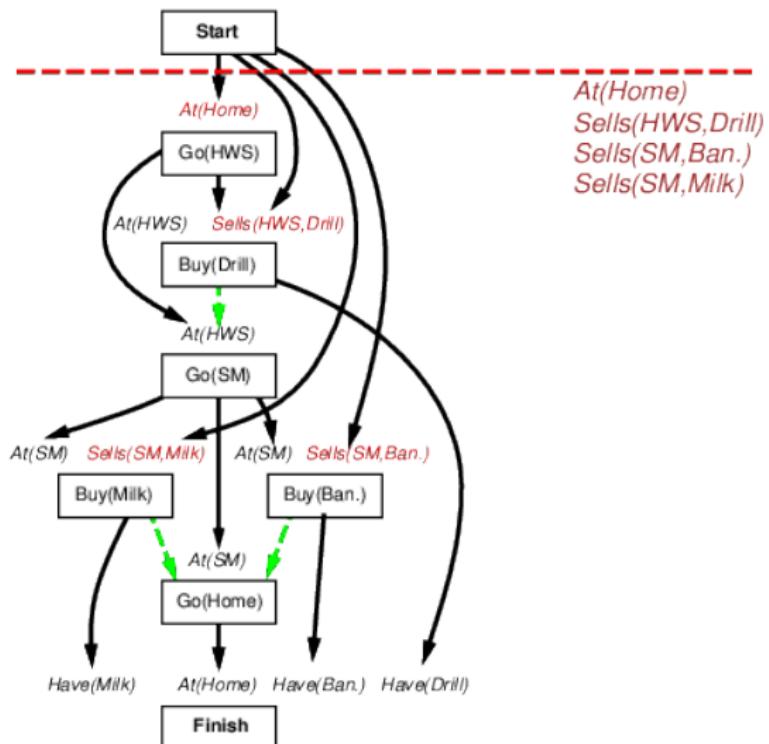
Factors in World Model Failure \leadsto Monitoring

- ▶ **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 \leadsto no color)
 - ▶ no provisions for exogenous events (someone knocks over a paint can)
- ▶ **Observation:** Without a way for monitoring for these, planning is very brittle.
- ▶ **Definition 8.5.** There are three levels of **execution monitoring**: before executing an action
 - ▶ **action monitoring** checks whether all preconditions still hold.
 - ▶ **plan monitoring** checks that the remaining plan will still succeed.
 - ▶ **goal monitoring** checks whether there is a better set of goals it could try to achieve.
- ▶ **Note:** 8.4 was a case of **action monitoring** leading to replanning.

- ▶ **Problem:** Need to upgrade planning 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
 - ⊇ all preconditions of remaining steps not achieved by remaining steps
 - ⊇ all causal link “crossing current time point”
- ▶ **Idea:** On failure, resume planning (e.g. by POP) to achieve open conditions from current state.
- ▶ **Definition 8.6. IPEM (Integrated Planning, Execution, and Monitoring):**
 - ▶ keep updating *Start* to match current state
 - ▶ links from actions replaced by links from *Start* when done

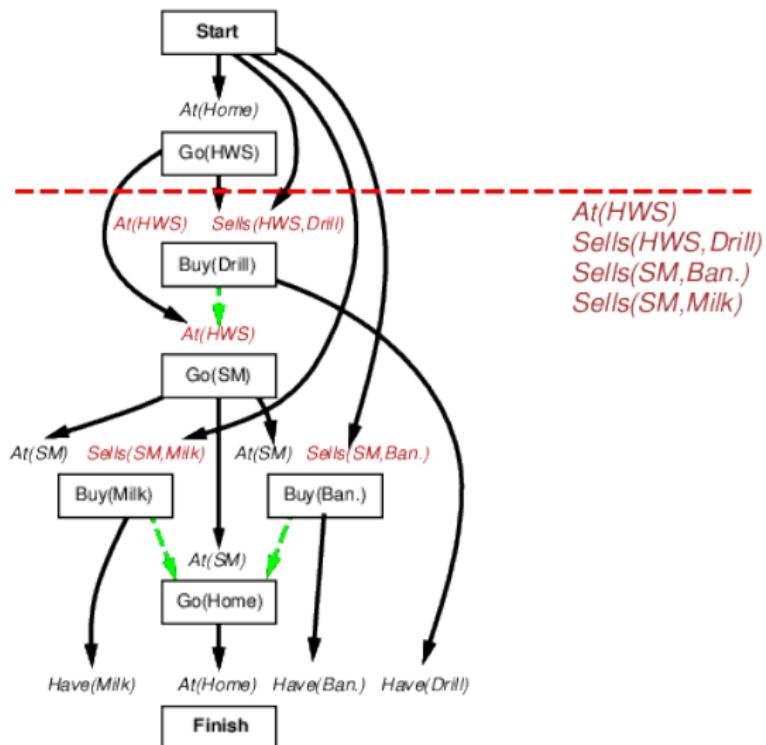
Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



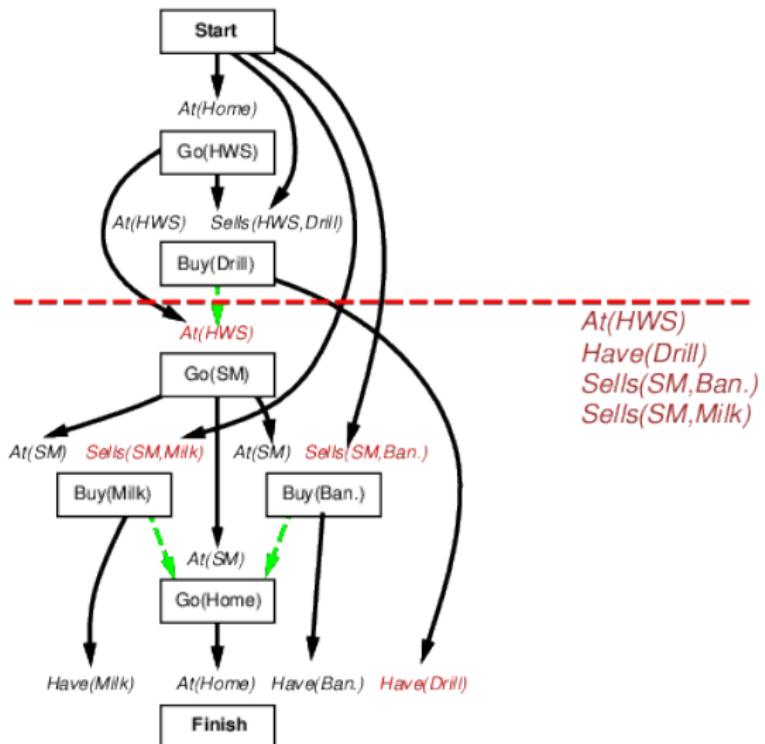
Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



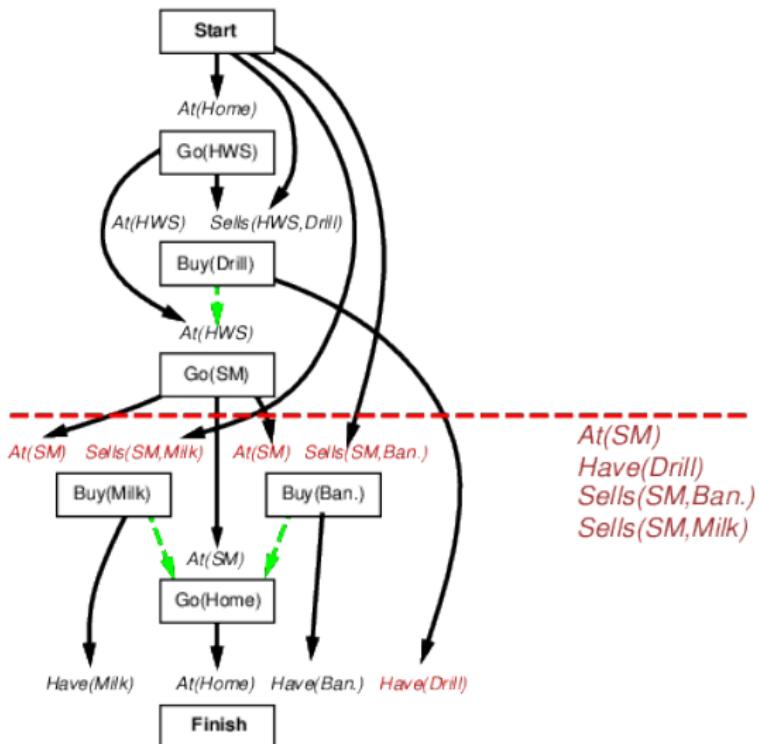
Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



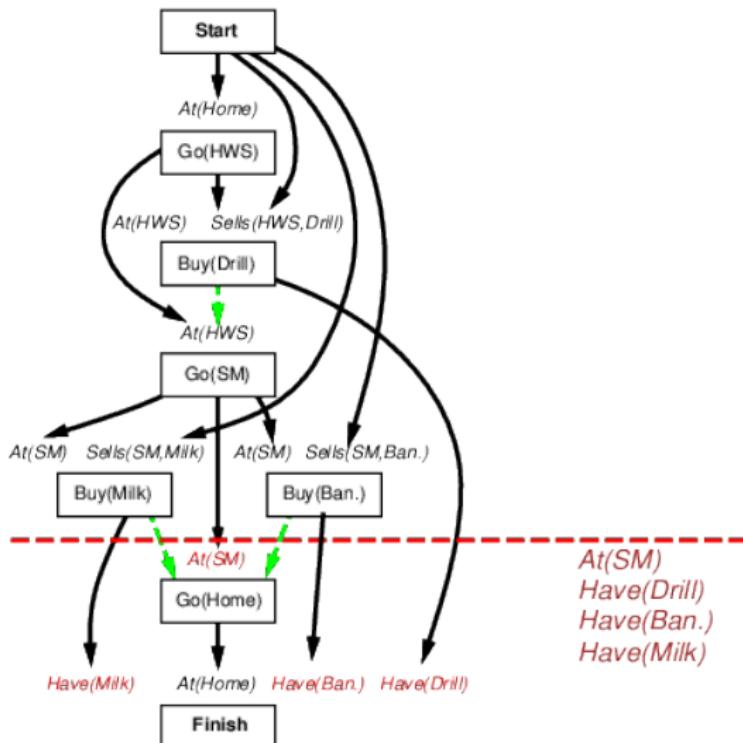
Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



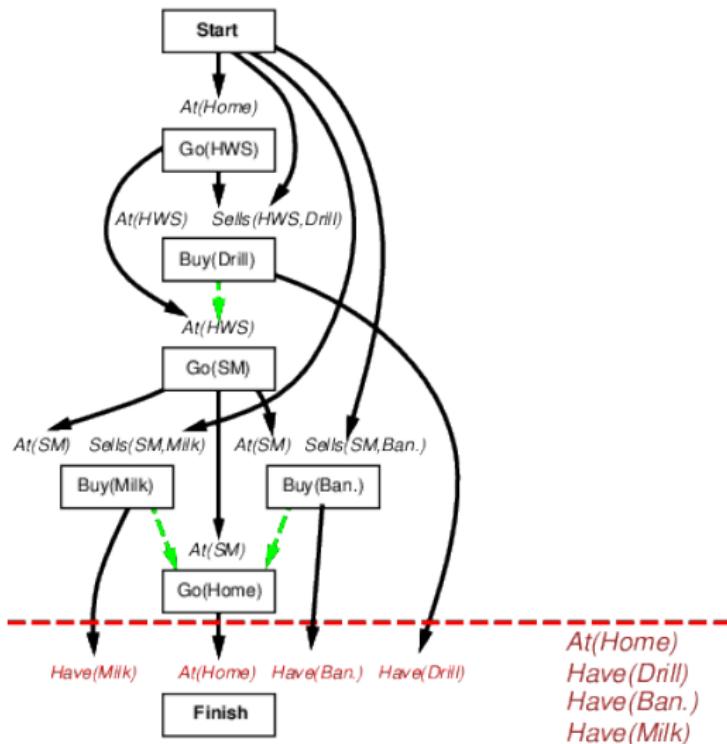
Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



Chapter 19 Semester Change-Over

1 What did we learn in AI 1?

Topics of AI-1 (Winter Semester)

- ▶ Getting Started
 - ▶ What is Artificial Intelligence
 - ▶ Logic Programming in Prolog
 - ▶ Intelligent Agents
- ▶ Problem Solving
 - ▶ Problem Solving and Search
 - ▶ Adversarial Search (Game playing)
 - ▶ Constraint Satisfaction Problems
- ▶ Knowledge and Reasoning
 - ▶ Formal Logic as the Mathematics of Meaning
 - ▶ Propositional Logic and Satisfiability
 - ▶ First-Order Logic and Theorem Proving
 - ▶ Logic Programming
 - ▶ Description Logics and Semantic Web
- ▶ Planning
 - ▶ Planning
 - ▶ Planning and Acting in the real world

(situating ourselves)

(An influential paradigm)

(a unifying framework)

(Black Box World States and Actions)

(A nice application of Search)

(Factored World States)

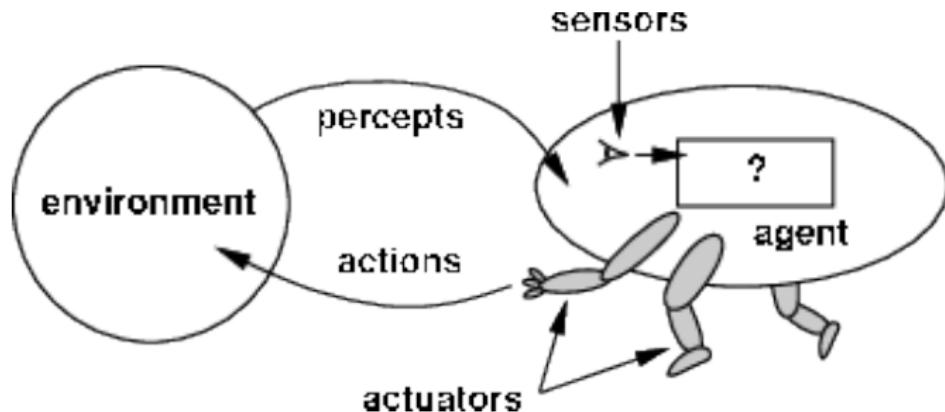
(Atomic Propositions)

(Quantification)

(Logic + Search \leadsto Programming)

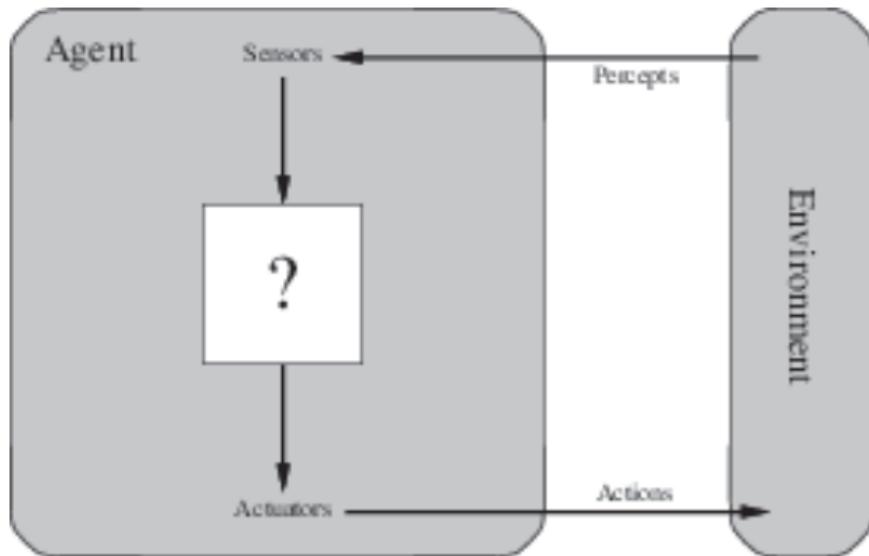
Rational Agents as an Evaluation Framework for AI

- Agents interact with the environment

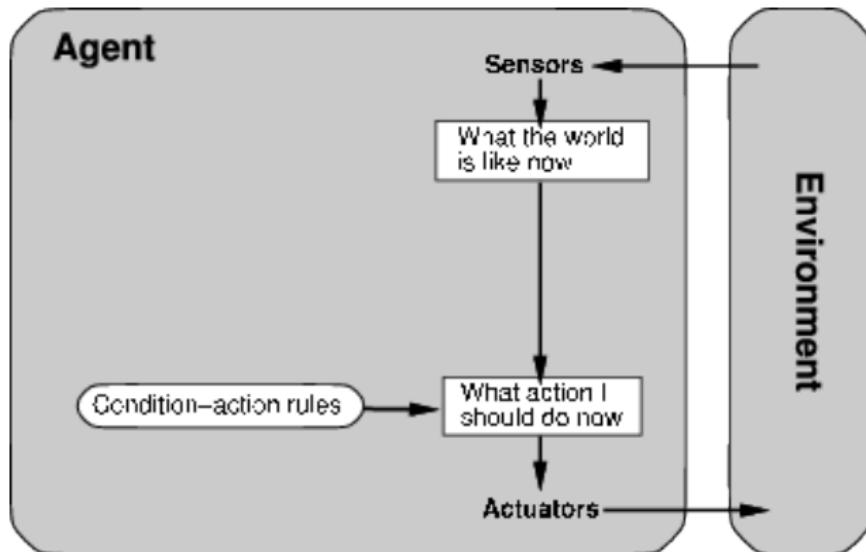


Rational Agents as an Evaluation Framework for AI

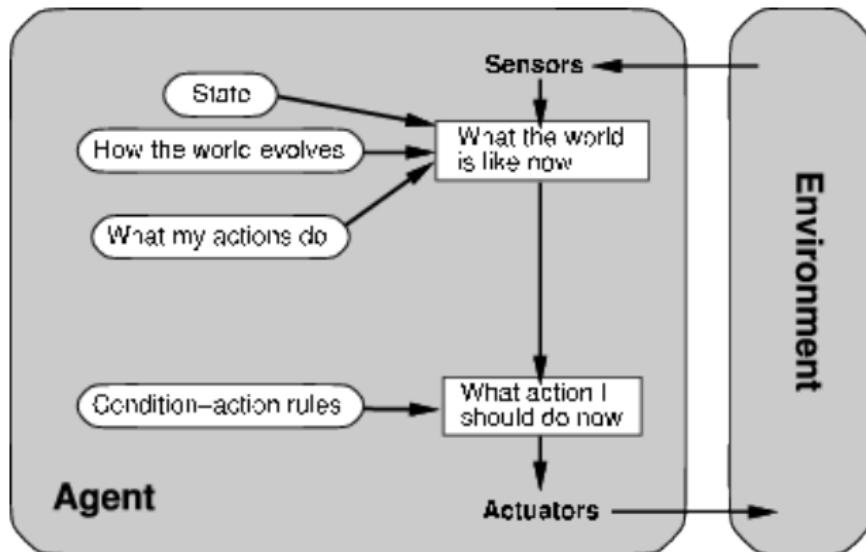
- ▶ General agent schema



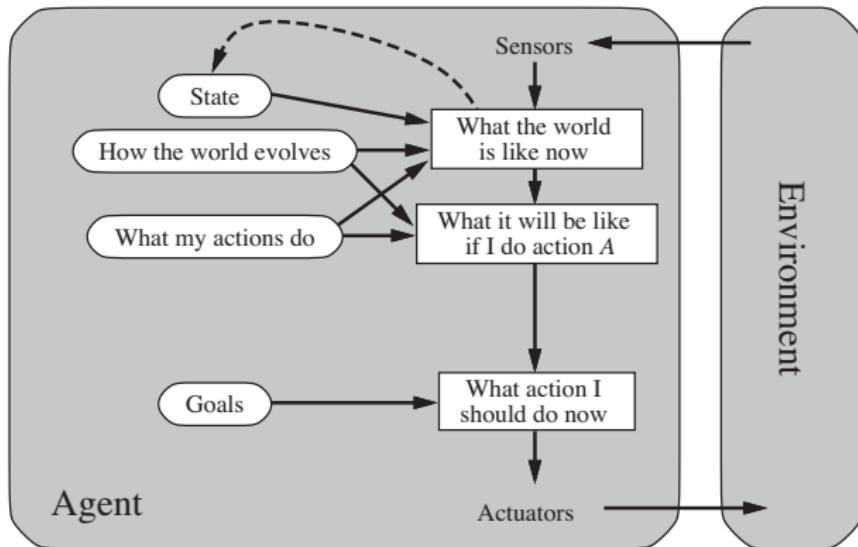
- ▶ Simple Reflex Agents



► Reflex Agents with State

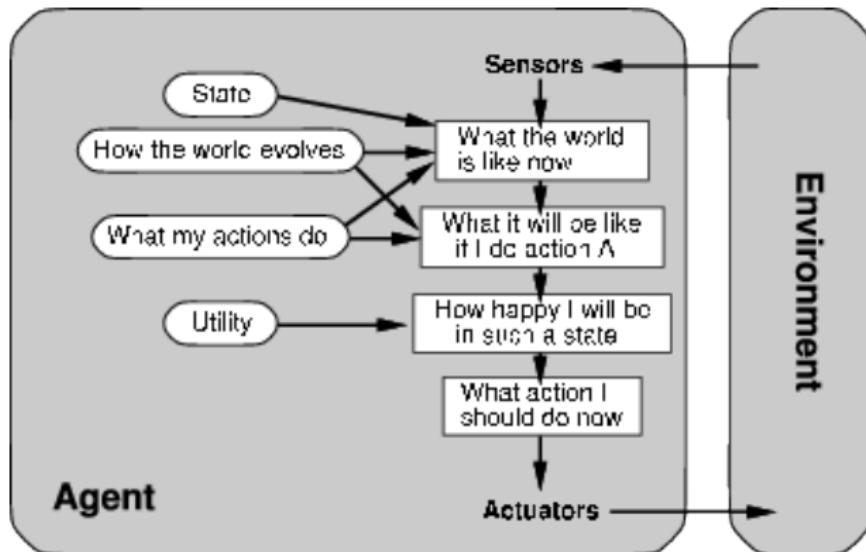


► Goal-Based Agents

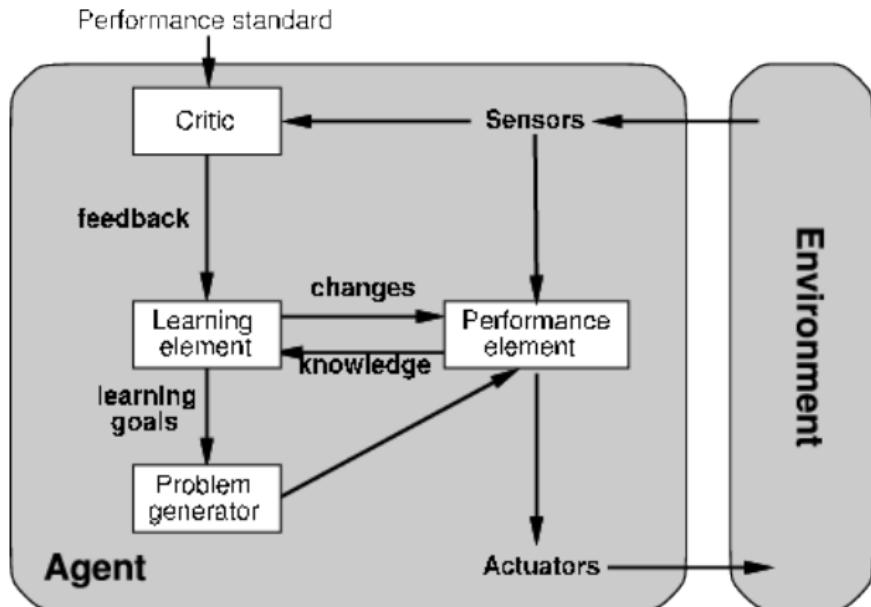


Rational Agents as an Evaluation Framework for AI

► Utility-Based Agent



► Learning Agents



- ▶ Idea: Try to design agents that are successful (do the right thing)
- ▶ Definition 1.1. An agent is called rational, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date. This is called the MEU principle.
- ▶ Note: A rational agent need not be perfect
 - ▶ only needs to maximize expected value (Rational \neq omniscient)
 - ▶ need not predict e.g. very unlikely but catastrophic events in the future
 - ▶ percepts may not supply all relevant information (Rational \neq clairvoyant)
 - ▶ if we cannot perceive things we do not need to react to them.
 - ▶ but we may need to try to find out about hidden dangers (exploration)
 - ▶ action outcomes may not be as expected (rational \neq successful)
 - ▶ but we may need to take action to ensure that they do (more often) (learning)
- ▶ Rational \leadsto exploration, learning, autonomy

Symbolic AI: Adding Knowledge to Algorithms

- ▶ Problem Solving (Black Box States, Transitions, Heuristics)
- ▶ Framework: Problem Solving and Search (basic tree/graph walking)
- ▶ Variant: Game playing (Adversarial Search) (Minimax + $\alpha\beta$ -Pruning)

Symbolic AI: Adding Knowledge to Algorithms

- ▶ Problem Solving (Black Box States, Transitions, Heuristics)
 - ▶ Framework: Problem Solving and Search (basic tree/graph walking)
 - ▶ Variant: Game playing (Adversarial Search) (Minimax + $\alpha\beta$ -Pruning)
- ▶ Constraint Satisfaction Problems (heuristic search over partial assignments)
 - ▶ States as partial variable assignments, transitions as assignment
 - ▶ Heuristics informed by current restrictions, constraint graph
 - ▶ Inference as constraint propagation (transferring possible values across arcs)

Symbolic AI: Adding Knowledge to Algorithms

- ▶ Problem Solving (Black Box States, Transitions, Heuristics)
 - ▶ Framework: Problem Solving and Search (basic tree/graph walking)
 - ▶ Variant: Game playing (Adversarial Search) (Minimax + $\alpha\beta$ -Pruning)
- ▶ Constraint Satisfaction Problems (heuristic search over partial assignments)
 - ▶ States as partial variable assignments, transitions as assignment
 - ▶ Heuristics informed by current restrictions, constraint graph
 - ▶ Inference as constraint propagation (transferring possible values across arcs)
- ▶ Describing world states by formal language (and drawing inferences)
 - ▶ Propositional Logic and DPLL (deciding entailment efficiently)
 - ▶ First-Order Logic and ATP (reasoning about infinite domains)
 - ▶ Digression: Logic Programming (logic + search)
 - ▶ Description Logics as moderately expressive, but decidable logics

Symbolic AI: Adding Knowledge to Algorithms

- ▶ Problem Solving (Black Box States, Transitions, Heuristics)
 - ▶ Framework: Problem Solving and Search (basic tree/graph walking)
 - ▶ Variant: Game playing (Adversarial Search) (Minimax + $\alpha\beta$ -Pruning)
- ▶ Constraint Satisfaction Problems (heuristic search over partial assignments)
 - ▶ States as partial variable assignments, transitions as assignment
 - ▶ Heuristics informed by current restrictions, constraint graph
 - ▶ Inference as constraint propagation (transferring possible values across arcs)
- ▶ Describing world states by formal language (and drawing inferences)
 - ▶ Propositional Logic and DPLL (deciding entailment efficiently)
 - ▶ First-Order Logic and ATP (reasoning about infinite domains)
 - ▶ Digression: Logic Programming (logic + search)
 - ▶ Description Logics as moderately expressive, but decidable logics
- ▶ Planning: Problem Solving using white-box world/action descriptions
 - ▶ Framework: describing world states in logic as sets of propositions and actions by preconditions and add/delete lists
 - ▶ Algorithms: e.g heuristic search by problem relaxations

- ▶ 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

Artificial Intelligence 2

Prof. Dr. Michael Kohlhase

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

2022-05-04

2 Administrativa

Prerequisites for AI-2

- ▶ **Content Prerequisites:** the mandatory courses in CS@FAU; Sem 1-4, in particular:
 - ▶ course “Mathematik C4” (InfMath4).
 - ▶ (very) elementary complexity theory. (big-Oh and friends)
- also AI-1 (“Artificial Intelligence I”) (of course)

Prerequisites for AI-2

- ▶ **Content Prerequisites:** the mandatory courses in CS@FAU; Sem 1-4, in particular:
 - ▶ course “Mathematik C4” (InfMath4).
 - ▶ (very) elementary complexity theory. (big-Oh and friends)
 - also AI-1 (“Artificial Intelligence I”) (of course)
- ▶ **Intuition:** (take them with a kilo of salt)
 - ▶ This is what I assume you know! (I have to assume something)
 - ▶ In many cases, the dependency of AI-2 on these is partial and “in spirit”.
 - ▶ If you have not taken these (or do not remember), read up on them as needed!

Prerequisites for AI-2

- ▶ **Content Prerequisites:** the mandatory courses in CS@FAU; Sem 1-4, in particular:
 - ▶ course “Mathematik C4” (InfMath4).
 - ▶ (very) elementary complexity theory. (big-Oh and friends)
 - also AI-1 (“Artificial Intelligence I”) (of course)
- ▶ **Intuition:** (take them with a kilo of salt)
 - ▶ This is what I assume you know! (I have to assume something)
 - ▶ In many cases, the dependency of AI-2 on these is partial and “in spirit”.
 - ▶ If you have not taken these (or do not remember), read up on them as needed!
- ▶ **The real Prerequisite:** Motivation, Interest, Curiosity, hard work. (AI-2 is non-trivial)

- ▶ **Content Prerequisites:** the mandatory courses in CS@FAU; Sem 1-4, in particular:
 - ▶ course “Mathematik C4” (InfMath4).
 - ▶ (very) elementary complexity theory. (big-Oh and friends)
 - also AI-1 (“Artificial Intelligence I”) (of course)
- ▶ **Intuition:** (take them with a kilo of salt)
 - ▶ This is what I assume you know! (I have to assume something)
 - ▶ In many cases, the dependency of AI-2 on these is partial and “in spirit”.
 - ▶ If you have not taken these (or do not remember), read up on them as needed!
- ▶ **The real Prerequisite:** Motivation, Interest, Curiosity, hard work. (AI-2 is non-trivial)
- ▶ You can do this course if you want! (and I hope you are successful)

- ▶ Academic Assessment: 90 minutes exam directly after courses end (~ Aug. 3 2022)
- ▶ Retake Exam: 90 min exam directly after courses end the following semester (~ Feb. 10. 2023)
- ▶ Mid-semester mini-exam: online, optional, corrected but ungraded, (so you can predict the exam style)
- ▶ Module Grade:
 - ▶ Grade via the exam (Klausur) ~ 100% of the grade
 - ▶ Results from “Übungen zu Künstliche Intelligenz” give up to 10% bonus to a passing exam (not passed ~ no bonus)
- ▶ I do not think that this is the best possible scheme, but I have very little choice.

AI-2 Homework Assignments

- ▶ **Homeworks:** will be small individual problem/programming/proof assignments
(but take time to solve) group submission if and only if explicitly permitted.
- ▶ **⚠ Double Jeopardy ⚠:** Homeworks only give 10% bonus points for the exam, but without trying you are unlikely to pass the exam.
- ▶ **Admin:** To keep things running smoothly
 - ▶ Homeworks will be posted on StudOn.
 - ▶ Sign up for AI-2 under <https://www.studon.fau.de/crs4419186.html>.
 - ▶ Homeworks are handed in electronically there. (plain text, program files, PDF)
 - ▶ **Go to the tutorials, discuss with your TA!** (they are there for you!)
- ▶ **Homework Discipline:**
 - ▶ **Start early!** (many assignments need more than one evening's work)
 - ▶ Don't start by sitting at a blank screen
 - ▶ Humans will be trying to understand the text/code/math when grading it.

Tutorials for Artificial Intelligence 1

- ▶ Weekly tutorials and homework assignments (first one in week two)
- ▶ Instructor/Lead TA: Florian Rabe (florian.rabe@fau.de) Room: 11.137 @ Händler building
- ▶ Tutorials: one each taught by Florian Rabe, ...
- ▶ Goal 1: Reinforce what was taught in class (you need practice)
- ▶ Goal 2: Allow you to ask any question you have in a small and protected environment
- ▶ Life-saving Advice: go to your tutorial, and prepare for it by having looked at the slides and the homework assignments

Textbook, Handouts and Information, Forums, Video

- ▶ **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 **third edition**, which is vastly improved over earlier ones.

Textbook, Handouts and Information, Forums, Video

- ▶ **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 **third edition**, which is vastly improved over earlier ones.
- ▶ **Course notes:** will be posted at
<http://kwarc.info/teaching/AI/notes.pdf>
 - ▶ more detailed than [RN09] in some areas
 - ▶ I mostly prepare them as we go along (**semantically preloaded** \rightsquigarrow **research resource**)
 - ▶ please e-mail me any errors/shortcomings you notice. (improve for the group)

Textbook, Handouts and Information, Forums, Video

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

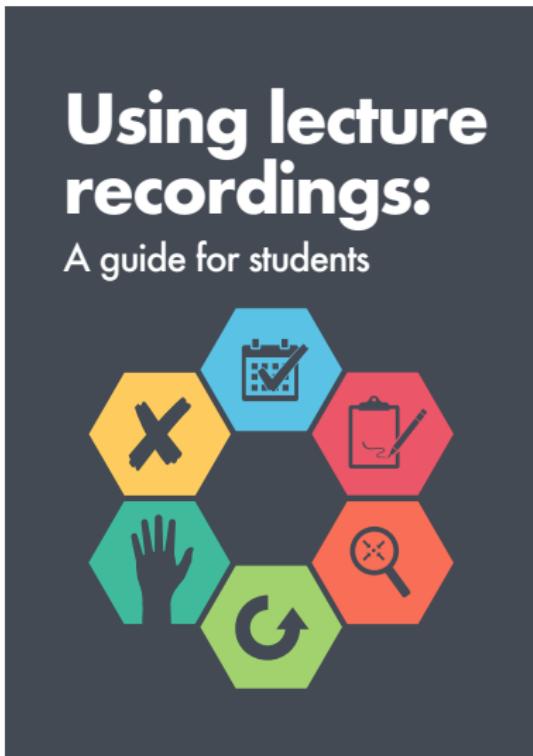
Make sure that you read the **third edition**, which is vastly improved over earlier ones.
- ▶ **Course notes:** will be posted at
<http://kwarc.info/teaching/AI/notes.pdf>
 - ▶ more detailed than [RN09] in some areas
 - ▶ I mostly prepare them as we go along (**semantically preloaded** \rightsquigarrow **research resource**)
 - ▶ please e-mail me any errors/shortcomings you notice. (**improve for the group**)
- ▶ **Course Forum:** on StudOn: <https://www.studon.fau.de/crs4419186.html>
for Announcements, homeworks, discussions

Textbook, Handouts and Information, Forums, Video

- ▶ **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 **third edition**, which is vastly improved over earlier ones.
- ▶ **Course notes:** will be posted at
<http://kwarc.info/teaching/AI/notes.pdf>
 - ▶ more detailed than [RN09] in some areas
 - ▶ I mostly prepare them as we go along (**semantically preloaded** \rightsquigarrow **research resource**)
 - ▶ please e-mail me any errors/shortcomings you notice. (**improve for the group**)
- ▶ **Course Forum:** on StudOn: <https://www.studon.fau.de/crs4419186.html> for Announcements, homeworks, discussions
- ▶ **Course Videos:**
 - ▶ **New and shiny:** Video course nuggets are available at
<https://fau.tv/course/id/2095> (**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>

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



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

- ▶ Some degree programs do not “import” the course Artificial Intelligence, and thus you may not be able to register for the exam via <https://campus.fau.de>.
 - ▶ Just send me an e-mail and come to the exam, we will issue a “Schein”.
 - ▶ Tell your program coordinator about AI-1/2 so that they remedy this situation
- ▶ In “Wirtschafts-Informatik” you can only take AI-1 and AI-2 together in the “Wahlpflichtbereich”.
- ▶ ECTS credits need to be divisible by five $\leftrightarrow 7.5 + 7.5 = 15$.

3 Overview over AI and Topics of AI-II

3.1 What is Artificial Intelligence?

What is Artificial Intelligence? Definition

- ▶ **Definition 3.1 (According to Wikipedia).** **Artificial Intelligence (AI)** is intelligence exhibited by machines
- ▶ **Definition 3.2 (also).** **Artificial Intelligence (AI)** is a sub-field of Computer Science that is concerned with the automation of intelligent behavior.
- ▶ **BUT:** it is already difficult to define "Intelligence" precisely
- ▶ **Definition 3.3 (Elaine Rich).** **Artificial Intelligence (AI)** studies how we can make the computer do things that humans can still do better at the moment.



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



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



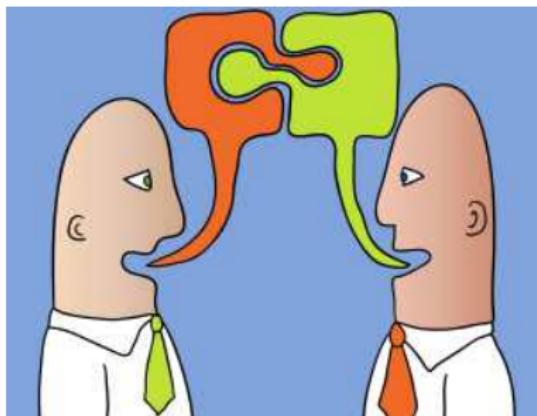
What is Artificial Intelligence? Components

- ▶ 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



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



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

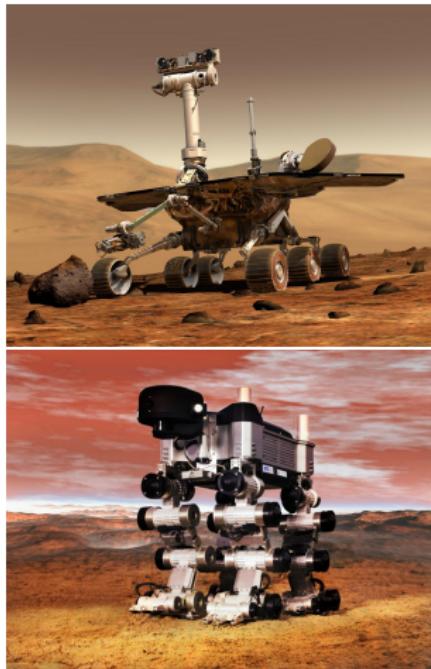


Luisenburg-Festspiele 2004 – "Anatevka" mit Günter Marz und Gisela Ehrenperger

3.2 Artificial Intelligence is here today!

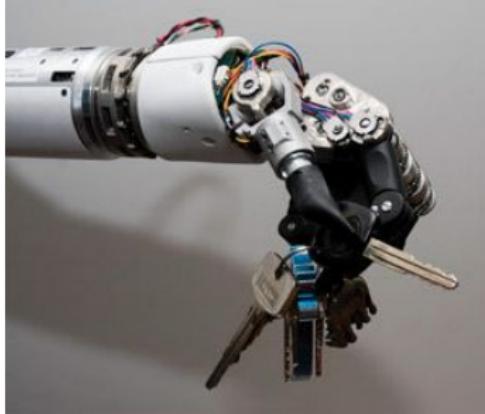
Artificial Intelligence is here today!

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



Artificial Intelligence is here today!

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



Artificial Intelligence is here today!

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



Artificial Intelligence is here today!

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



Artificial Intelligence is here today!

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



© 1999 Randy Glasbergen. www.glasbergen.com



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

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



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

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



- ▶ [AlphaGo](#) is a program by Google DeepMind to play the board game [go](#).
In December 2017 [AlphaZero](#), a successor of [AlphaGo](#) “learned” the games [go](#), [chess](#), and [shogi](#) in 24 hours, achieving a superhuman level of play in these three games by defeating world-champion programs.

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



- ▶ [AlphaGo](#) is a program by Google DeepMind to play the board game [go](#).
By September 2019, [AlphaStar](#), a variant of [AlphaGo](#), attained “grandmaster level” in [Starcraft II](#), a real-time strategy game with partially observable state.
AlphaStar now among the top 0.2% of human players.

The AI Conundrum

- ▶ **Observation:** Reserving the term “Artificial Intelligence” has been quite a land-grab!
- ▶ **But:** researchers at the Dartmouth Conference (1950) really thought they would solve AI in two/three decades.
- ▶ **Consequence:** AI still asks the big questions.
- ▶ **Another Consequence:** AI as a field is an incubator for many innovative technologies.
- ▶ **AI Conundrum:** Once AI solves a subfield it is called “Computer Science”.
(becomes a separate subfield of CS)
- ▶ **Example 3.4.** Functional/Logic Programming, Automated Theorem Proving, Planning, Machine Learning, Knowledge Representation, ...
- ▶ **Still Consequence:** AI research was alternately flooded with money and cut off brutally.

3.3 Ways to Attack the AI Problem

Three Main Approaches to Artificial Intelligence

- ▶ **Definition 3.5.** **Symbolic AI** is based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into structures (expressions) and manipulating them (using processes) to produce new expressions.

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

- ▶ **Definition 3.5.** **Symbolic AI** is based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, combining them into structures (expressions) and manipulating them (using processes) to produce new expressions.
- ▶ **Definition 3.6.** **Statistical AI** remedies the two shortcomings of **symbolic AI** approaches: that all concepts represented by symbols are crisply defined, and that all aspects of the world are knowable/representable in principle. **Statistical AI** adopts sophisticated mathematical models of uncertainty and uses them to create more accurate world models and reason about them.
- ▶ **Definition 3.7.** **Subsymbolic AI** attacks the assumption of **symbolic** and **statistical AI** that intelligence can be achieved by reasoning about the state of the world. Instead it posits that intelligence must be **embodied** – i.e. situated in the world and interact with it via sensors and actuators. The main method for realizing intelligent behavior is by learning from the world, i.e. **machine learning**.

Two ways of reaching Artificial Intelligence?

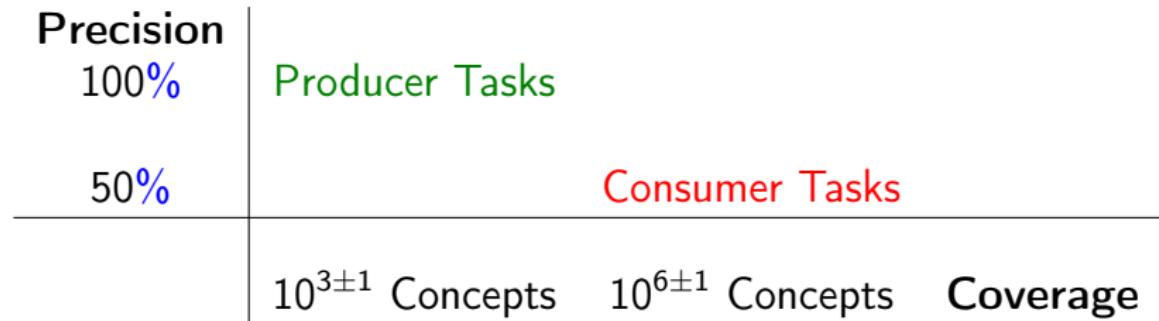
- We can classify the AI approaches by their coverage and the analysis depth (**they are complementary**)

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

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

Environmental Niches for both Approaches to AI

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



- ▶ **General Rule:** **Subsymbolic AI** is well-suited for **consumer tasks**, while **symbolic AI** is better-suited for **producer tasks**.
- ▶ A domain of **producer tasks** I am interested in: Mathematical/Technical Documents.

To get this out of the way . . .



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

3.4 AI in the KWARC Group

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

Overview: KWARC Research and Projects

Applications: eMath 3.0, Active Documents, Active Learning, Semantic Spreadsheets/CAD/CAM, Change Management, Global Digital Math Library, Math Search Systems, [SMGloM](#): Semantic Multilingual Math Glossary, Serious Games, ...

Foundations of Math:

- ▶ [MathML](#), [OpenMath](#)
- ▶ advanced Type Theories
- ▶ [MMT](#): Meta Meta Theory
- ▶ Logic Morphisms/Atlas
- ▶ Theorem Prover/CAS Interoperability
- ▶ Mathematical Models/Simulation

KM & Interaction:

- ▶ Semantic Interpretation (aka. Framing)
- ▶ math-literate interaction
- ▶ MathHub: math archives & active docs
- ▶ Semantic Alliance: embedded semantic services

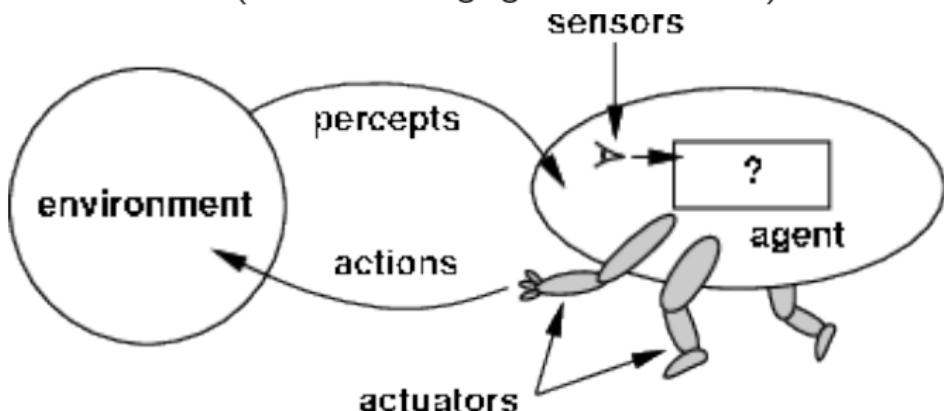
Semantization:

- ▶ [LATEXML](#): [LATEX](#) → XML
- ▶ [STE](#): Semantic [LATEX](#)
- ▶ invasive editors
- ▶ Context-Aware IDEs
- ▶ Mathematical Corpora
- ▶ Linguistics of Math
- ▶ ML for Math Semantics Extraction

Foundations: Computational Logic, Web Technologies, [OMDoc/MMT](#)

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

- ▶ **Definition 3.8.** An **agent** is anything that
 - ▶ perceives its **environment** via **sensors** (means of sensing the **environment**)
 - ▶ acts on it with **actuators** (means of changing the environment).



- ▶ **Example 3.9.** Agents include humans, robots, softbots, thermostats, etc.

- ▶ We construct rational agents.
- ▶ An agent is an entity that perceives its environment through sensors and acts upon that environment through actuators.
- ▶ A rational agent is an agent maximizing its **expected** performance measure.
- ▶ In AI 1 we dealt mainly with a logical approach to agent design (no uncertainty).
- ▶ We ignored
 - ▶ interface to environment (sensors, actuators)
 - ▶ uncertainty
 - ▶ the possibility of self-improvement (learning)

- ▶ 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

Part V Reasoning with Uncertain Knowledge

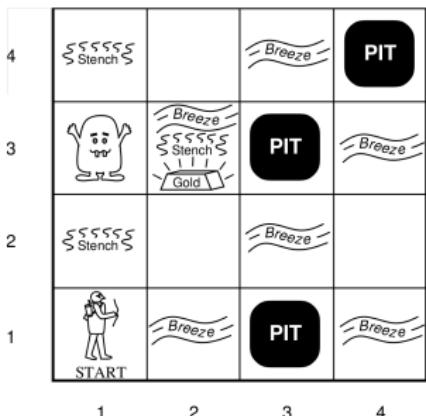
Chapter 20 Quantifying Uncertainty

1 Dealing with Uncertainty: Probabilities

1.1 Sources of Uncertainty

Sources of Uncertainty in Decision-Making

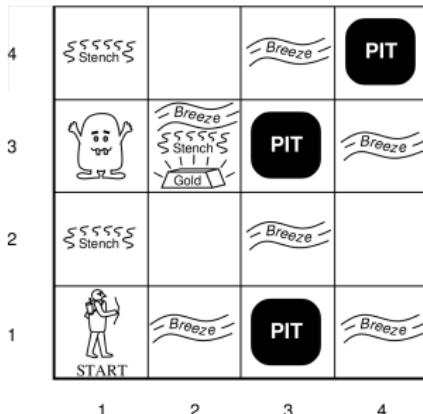
Where's that d... Wumpus?
And where am I, anyway??



- ▶ Non-deterministic actions:
 - ▶ "When I try to go forward in this dark cave, I might actually go forward-left or forward-right."

Sources of Uncertainty in Decision-Making

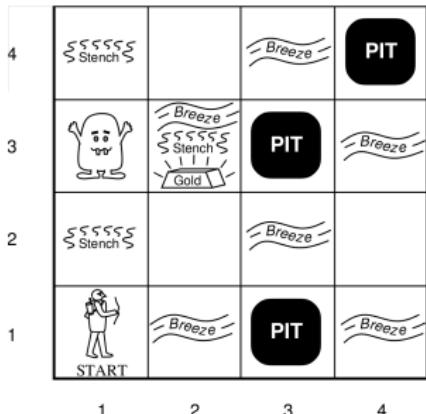
Where's that d... Wumpus?
And where am I, anyway??



- ▶ Non-deterministic actions:
 - ▶ "When I try to go forward in this dark cave, I might actually go forward-left or forward-right."
- ▶ Partial observability with unreliable sensors:
 - ▶ "Did I feel a breeze right now?";
 - ▶ "I think I might smell a Wumpus here, but I got a cold and my nose is blocked."
 - ▶ "According to the heat scanner, the Wumpus is probably in cell [2,3]."

Sources of Uncertainty in Decision-Making

Where's that d... Wumpus?
And where am I, anyway??



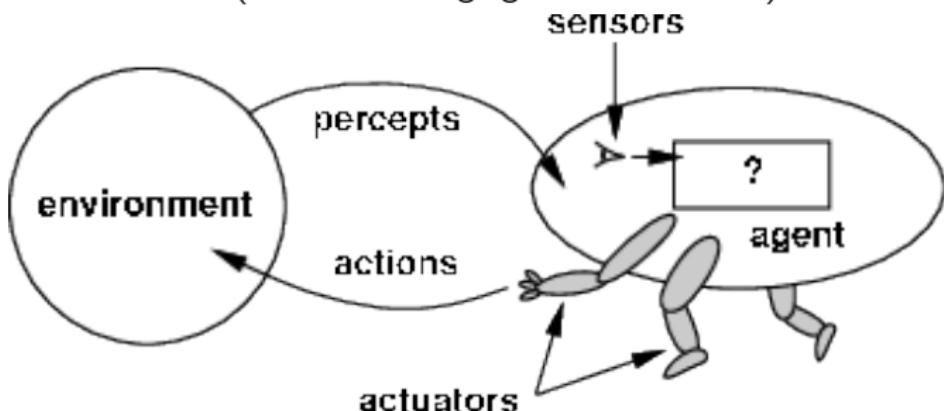
- ▶ Non-deterministic actions:
 - ▶ "When I try to go forward in this dark cave, I might actually go forward-left or forward-right."
- ▶ Partial observability with unreliable sensors:
 - ▶ "Did I feel a breeze right now?";
 - ▶ "I think I might smell a Wumpus here, but I got a cold and my nose is blocked."
 - ▶ "According to the heat scanner, the Wumpus is probably in cell [2,3]."
- ▶ Uncertainty about the domain behavior:
 - ▶ "Are you *sure* the Wumpus never moves?"

- ▶ **Robot Localization:** Suppose we want to support localization using landmarks to narrow down the area.
- ▶ **Example 1.1.** *If you see the Eiffel tower, then you're in Paris.*

- ▶ **Robot Localization:** Suppose we want to support localization using landmarks to narrow down the area.
- ▶ **Example 1.1.** *If you see the Eiffel tower, then you're in Paris.*
- ▶ **Difficulty:** Sensors can be imprecise.
 - ▶ Even if a landmark is perceived, we cannot conclude with certainty that the robot is at that location.
 - ▶ *This is the half-scale Las Vegas copy, you dummy.*
 - ▶ Even if a landmark is *not* perceived, we cannot conclude with certainty that the robot is *not* at that location.
 - ▶ *Top of Eiffel tower hidden in the clouds.*
- ▶ Only the **probability** of being at a location increases or decreases.

1.2 Recap: Rational Agents as a Conceptual Framework

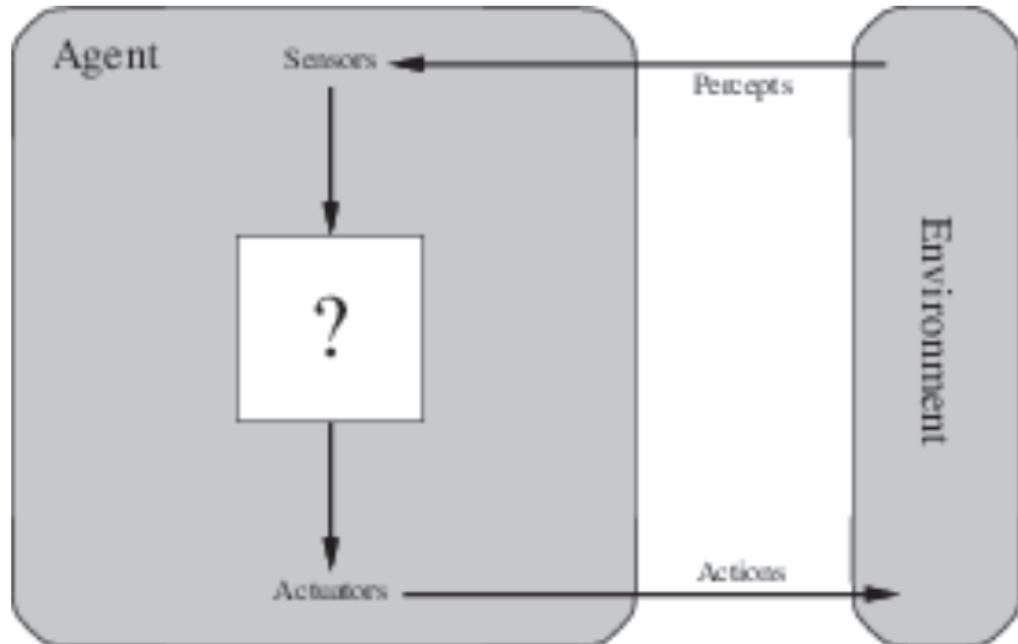
- ▶ **Definition 1.2.** An **agent** is anything that
 - ▶ perceives its **environment** via **sensors** (means of sensing the **environment**)
 - ▶ acts on it with **actuators** (means of changing the environment).



- ▶ **Example 1.3.** Agents include humans, robots, softbots, thermostats, etc.

Agent Schema: Visualizing the Internal Agent Structure

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



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

- ▶ Idea: Try to design agents that are successful (aka. “do the right thing”)
- ▶ **Definition 1.4.** A **performance measure** is a function that evaluates a sequence of environments.
- ▶ **Example 1.5.** A **performance measure** for the vacuum cleaner world could
 - ▶ award one point per square cleaned up in time T ?
 - ▶ award one point per clean square per time step, minus one per move?
 - ▶ penalize for $> k$ dirty squares?
- ▶ **Definition 1.6.** An **agent** is called **rational**, if it chooses whichever **action** maximizes the expected value of the performance measure given the **percept** sequence to date.
- ▶ **Question:** Why is **rationality** a good quality to aim for?

Consequences of Rationality: Exploration, Learning, Autonomy

- ▶ Note: a rational need not be perfect
 - ▶ only needs to maximize **expected value** (Rational \neq omniscient)
 - ▶ need not predict e.g. very unlikely but catastrophic events in the future
 - ▶ **percepts** may not supply all relevant information (Rational \neq clairvoyant)
 - ▶ if we cannot perceive things we do not need to react to them.
 - ▶ but we may need to try to find out about hidden dangers (exploration)
 - ▶ **action** outcomes may not be as expected (rational \neq successful)
 - ▶ but we may need to take **action** to ensure that they do (more often) (learning)
- ▶ Rational \leadsto exploration, learning, autonomy
- ▶ **Definition 1.7.** An **agent** is called **autonomous**, if it does not rely on the prior knowledge of the designer.
- ▶ Autonomy avoids fixed behaviors that can become unsuccessful in a changing environment. (anything else would be irrational)
- ▶ The **agent** has to learn all relevant traits, invariants, properties of the environment and **actions**.

PEAS: Describing the Task Environment

- ▶ **Observation:** To design a rational agent, we must specify the **task environment** in terms of **performance measure**, **environment**, **actuators**, and **sensors**, together called the **PEAS** components.
- ▶ **Example 1.8.** 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 1.9 (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)

Environment types

- ▶ **Observation 1.10.** 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 1.11.** For an agent a we classify the environment e of a by its type, which is one of the following. We call e
 1. **fully observable**, iff the a 's sensors give it access to the complete state of the environment at any point in time, else **partially observable**.
 2. **deterministic**, iff the next state of the environment is completely determined by the current state and a 's **action**, else **stochastic**.
 3. **episodic**, iff a 's experience is divided into atomic **episodes**, where it perceives and then performs a single **action**. Crucially the next episode does not depend on previous ones. Non-**episodic environments** are called **sequential**.
 4. **dynamic**, iff the environment can change without an **action** performed by a , else **static**. If the environment does not change but a 's performance measure does, we call e **semidynamic**.
 5. **discrete**, iff the sets of e 's state and a 's **actions** are countable, else **continuous**.
 6. **single agent**, iff only a acts on e ; else **multi agent**(when must we count parts of e as agents?)

Environment Types (Examples)

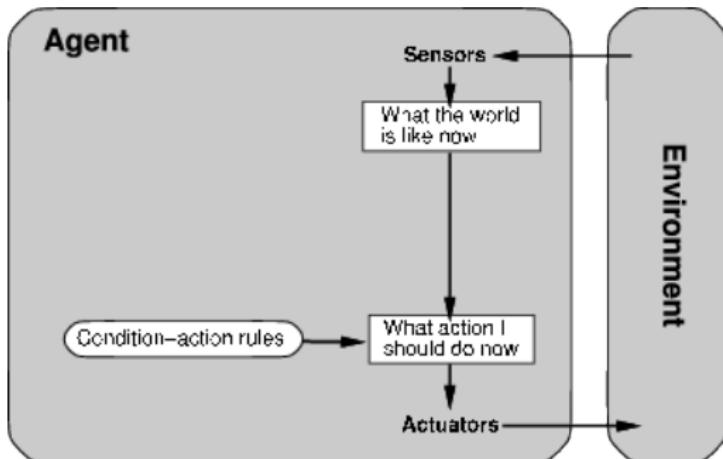
- **Example 1.12.** Some environments classified:

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

- **Observation 1.13.** The real world is (of course) **partially observable**, **stochastic**, **sequential**, **dynamic**, **continuous**, and **multi agent** (**worst case for AI**)

Simple reflex agents

- ▶ **Definition 1.14.** A **simple reflex agent** is an **agent** a that only bases its actions on the last percept: $f_a: \mathcal{P} \rightarrow \mathcal{A}$.
- ▶ **Agent Schema:**

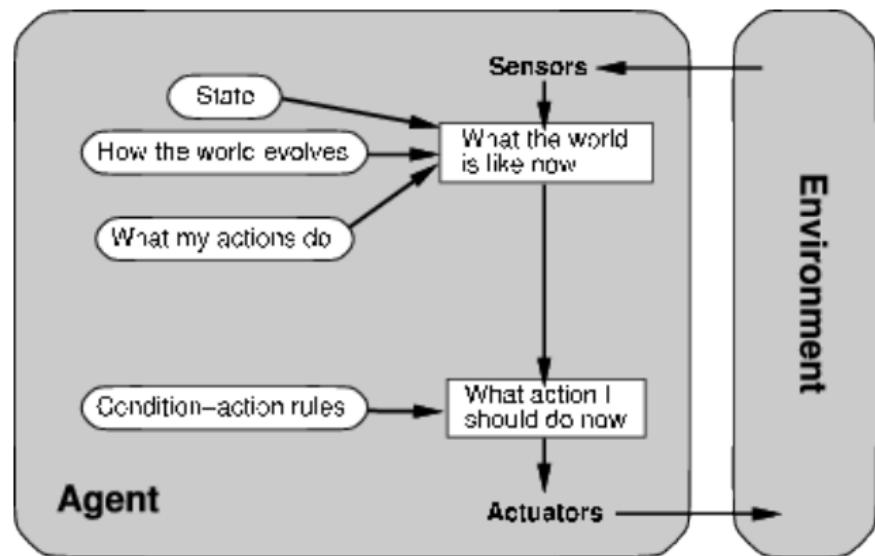


- ▶ **Example 1.15 (Agent Program).**

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

Model-based Reflex Agents

- ▶ Idea: Keep track of the **state** of the world we cannot see now in an internal model.
- ▶ **Definition 1.16.** A **model based agent** (also called **reflex agent with state**) whose **agent function** depends on a model of the world (called the **state** or **world model**).
- ▶ Agent Schema:



1.3 Agent Architectures based on Belief States

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!

- ▶ **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 1.17.** 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 **transition model** that updates the **belief state** based on **sensor** information and **actions**.

- ▶ **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 1.17.** 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 **transition model** that updates the **belief state** based on **sensor** information and **actions**.
- ▶ **Idea:** The agent **environment** determines what the world model can be.

- ▶ **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 1.17.** 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 **transition model** that updates the **belief state** based on **sensor** information and **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 set (we call its member the **world state**) and the **transition model** is a function from **states** and **actions** to **states**: a **transition function**.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model. What we can do with it depends on representation and inference.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model.
What we can do with it depends on representation and inference.
- ▶ **Search-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ "current state"
 - ▶ no inference.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model.
What we can do with it depends on representation and inference.
- ▶ **Search-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ "current state"
 - ▶ no inference.
- ▶ **CSP-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ constraint network
 - ▶ inference $\hat{=}$ constraint propagation.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model.
What we can do with it depends on representation and inference.
- ▶ **Search-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ "current state"
 - ▶ no inference.
- ▶ **CSP-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ constraint network
 - ▶ inference $\hat{=}$ constraint propagation.
- ▶ **Logic-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ logical formula
 - ▶ inference $\hat{=}$ e.g. DPLL or resolution.

World Models by Agent Type

- ▶ **Note:** All of these considerations only give requirements to the world model.
What we can do with it depends on representation and inference.
- ▶ **Search-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ "current state"
 - ▶ no inference.
- ▶ **CSP-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ constraint network
 - ▶ inference $\hat{=}$ constraint propagation.
- ▶ **Logic-based Agents:** In a **fully observable, deterministic environment**
 - ▶ **world state** $\hat{=}$ logical formula
 - ▶ inference $\hat{=}$ e.g. DPLL or resolution.
- ▶ **Planning Agents:** In a **fully observable, deterministic, environment**
 - ▶ **world state** $\hat{=}$ PL0, transition model $\hat{=}$ STRIPS,
 - ▶ inference $\hat{=}$ state/plan space search.

- ▶ In a **fully observable**, but **stochastic** environment,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ \leadsto generalize the **transition function** to a **transition relation**.

- ▶ In a **fully observable**, but **stochastic** environment,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ \leadsto generalize the **transition function** to a **transition relation**.
- ▶ Note: This even applies to **online problem solving**, where we can just perceive the **state**.
(e.g. when we want to optimize utility)

World Models for Complex Environments

- ▶ In a **fully observable**, but **stochastic** environment,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ \leadsto generalize the **transition function** to a **transition relation**.
- ▶ Note: This even applies to **online problem solving**, where we can just perceive the **state**.
(e.g. when we want to optimize utility)
- ▶ In a **deterministic**, but **partially observable environment**,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ we can use **transition functions**.
 - ▶ We need a **sensor model**, which predicts the influence of **percepts** on the **belief state**
 - during update.

- ▶ In a **fully observable**, but **stochastic** environment,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ \leadsto generalize the **transition function** to a **transition relation**.
- ▶ Note: This even applies to **online problem solving**, where we can just perceive the **state**.
(e.g. when we want to optimize utility)
- ▶ In a **deterministic**, but **partially observable environment**,
 - ▶ the **belief state** must deal with a set of possible **states**.
 - ▶ we can use **transition functions**.
 - ▶ We need a **sensor model**, which predicts the influence of **percepts** on the **belief state**
 - during update.
- ▶ In a **stochastic**, **partially observable environment**,
 - ▶ mix the ideas from the last two.
(sensor model + transition relation)

Preview: New World Models (Belief) \leadsto new Agent Types

- ▶ Probabilistic Agents: In a partially observable environment
 - ▶ belief state $\hat{=}$ Bayesian networks,
 - ▶ inference $\hat{=}$ probabilistic inference.

- ▶ Probabilistic Agents: In a partially observable environment
 - ▶ belief state $\hat{=}$ Bayesian networks,
 - ▶ inference $\hat{=}$ probabilistic inference.
- ▶ Decision-Theoretic Agents: In a partially observable, stochastic environment
 - ▶ belief state + transition model $\hat{=}$ decision networks,
 - ▶ inference $\hat{=}$ maximizing expected utility.
- ▶ We will study them in detail in the coming semester.

1.4 Modeling Uncertainty

Wumpus World Revisited

- ▶ **Recall:** We have updated agents with world/transition models with possible worlds.
- ▶ **Problem:** But pure sets of possible worlds are not enough
- ▶ **Example 1.18 (Beware of the Pit).**
 - ▶ We have a maze with pits that are detected in neighbouring squares via breeze (Wumpus and gold will not be assumed now).
 - ▶ Where does the agent should go, if there is breeze at (1,2) and (2,1)?
 - ▶ **Problem:** (1,3), (2,2), and (3,1) are all unsafe! (there are possible worlds with pits in any of them)
 - ▶ **Idea:** We need world models that estimate the pit-likelihood in cells!

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 B OK	2,2	3,2	4,2
1,1	2,1 B OK	3,1	4,1

- ▶ **Example 1.19 (title=Diagnosis).** We want to build an expert dental diagnosis system, that deduces the cause (the disease) from the symptoms.
- ▶ Can we base this on logic?

- ▶ **Example 1.19 (title=Diagnosis).** We want to build an expert dental diagnosis system, that deduces the cause (the disease) from the symptoms.
- ▶ Can we base this on logic?
- ▶ **Attempt 1:** Say we have a toothache. How's about:

$$\forall p. \text{Symptom}(p, \text{toothache}) \Rightarrow \text{Disease}(p, \text{cavity})$$

- ▶ Is this rule correct?

- ▶ **Example 1.19 (title=Diagnosis).** We want to build an expert dental diagnosis system, that deduces the cause (the disease) from the symptoms.
- ▶ Can we base this on logic?
- ▶ **Attempt 1:** Say we have a toothache. How's about:

$$\forall p. \text{Symptom}(p, \text{toothache}) \Rightarrow \text{Disease}(p, \text{cavity})$$

- ▶ Is this rule correct?
- ▶ No, toothaches may have different causes ("cavity" $\hat{=}$ "Loch im Zahn").
- ▶ **Attempt 2:** So what about this:

$$\forall p. \text{Symptom}(p, \text{toothache}) \Rightarrow \text{Disease}(p, \text{cavity}) \vee \text{Disease}(p, \text{gingivitis}) \vee \dots$$

- ▶ We don't know all possible causes.
- ▶ And we'd like to be able to deduce which causes are more plausible!

Uncertainty and Logic, ctd.

- ▶ Attempt 3: Perhaps a causal rule is better?

$$\forall p. \text{Disease}(p, \text{cavity}) \Rightarrow \text{Symptom}(p, \text{toothache})$$

- ▶ Question: Is this rule correct?

Uncertainty and Logic, ctd.

- ▶ Attempt 3: Perhaps a causal rule is better?

$$\forall p. \text{Disease}(p, \text{cavity}) \Rightarrow \text{Symptom}(p, \text{toothache})$$

- ▶ Question: Is this rule correct?
- ▶ Answer: No, not all cavities cause toothaches.
- ▶ Question: Does this rule allow to deduce a cause from a symptom?

Uncertainty and Logic, ctd.

- ▶ Attempt 3: Perhaps a causal rule is better?

$$\forall p. \text{Disease}(p, \text{cavity}) \Rightarrow \text{Symptom}(p, \text{toothache})$$

- ▶ Question: Is this rule correct?
- ▶ Answer: No, not all cavities cause toothaches.
- ▶ Question: Does this rule allow to deduce a cause from a symptom?
- ▶ Answer: No, setting $\text{Symptom}(p, \text{toothache})$ to true here has no consequence on the truth of $\text{Disease}(p, \text{cavity})$.

- ▶ Attempt 3: Perhaps a causal rule is better?

$$\forall p. \text{Disease}(p, \text{cavity}) \Rightarrow \text{Symptom}(p, \text{toothache})$$

- ▶ Question: Is this rule correct?
- ▶ Answer: No, not all cavities cause toothaches.
- ▶ Question: Does this rule allow to deduce a cause from a symptom?
- ▶ Answer: No, setting $\text{Symptom}(p, \text{toothache})$ to true here has no consequence on the truth of $\text{Disease}(p, \text{cavity})$.
- ▶ Note: If $\text{Symptom}(p, \text{toothache})$ is false, we would conclude $\neg \text{Disease}(p, \text{cavity})$... which would be incorrect, cf. previous question.

- ▶ Attempt 3: Perhaps a causal rule is better?

$$\forall p. \text{Disease}(p, \text{cavity}) \Rightarrow \text{Symptom}(p, \text{toothache})$$

- ▶ Question: Is this rule correct?
- ▶ Answer: No, not all cavities cause toothaches.
- ▶ Question: Does this rule allow to deduce a cause from a symptom?
- ▶ Answer: No, setting $\text{Symptom}(p, \text{toothache})$ to true here has no consequence on the truth of $\text{Disease}(p, \text{cavity})$.
- ▶ Note: If $\text{Symptom}(p, \text{toothache})$ is false, we would conclude $\neg \text{Disease}(p, \text{cavity})$... which would be incorrect, cf. previous question.
- ▶ Anyway, this still doesn't allow to compare the plausibility of different causes.
- ▶ Summary: Logic does not allow to weigh different alternatives, and it does not allow to express incomplete knowledge ("cavity does not always come with a toothache, nor vice versa").

- ▶ Question: What do we model with probabilities?

- ▶ **Question:** What do we model with probabilities?
- ▶ **Answer:** Incomplete knowledge!
 - ▶ We are certain, but we *believe to a certain degree* that something is true.
 - ▶ Probability $\hat{=}$ Our degree of belief, given our current knowledge.

- ▶ **Question:** What do we model with probabilities?
- ▶ **Answer:** Incomplete knowledge!
 - ▶ We are certain, but we *believe to a certain degree* that something is true.
 - ▶ Probability $\hat{=}$ Our degree of belief, given our current knowledge.
- ▶ **Example 1.20 (Diagnosis).**
 - ▶ Symptom(p , toothache) \Rightarrow Disease(p , cavity) with 80% probability.
 - ▶ But, for any given p , in reality we do, or do not, have cavity: 1 or 0!
 - ▶ The “probability” depends on our knowledge!
 - ▶ The “80%” refers to the fraction of cavities within the set of all p' that are indistinguishable from p based on our knowledge.
 - ▶ If we receive new knowledge (e.g., Disease(p , gingivitis)), the probability changes!
- ▶ Probabilities represent and measure the uncertainty that stems from lack of knowledge.

How to Obtain Probabilities?

- ▶ Assessing probabilities through statistics:
 - ▶ The agent is 90% convinced by its sensor information. (in 9 out of 10 cases, the information is correct)
 - ▶ $\text{Disease}(p, \text{cavity}) \Rightarrow \text{Symptom}(p, \text{toothache})$ with 80% probability
:= 8 out of 10 persons with a cavity have toothache.
- ▶ **Definition 1.21.** The process of estimating a probability P using statistics is called **assessing P** .
- ▶ **Observation:** Assessing even a single P can require huge effort!
- ▶ **Example 1.22.** The likelihood of making it to the university within 10 minutes.
- ▶ **What is probabilistic reasoning?** Deducing probabilities from knowledge about other probabilities.
- ▶ **Idea:** Probabilistic reasoning determines, based on probabilities that are (relatively) easy to **assess**, probabilities that are difficult to **assess**.

1.5 Acting under Uncertainty

- ▶ **Example 1.23 (Giving a lecture).**
 - ▶ **Goal:** Be in HS002 at 10:15 to give a lecture.

- ▶ **Example 1.23 (Giving a lecture).**
 - ▶ **Goal:** Be in HS002 at 10:15 to give a lecture.
 - ▶ **Possible plans:**
 - ▶ P_1 : Get up at 8:00, leave at 8:40, arrive at 9:00.
 - ▶ P_2 : Get up at 9:50, leave at 10:05, arrive at 10:15.

► Example 1.23 (Giving a lecture).

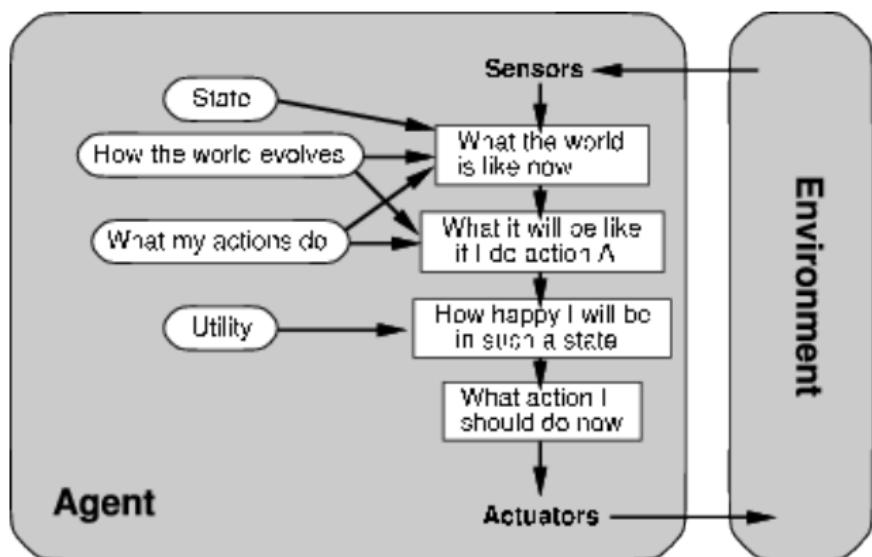
- **Goal:** Be in HS002 at 10:15 to give a lecture.
- **Possible plans:**
 - P_1 : Get up at 8:00, leave at 8:40, arrive at 9:00.
 - P_2 : Get up at 9:50, leave at 10:05, arrive at 10:15.
- **Decision:** Both plans are correct, but P_2 succeeds only with probability 50%, and giving a lecture is important, so P_1 is the plan of choice.

- ▶ **Example 1.23 (Giving a lecture).**
 - ▶ **Goal:** Be in HS002 at 10:15 to give a lecture.
 - ▶ **Possible plans:**
 - ▶ P_1 : Get up at 8:00, leave at 8:40, arrive at 9:00.
 - ▶ P_2 : Get up at 9:50, leave at 10:05, arrive at 10:15.
 - ▶ **Decision:** Both plans are correct, but P_2 succeeds only with probability 50%, and giving a lecture is important, so P_1 is the plan of choice.
- ▶ **Example 1.24 (Better Example).** Which train to take to Frankfurt airport?

- ▶ **Here:** We're only concerned with deducing the likelihood of facts, not with action choice. In general, selecting actions is of course important.
- ▶ **Rational Agents:**
 - ▶ We have a choice of **actions** (go to FRA early, go to FRA just in time).
 - ▶ These can lead to different solutions with different probabilities.
 - ▶ The actions have different **costs**.
 - ▶ The results have different **utilities** (safe timing/dislike airport food).
- ▶ A rational agent chooses the action with the **maximum expected utility**.
- ▶ Decision Theory = Utility Theory + Probability Theory.

Utility-based agents

- ▶ **Definition 1.25.** A **utility based agent** uses a **world model** along with a **utility function** that influences its **preferences** among the **states** of that world. It chooses the **action** that leads to the best **expected utility**, which is computed by averaging over all possible outcome **states**, weighted by the probability of the outcome.
- ▶ **Agent Schema:**



► Example 1.26 (A particular kind of utility-based agent).

function DT-AGENT(*percept*) **returns** an *action*

persistent: *belief_state*, probabilistic beliefs about the current state of the world
 action, the agent's action

 update *belief_state* based on *action* and *percept*

 calculate outcome probabilities for actions,

 given action descriptions and current *belief_state*

 select *action* with highest expected utility

 given probabilities of outcomes and utility information

return *action*

1.6 Agenda for this Chapter: Basics of Probability Theory

Our Agenda for This Topic

- ▶ Our treatment of the topic “Probabilistic Reasoning” consists of this Chapter and the next.
 - ▶ This Chapter: All the basic machinery at use in [Bayesian networks](#).
 - ▶ Next Chapter: [Bayesian networks](#): What they are, how to build them, how to use them.
- ▶ [Bayesian networks](#) are the most wide-spread and successful practical framework for probabilistic reasoning.

Our Agenda for This Chapter

- ▶ **Unconditional Probabilities and Conditional Probabilities:** Which concepts and properties of probabilities will be used?
 - ▶ Mostly a recap of things you're familiar with from school.
- ▶ **Independence and Basic Probabilistic Reasoning Methods:** What simple methods are there to avoid enumeration and to deduce probabilities from other probabilities?
 - ▶ A basic tool set we'll need. (Still familiar from school?)
- ▶ **Bayes' Rule:** What's that "Bayes"? How is it used and why is it important?
 - ▶ The basic insight about how to invert the "direction" of conditional probabilities.
- ▶ **Conditional Independence:** How to capture and exploit complex relations between **random variables**?
 - ▶ Explains the difficulties arising when using Bayes' rule on multiple evidences. **conditional independence** is used to ameliorate these difficulties.

2 Unconditional Probabilities

- ▶ **Definition 2.1.** A **probability theory** is an assertion language for talking about **possible worlds** and an inference method for quantifying the **degree of belief** in such assertions.
- ▶ **Remark:** Like **logic**, but for non-binary **belief degree**.
- ▶ The possible worlds are **mutually exclusive**: different possible worlds cannot both be the case and **exhaustive**: one possible world must be the case.
- ▶ This determines the set of **possible worlds**.
- ▶ **Example 2.2.** If we roll two (distinguishable) dice with six sides, then we have 36 **possible worlds**: $(1,1)$, $(2,1)$, ..., $(6,6)$.
- ▶ We will restrict ourselves to a **discrete, countable sample space**. (others more complicated, less useful in AI)
- ▶ **Definition 2.3.** A **probability model** $\langle \Omega, P \rangle$ consists of a **countable** set Ω of **possible worlds** called the **sample space** and a **probability function** $P: \Omega \rightarrow \mathbb{R}$, such that $0 \leq P(\omega) \leq 1$ for all $\omega \in \Omega$ and $\sum_{\omega \in \Omega} P(\omega) = 1$.

- ▶ **Definition 2.4.** A **random variable** (also called **random quantity**, **aleatory variable**, or **stochastic variable**) is a variable quantity whose **value** depends on possible outcomes of unknown **variables** and processes we do not understand.
- ▶ **Definition 2.5.** If X is a **random variable** and x a possible **value**, we will refer to the fact $X = x$ as an **outcome** and a set of **outcomes** as an **event**. The set of possible **outcomes** of X is called the **domain** of X .
- ▶ The notation **uppercase “ X ”** for a **random variable**, and **lowercase “ x ”** for one of its values will be used frequently. (following Russel/Norvig)
- ▶ **Definition 2.6.** Given a **random variable** X , $P(X = x)$ denotes the **prior probability**, or **unconditional probability**, that X has value x in the absence of any other information.
- ▶ **Example 2.7.** $P(\text{Cavity} = \text{T}) = 0.2$, where Cavity is a random variable whose value is true iff some given person has a cavity.

Types of Random Variables

- ▶ **Definition 2.8.** We say that a random variable X is **finite domain**, iff the domain D of X is finite and **Boolean**, iff $D = \{T,F\}$.
- ▶ **Note:** In general, random variables can have arbitrary domains. In AI-1, we restrict ourselves to finite domain and Boolean random variables.
- ▶ **Example 2.9.** Some prior probabilities

$$P(\text{Weather} = \text{sunny}) = 0.7$$

$$P(\text{Weather} = \text{rain}) = 0.2$$

$$P(\text{Weather} = \text{cloudy}) = 0.08$$

$$P(\text{Weather} = \text{snow}) = 0.02$$

$$P(\text{Headache} = T) = 0.1$$

Unlike us, Russel and Norvig live in California ... :-(:-(

- ▶ Convenience Notations:
 - ▶ By convention, we denote Boolean random variables with A, B , and more general finite domain random variables with X, Y .
 - ▶ For a Boolean random variable Name, we write name for the outcome $\text{Name} = T$ and $\neg\text{name}$ for $\text{Name} = F$.
(Follows Russel/Norvig as well)

Probability Distributions

- ▶ **Definition 2.10.** The **probability distribution** for a **random variable** X , written $P(X)$, is the vector of probabilities for the (ordered) domain of X .
- ▶ **Example 2.11.** Probability distributions for finite domain and Boolean random variables

$$P(\text{Headache}) = \langle 0.1, 0.9 \rangle$$

$$P(\text{Weather}) = \langle 0.7, 0.2, 0.08, 0.02 \rangle$$

define the probability distribution for the random variables Headache and Weather.

- ▶ **Definition 2.12.** Given a subset $Z \subseteq \{X_1, \dots, X_n\}$ of random variables, an **event** is an assignment of values to the variables in Z . The **joint probability distribution**, written $P(Z)$, lists the probabilities of all events.
- ▶ **Example 2.13.** $P(\text{Headache}, \text{Weather})$ is

	Headache = T	Headache = F
Weather = sunny	$P(W = \text{sunny} \wedge \text{headache})$	$P(W = \text{sunny} \wedge \neg \text{headache})$
Weather = rain		
Weather = cloudy		
Weather = snow		

The Full Joint Probability Distribution

- ▶ **Definition 2.14.** Given random variables $\{X_1, \dots, X_n\}$, an **atomic event** is an assignment of values to all variables.
- ▶ **Example 2.15.** If A and B are Boolean random variables, then we have four atomic events: $a \wedge b$, $a \wedge \neg b$, $\neg a \wedge b$, $\neg a \wedge \neg b$.
- ▶ **Definition 2.16.** Given random variables $\{X_1, \dots, X_n\}$, the **full joint probability distribution**, denoted $P(X_1, \dots, X_n)$, lists the probabilities of all atomic events.
- ▶ **Observation:** Given random variables $\{X_1, \dots, X_n\}$ with domains D_1, \dots, D_n , the full joint probability distribution is an n -dimensional array of size $\langle D_1, \dots, D_n \rangle$.
- ▶ **Example 2.17.** $P(\text{Cavity}, \text{Toothache})$

	toothache	\neg toothache
cavity	0.12	0.08
\neg cavity	0.08	0.72

- ▶ **Note:** All atomic events are disjoint (their pairwise conjunctions all are equivalent to F); the sum of all fields is 1 (the disjunction over all atomic events is T).

- ▶ **Definition 2.18.** Given random variables $\{X_1, \dots, X_n\}$, a **proposition** is a PL^0 wff over the atoms $X_i = x_i$ where x_i is a value in the domain of X_i . A function P that maps propositions into $[0,1]$ is a **probability measure** if
 1. $P(\top) = 1$ and
 2. for all propositions A , $P(A) = \sum_{e \models A} P(e)$ where e is an **atomic event**.
- ▶ **Propositions** represent sets of **atomic events**: the interpretations satisfying the formula.
- ▶ **Example 2.19.** $P(\text{cavity} \wedge \text{toothache}) = 0.12$ is the probability that some given person has both a cavity and a toothache. (Note the use of cavity for Cavity = \top and toothache for Toothache = \top .)
- ▶ **Notes:**
 - ▶ Instead of $P(a \wedge b)$, we often write $P(a, b)$.
 - ▶ Propositions can be viewed as **Boolean random variables**; we will denote them with A, B as well.

3 Conditional Probabilities

Conditional Probabilities: Intuition

- ▶ Do probabilities change as we gather new knowledge?

- ▶ Do probabilities change as we gather new knowledge?
- ▶ Yes! Probabilities model our *belief*, thus they depend on our knowledge.
- ▶ **Example 3.1.** Your “probability of missing the connection train” increases when you are informed that your current train has 30 minutes delay.
- ▶ **Example 3.2.** The “probability of cavity” increases when the doctor is informed that the patient has a toothache.

- ▶ Do probabilities change as we gather new knowledge?
- ▶ Yes! Probabilities model our *belief*, thus they depend on our knowledge.
- ▶ **Example 3.1.** Your “probability of missing the connection train” increases when you are informed that your current train has 30 minutes delay.
- ▶ **Example 3.2.** The “probability of cavity” increases when the doctor is informed that the patient has a toothache.
- ▶ In the presence of additional information, we can no longer use the unconditional (*prior!*) probabilities.
- ▶ Given propositions A and B , $P(a|b)$ denotes the **conditional probability** of a (i.e., $A = \text{True}$) given that all we know is b (i.e., $B = \text{True}$).

- ▶ Do probabilities change as we gather new knowledge?
- ▶ Yes! Probabilities model our *belief*, thus they depend on our knowledge.
- ▶ **Example 3.1.** Your “probability of missing the connection train” increases when you are informed that your current train has 30 minutes delay.
- ▶ **Example 3.2.** The “probability of cavity” increases when the doctor is informed that the patient has a toothache.
- ▶ In the presence of additional information, we can no longer use the unconditional (*prior!*) probabilities.
- ▶ Given propositions A and B , $P(a|b)$ denotes the **conditional probability** of a (i.e., $A = \text{T}$) given that all we know is b (i.e., $B = \text{T}$).
- ▶ **Example 3.3.** $P(\text{cavity}) = 0.2$ vs. $P(\text{cavity}|\text{toothache}) = 0.6$.
- ▶ **Example 3.4.** $P(\text{cavity}|(\text{toothache} \wedge \neg \text{cavity})) = 0$

- ▶ **Definition 3.5.** Given propositions A and B where $P(b) \neq 0$, the **conditional probability**, or **posterior probability**, of a given b , written $P(a|b)$, is defined as:

$$P(a|b) := \frac{P(a \wedge b)}{P(b)}$$

- ▶ **Intuition:** The likelihood of having a and b , within the set of outcomes where we have b .
- ▶ **Example 3.6.** $P(\text{cavity} \wedge \text{toothache}) = 0.12$ and $P(\text{toothache}) = 0.2$ yield $P(\text{cavity}|\text{toothache}) = 0.6$.

Conditional Probability Distributions

- ▶ **Definition 3.7.** Given random variables X and Y , the **conditional probability distribution** of X given Y , written $P(X|Y)$, i.e. with a boldface P , is the table of all conditional probabilities of values of X given values of Y .
- ▶ For sets of variables: $P(X_1, \dots, X_n | Y_1, \dots, Y_m)$.
- ▶ **Example 3.8.** $P(\text{Weather}|\text{Headache}) =$

	Headache = T	Headache = F
Weather = sunny	$P(W = \text{sunny} \text{headache})$	$P(W = \text{sunny} \neg \text{headache})$
Weather = rain		
Weather = cloudy		
Weather = snow		

What is *The probability of sunshine given that I have a headache?*

- ▶ If you're susceptible to headaches depending on weather conditions, this makes sense. Otherwise, the two variables are **independent**. (see next section)

4 Independence

Working with the Full Joint Probability Distribution

- ▶ **Example 4.1.** Consider the following full joint probability distribution:

	toothache	\neg toothache
cavity	0.12	0.08
\neg cavity	0.08	0.72

- ▶ How to compute $P(\text{cavity})$?

Working with the Full Joint Probability Distribution

- **Example 4.1.** Consider the following full joint probability distribution:

	toothache	\neg toothache
cavity	0.12	0.08
\neg cavity	0.08	0.72

- How to compute $P(\text{cavity})$?
- Sum across the row:

$$P(\text{cavity} \wedge \text{toothache}) + P(\text{cavity} \wedge \neg \text{toothache}) = 0.2$$

- How to compute $P(\text{cavity} \vee \text{toothache})$?

Working with the Full Joint Probability Distribution

- **Example 4.1.** Consider the following full joint probability distribution:

	toothache	\neg toothache
cavity	0.12	0.08
\neg cavity	0.08	0.72

- How to compute $P(\text{cavity})$?
- Sum across the row:

$$P(\text{cavity} \wedge \text{toothache}) + P(\text{cavity} \wedge \neg \text{toothache}) = 0.2$$

- How to compute $P(\text{cavity} \vee \text{toothache})$?
- Sum across atomic events:

$$P(\text{cavity} \wedge \text{toothache}) + P(\neg \text{cavity} \wedge \text{toothache}) + P(\text{cavity} \wedge \neg \text{toothache}) = 0.28$$

- How to compute $P(\text{cavity} | \text{toothache})$?

Working with the Full Joint Probability Distribution

- ▶ **Example 4.1.** Consider the following full joint probability distribution:

	toothache	\neg toothache
cavity	0.12	0.08
\neg cavity	0.08	0.72

- ▶ How to compute $P(\text{cavity})$?
- ▶ Sum across the row:

$$P(\text{cavity} \wedge \text{toothache}) + P(\text{cavity} \wedge \neg \text{toothache}) = 0.2$$

- ▶ How to compute $P(\text{cavity} \vee \text{toothache})$?
- ▶ Sum across atomic events:

$$P(\text{cavity} \wedge \text{toothache}) + P(\neg \text{cavity} \wedge \text{toothache}) + P(\text{cavity} \wedge \neg \text{toothache}) = 0.28$$

- ▶ How to compute $P(\text{cavity} | \text{toothache})$?
- ▶
$$\frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})}$$
- ▶ All relevant probabilities can be computed using the full joint probability distribution, by expressing propositions as disjunctions of atomic events.

Working with the Full Joint Probability Distribution??

- ▶ Question: Is it a good idea to use the full joint probability distribution?

Working with the Full Joint Probability Distribution??

- ▶ **Question:** Is it a good idea to use the full joint probability distribution?
- ▶ **Answer:** No:
 - ▶ Given n random variables with k values each, the full joint probability distribution contains k^n probabilities.
 - ▶ Computational cost of dealing with this size.
 - ▶ Practically impossible to assess all these probabilities.

Working with the Full Joint Probability Distribution??

- ▶ Question: Is it a good idea to use the full joint probability distribution?
- ▶ Answer: No:
 - ▶ Given n random variables with k values each, the full joint probability distribution contains k^n probabilities.
 - ▶ Computational cost of dealing with this size.
 - ▶ Practically impossible to assess all these probabilities.
- ▶ Question: So, is there a compact way to represent the full joint probability distribution? Is there an efficient method to work with that representation?

Working with the Full Joint Probability Distribution??

- ▶ Question: Is it a good idea to use the full joint probability distribution?
- ▶ Answer: No:
 - ▶ Given n random variables with k values each, the full joint probability distribution contains k^n probabilities.
 - ▶ Computational cost of dealing with this size.
 - ▶ Practically impossible to assess all these probabilities.
- ▶ Question: So, is there a compact way to represent the full joint probability distribution? Is there an efficient method to work with that representation?
- ▶ Answer: Not in general, but it works in many cases. We can work directly with conditional probabilities, and exploit conditional independence.
- ▶ Eventually: Bayesian networks. (First, we do the simple case)

- ▶ **Definition 4.2.** Events a and b are **independent** if $P(a \wedge b) = P(a) \cdot P(b)$.
- ▶ **Assertion** Given independent events a and b where $P(b) \neq 0$, we have $P(a|b) = P(a)$.

▶ **Proof:**

1. By definition, $P(a|b) = \frac{P(a \wedge b)}{P(b)}$,
2. which by **independence** is equal to $\frac{P(a) \cdot P(b)}{P(b)} = P(a)$.

□

- ▶ Similarly, if $P(a) \neq 0$, we have $P(b|a) = P(b)$.
- ▶ **Definition 4.3.** Random variables X and Y are **independent** if $P(X, Y) = P(X) \otimes P(Y)$. (System of equations given by outer product!)

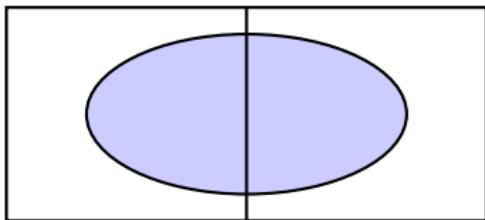
Independence (Examples)

► Example 4.4.

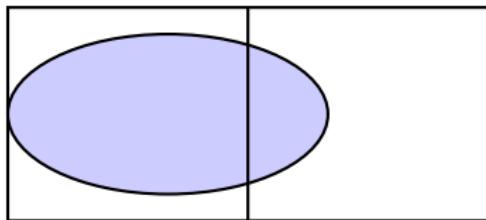
- $P(\text{Die1} = 6 \wedge \text{Die2} = 6) = 1/36$.
- $P(W = \text{sunny} | \text{headache}) = P(W = \text{sunny})$ (unless you're weather-sensitive; cf. slide 735)
- But toothache and cavity are NOT independent.
- The fraction of "cavity" is higher within "toothache" than within " \neg toothache".
 $P(\text{toothache}) = 0.2$ and $P(\text{cavity}) = 0.2$, but $P(\text{toothache} \wedge \text{cavity}) = 0.12 > 0.04$.

► Intuition:

Independent



Dependent



Oval independent of rectangle, iff split equally

Illustration: Exploiting Independence

- **Example 4.5.** Consider (again) the following full joint probability distribution:

	toothache	\neg toothache
cavity	0.12	0.08
\neg cavity	0.08	0.72

Adding variable Weather with values sunny, rain, cloudy, snow, the full joint probability distribution contains 16 probabilities.

But your teeth do not influence the weather, nor vice versa!

- Weather is independent of each of Cavity and Toothache: For all value combinations (c, t) of Cavity and Toothache, and for all values w of Weather, we have $P(c \wedge t \wedge w) = P(c \wedge t) \cdot P(w)$.
- $P(\text{Cavity, Toothache, Weather})$ can be reconstructed from the separate tables $P(\text{Cavity, Toothache})$ and $P(\text{Weather})$. (8 probabilities)
- Independence can be exploited to represent the full joint probability distribution more compactly.
- Sometimes, variables are independent only under particular conditions: conditional independence, see later.

5 Basic Probabilistic Reasoning Methods

The Product Rule

- ▶ **Definition 5.1.** The following identity is called the **product rule**:
Given propositions a and b , $P(a \wedge b) = P(a|b) \cdot P(b)$.
- ▶ **Example 5.2.** $P(\text{cavity} \wedge \text{toothache}) = P(\text{toothache}|\text{cavity}) \cdot P(\text{cavity})$.
- ▶ If we know the values of $P(a|b)$ and $P(b)$, then we can compute $P(a \wedge b)$.
- ▶ Similarly, $P(a \wedge b) = P(b|a) \cdot P(a)$.
- ▶ **Definition 5.3.** We use the **component wise array product** (bold dot)
 $P(X, Y) = P(X|Y) \cdot P(Y)$ as a summary notation for the equation system
 $P(x_i, y_j) = P(x_i|y_j) \cdot P(y_j)$ where i, j range over domain sizes of X and Y .
- ▶ **Example 5.4.** $P(\text{Weather}, \text{Ache}) = P(\text{Weather}|\text{Ache}) \cdot P(\text{Ache})$ is

$$P(W = \text{sunny} \wedge \text{ache}) = P(W = \text{sunny}|\text{ache}) \cdot P(\text{ache})$$

$$P(W = \text{rain} \wedge \text{ache}) = P(W = \text{rain}|\text{ache}) \cdot P(\text{ache})$$

$$\dots = \dots$$

$$P(W = \text{snow} \wedge \neg \text{ache}) = P(W = \text{snow}|\neg \text{ache}) \cdot P(\neg \text{ache})$$

- ▶ **Note:** The **outer product** in $P(X, Y) = P(X) \cdot P(Y)$ is just by coincidence, we will use $P(X, Y) = P(X) \cdot P(Y)$ instead.

The Chain Rule

- **Definition 5.5 (Chain Rule).** Given random variables X_1, \dots, X_n , we have

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_2 | X_1) \cdot P(X_1)$$

This identity is called the **chain rule**.

- **Example 5.6.**

$$\begin{aligned} & P(\neg\text{brush} \wedge \text{cavity} \wedge \text{toothache}) \\ &= P(\text{toothache} | \text{cavity}, \neg\text{brush}) \cdot P(\text{cavity}, (\neg\text{brush})) \\ &= P(\text{toothache} | \text{cavity}, \neg\text{brush}) \cdot P(\text{cavity} | \neg\text{brush}) \cdot P(\neg\text{brush}) \end{aligned}$$

- **Proof:** Iterated application of Product Rule

1. $P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1}, \dots, X_1)$ by Product Rule.
2. In turn, $P(X_{n-1}, \dots, X_1) = P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot P(X_{n-2}, \dots, X_1)$, etc.



- **Note:** This works for any ordering of the variables.

- We can recover the probability of atomic events from sequenced conditional probabilities for any ordering of the variables.
- First of the four basic techniques in Bayesian networks.

- ▶ Extracting a sub-distribution from a larger joint distribution:
- ▶ Assertion (Marginalization) Given sets X and Y of random variables, we have:

$$P(X) = \sum_{y \in Y} P(X, y)$$

where $\sum_{y \in Y}$ sums over all possible value combinations of Y.

- ▶ Example 5.7. (Note: Equation system!)

$$P(\text{Cavity}) = \sum_{y \in \text{Toothache}} P(\text{Cavity}, y)$$

$$P(\text{cavity}) = P(\text{cavity, toothache}) + P(\text{cavity, } (\neg\text{toothache}))$$

$$P(\neg\text{cavity}) = P(\neg\text{cavity, toothache}) + P(\neg\text{cavity, } (\neg\text{toothache}))$$

Normalization: Idea

- **Problem:** We know $P(\text{cavity} \wedge \text{toothache})$ but don't know $P(\text{toothache})$.

Normalization: Idea

- ▶ Problem: We know $P(\text{cavity} \wedge \text{toothache})$ but don't know $P(\text{toothache})$.
- ▶ Step 1: Case distinction over values of Cavity: ($P(\text{toothache})$ as an unknown)

$$P(\text{cavity}|\text{toothache}) = \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.12}{P(\text{toothache})}$$

$$P(\neg\text{cavity}|\text{toothache}) = \frac{P(\neg\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.08}{P(\text{toothache})}$$

Normalization: Idea

- ▶ Problem: We know $P(\text{cavity} \wedge \text{toothache})$ but don't know $P(\text{toothache})$.
- ▶ Step 1: Case distinction over values of Cavity: ($P(\text{toothache})$ as an unknown)

$$P(\text{cavity}|\text{toothache}) = \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.12}{P(\text{toothache})}$$

$$P(\neg\text{cavity}|\text{toothache}) = \frac{P(\neg\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.08}{P(\text{toothache})}$$

- ▶ Step 2: Assuming placeholder $\alpha := 1/P(\text{toothache})$:

$$P(\text{cavity}|\text{toothache}) = \alpha P(\text{cavity} \wedge \text{toothache}) = \alpha \cdot 0.12$$

$$P(\neg\text{cavity}|\text{toothache}) = \alpha P(\neg\text{cavity} \wedge \text{toothache}) = \alpha \cdot 0.08$$

Normalization: Idea

- ▶ Problem: We know $P(\text{cavity} \wedge \text{toothache})$ but don't know $P(\text{toothache})$.
- ▶ Step 1: Case distinction over values of Cavity: ($P(\text{toothache})$ as an unknown)

$$P(\text{cavity}|\text{toothache}) = \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.12}{P(\text{toothache})}$$

$$P(\neg\text{cavity}|\text{toothache}) = \frac{P(\neg\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.08}{P(\text{toothache})}$$

- ▶ Step 2: Assuming placeholder $\alpha := 1/P(\text{toothache})$:

$$P(\text{cavity}|\text{toothache}) = \alpha P(\text{cavity} \wedge \text{toothache}) = \alpha \cdot 0.12$$

$$P(\neg\text{cavity}|\text{toothache}) = \alpha P(\neg\text{cavity} \wedge \text{toothache}) = \alpha \cdot 0.08$$

- ▶ Step 3: Fixing toothache to be true, view $P(\text{cavity} \wedge \text{toothache})$ vs. $P(\neg\text{cavity} \wedge \text{toothache})$ as the “relative weights of $P(\text{cavity})$ vs. $P(\neg\text{cavity})$ within toothache”.

Then normalize their summed-up weight to 1:

$$1 = \alpha(0.12 + 0.08) \leadsto \alpha = \frac{1}{0.12 + 0.08} = \frac{1}{0.2} = 5$$

Normalization: Idea

- ▶ Problem: We know $P(\text{cavity} \wedge \text{toothache})$ but don't know $P(\text{toothache})$.
- ▶ Step 1: Case distinction over values of Cavity: ($P(\text{toothache})$ as an unknown)

$$P(\text{cavity}|\text{toothache}) = \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.12}{P(\text{toothache})}$$

$$P(\neg\text{cavity}|\text{toothache}) = \frac{P(\neg\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.08}{P(\text{toothache})}$$

- ▶ Step 2: Assuming placeholder $\alpha := 1/P(\text{toothache})$:

$$P(\text{cavity}|\text{toothache}) = \alpha P(\text{cavity} \wedge \text{toothache}) = \alpha \cdot 0.12$$

$$P(\neg\text{cavity}|\text{toothache}) = \alpha P(\neg\text{cavity} \wedge \text{toothache}) = \alpha \cdot 0.08$$

- ▶ Step 3: Fixing toothache to be true, view $P(\text{cavity} \wedge \text{toothache})$ vs. $P(\neg\text{cavity} \wedge \text{toothache})$ as the "relative weights of $P(\text{cavity})$ vs. $P(\neg\text{cavity})$ within toothache".

Then normalize their summed-up weight to 1:

$$1 = \alpha(0.12 + 0.08) \leadsto \alpha = \frac{1}{0.12 + 0.08} = \frac{1}{0.2} = 5$$

- ▶ α is a **normalization constant** scaling the sum of relative weights to 1.

Normalization: Formal

- ▶ **Definition 5.8.** Given a vector $\langle w_1, \dots, w_k \rangle$ of numbers in $[0,1]$ where $\sum_{i=1}^k w_i \leq 1$, the **normalization constant** α is $\alpha \langle w_1, \dots, w_k \rangle := \frac{1}{\sum_{i=1}^k w_i}$.
- ▶ **Note:** The condition $\sum_{i=1}^k w_i \leq 1$ is needed because these will be relative weights, i.e. case distinction over a subset of all worlds (the one fixed by the knowledge in our conditional probability).
- ▶ **Example 5.9.** $\alpha \langle 0.12, 0.08 \rangle = 5 \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle$.

- ▶ **Example 5.10.** $\alpha \langle P(\text{cavity} \wedge \text{toothache}), P(\neg \text{cavity} \wedge \text{toothache}) \rangle = \alpha \langle 0.12, 0.08 \rangle$, so $P(\text{cavity} | \text{toothache}) = 0.6$, and $P(\neg \text{cavity} | \text{toothache}) = 0.4$.
- ▶ Another way of saying this is: “We use α as a placeholder for $1/P(e)$, which we compute using the sum of relative weights by Marginalization.”
- ▶ **Computation Rule: Normalization+Marginalization**
Given “query variable” X , “observed event” e , and “hidden variables” set Y :

$$P(X|e) = \alpha \cdot P(X, e) = \alpha \cdot \left(\sum_{y \in Y} P(X, e, y) \right)$$

- ▶ Second of the four basic techniques in Bayesian networks.

Normalization: Formal

- ▶ **Definition 5.8.** Given a vector $\langle w_1, \dots, w_k \rangle$ of numbers in $[0,1]$ where $\sum_{i=1}^k w_i \leq 1$, the **normalization constant** α is $\alpha \langle w_1, \dots, w_k \rangle := \frac{1}{\sum_{i=1}^k w_i}$.
- ▶ **Note:** The condition $\sum_{i=1}^k w_i \leq 1$ is needed because these will be relative weights, i.e. case distinction over a subset of all worlds (the one fixed by the knowledge in our conditional probability).
- ▶ **Example 5.9.** $\alpha \langle 0.12, 0.08 \rangle = 5 \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle$.
- ▶ **Assertion (Normalization)** Given a random variable X and an event e , we have $P(X|e) = \alpha P(X, e)$.

Proof:

1. For each value x of X , $P(X = x|e) = P(X = x \wedge e)/P(e)$.
2. So all we need to prove is that $\alpha = 1/P(e)$.
3. By definition, $\alpha = 1/\sum_x P(X = x \wedge e)$, so we need to prove

$$P(e) = \sum_x P(X = x \wedge e)$$

which holds by marginalization.



- ▶ **Example 5.10.** $\alpha \langle P(\text{cavity} \wedge \text{toothache}), P(\neg \text{cavity} \wedge \text{toothache}) \rangle = \alpha \langle 0.12, 0.08 \rangle$, so $P(\text{cavity}|\text{toothache}) = 0.6$, and $P(\neg \text{cavity}|\text{toothache}) = 0.4$.
- ▶ **FAU** Another way of saying this is: "We use α as a placeholder for $1/P(e)$, which

6 Bayes' Rule

- ▶ **Definition 6.1 (Bayes' Rule).** Given propositions A and B where $P(a) \neq 0$ and $P(b) \neq 0$, we have:

$$P(a|b) = \frac{P(b|a) \cdot P(a)}{P(b)}$$

This equation is called **Bayes' rule**.

- ▶ **Proof:**

1. By definition, $P(a|b) = \frac{P(a \wedge b)}{P(b)}$
2. by the product rule $P(a \wedge b) = P(b|a) \cdot P(a)$ is equal to the claim.

□

- ▶ **Notation:** This is a system of equations!

$$P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)}$$

Applying Bayes' Rule

- ▶ **Example 6.2.** Say we know that $P(\text{toothache}|\text{cavity}) = 0.6$, $P(\text{cavity}) = 0.2$, and $P(\text{toothache}) = 0.2$.
We can we compute $P(\text{cavity}|\text{toothache})$: By Bayes' rule,
$$P(\text{cavity}|\text{toothache}) = \frac{P(\text{toothache}|\text{cavity}) \cdot P(\text{cavity})}{P(\text{toothache})} = \frac{0.6 \cdot 0.2}{0.2} = 0.6.$$
- ▶ Ok, but: Why don't we simply assess $P(\text{cavity}|\text{toothache})$ directly?
- ▶ **Definition 6.3.** $P(\text{toothache}|\text{cavity})$ is **causal**, $P(\text{cavity}|\text{toothache})$ is **diagnostic**.
- ▶ **Intuition:** Causal dependencies are robust over frequency of the causes.
- ▶ **Example 6.4.** If there is a cavity epidemic then $P(\text{cavity}|\text{toothache})$ increases, but $P(\text{toothache}|\text{cavity})$ remains the same. (only depends on how cavities "work")
- ▶ Also, causal dependencies are often easier to **assess**.
- ▶ **Intuition:** "reason about causes in order to draw conclusions about symptoms".
- ▶ Bayes' rule allows to perform diagnosis (observing a symptom, what is the cause?) based on prior probabilities and causal dependencies.

Extended Example: Bayes' Rule and Meningitis

- Facts known to doctors:

- The prior probabilities of meningitis (m) and stiff neck (s) are $P(m) = 0.00002$ and $P(s) = 0.01$.
- Meningitis causes a stiff neck 70% of the time: $P(s|m) = 0.7$.

- Doctor d uses Bayes' Rule:

$$P(m|s) = \frac{P(s|m) \cdot P(m)}{P(s)} = \frac{0.7 \cdot 0.00002}{0.01} = 0.0014 \sim \frac{1}{700}.$$

- Even though stiff neck is strongly indicated by meningitis $(P(s|m) = 0.7)$
- the probability of meningitis in the patient remains small.
- The prior probability of stiff necks is much higher than that of meningitis.
- Doctor d' knows $P(m|s)$ from observation; she does not need Bayes' rule!
- Indeed, but what if a meningitis epidemic erupts
- Then d knows that $P(m|s)$ grows proportionally with $P(m)$ $(d'$ clueless)

7 Conditional Independence

Bayes' Rule with Multiple Evidence

- **Example 7.1.** Say we know from medicinical studies that $P(\text{cavity}) = 0.2$, $P(\text{toothache}|\text{cavity}) = 0.6$, $P(\text{toothache}|\neg\text{cavity}) = 0.1$, $P(\text{catch}|\text{cavity}) = 0.9$, and $P(\text{catch}|\neg\text{cavity}) = 0.2$.

Now, in case we did observe the symptoms toothache and catch (the dentist's probe catches in the aching tooth), what would be the likelihood of having a cavity? What is $P(\text{cavity}|(\text{toothache}\wedge\text{catch}))$?

- **Trial 1:** Bayes' rule

$$P(\text{cavity}|(\text{toothache}\wedge\text{catch})) = \frac{P(\text{toothache}\wedge\text{catch}|\text{cavity}) \cdot P(\text{cavity})}{P(\text{toothache}\wedge\text{catch})}$$

Bayes' Rule with Multiple Evidence

- **Example 7.1.** Say we know from medicinical studies that $P(\text{cavity}) = 0.2$, $P(\text{toothache}|\text{cavity}) = 0.6$, $P(\text{toothache}|\neg\text{cavity}) = 0.1$, $P(\text{catch}|\text{cavity}) = 0.9$, and $P(\text{catch}|\neg\text{cavity}) = 0.2$.

Now, in case we did observe the symptoms toothache and catch (the dentist's probe catches in the aching tooth), what would be the likelihood of having a cavity? What is $P(\text{cavity}|(\text{toothache}\wedge\text{catch}))$?

- **Trial 1:** Bayes' rule

$$P(\text{cavity}|(\text{toothache}\wedge\text{catch})) = \frac{P(\text{toothache}\wedge\text{catch}|\text{cavity}) \cdot P(\text{cavity})}{P(\text{toothache}\wedge\text{catch})}$$

- **Trial 2:** Normalization $P(X|e) = \alpha P(X, e)$ then Product Rule

$P(X, e) = P(e|X) \cdot P(X)$, with $X = \text{Cavity}$, $e = \text{toothache} \wedge \text{catch}$:

$$P(\text{Cavity}|(\text{catch}\wedge\text{toothache})) = \alpha \cdot P(\text{toothache}\wedge\text{catch}|\text{Cavity}) \cdot P(\text{Cavity})$$

$$P(\text{cavity}|(\text{catch}\wedge\text{toothache})) = \alpha \cdot P(\text{toothache}\wedge\text{catch}|\text{cavity}) \cdot P(\text{cavity})$$

$$P(\neg\text{cavity}|(\text{catch}\wedge\text{toothache})) = \alpha P(\text{toothache}\wedge\text{catch}|\neg\text{cavity}) P(\neg\text{cavity})$$

- ▶ $P(\text{Cavity} | (\text{toothache} \wedge \text{catch})) = \alpha P(\text{toothache} \wedge \text{catch} | \text{Cavity}) \cdot P(\text{Cavity})$
- ▶ **Question:** So, is everything fine?

- ▶ $P(\text{Cavity} | (\text{toothache} \wedge \text{catch})) = \alpha P(\text{toothache} \wedge \text{catch} | \text{Cavity}) \cdot P(\text{Cavity})$
- ▶ **Question:** So, is everything fine?
- ▶ **Answer:** No! We need $P(\text{toothache} \wedge \text{catch} | \text{Cavity})$, i.e. causal dependencies for all combinations of symptoms! (» 2, in general)

- ▶ $P(\text{Cavity} | (\text{toothache} \wedge \text{catch})) = \alpha P(\text{toothache} \wedge \text{catch} | \text{Cavity}) \cdot P(\text{Cavity})$
- ▶ **Question:** So, is everything fine?
- ▶ **Answer:** No! We need $P(\text{toothache} \wedge \text{catch} | \text{Cavity})$, i.e. causal dependencies for all combinations of symptoms! (» 2, in general)
- ▶ **Question:** Are Toothache and Catch independent?

- ▶ $P(\text{Cavity} | (\text{toothache} \wedge \text{catch})) = \alpha P(\text{toothache} \wedge \text{catch} | \text{Cavity}) \cdot P(\text{Cavity})$
- ▶ **Question:** So, is everything fine?
- ▶ **Answer:** No! We need $P(\text{toothache} \wedge \text{catch} | \text{Cavity})$, i.e. causal dependencies for all combinations of symptoms! ($\gg 2$, in general)
- ▶ **Question:** Are Toothache and Catch independent?
- ▶ **Answer:** No. If a probe catches, we probably have a cavity which probably causes toothache.
- ▶ **But:** They are conditionally independent given the presence or absence of a cavity!

Conditional Independence

- **Definition 7.2.** Given sets of random variables Z_1 , Z_2 , and Z , we say that Z_1 and Z_2 are **conditionally independent** given Z if:

$$P(Z_1, Z_2 | Z) = P(Z_1 | Z) \cdot P(Z_2 | Z)$$

We alternatively say that Z_1 is **conditionally independent** of Z_2 given Z .

- **Example 7.3.** Catch and Toothache are **conditionally independent** given Cavity.

- For cavity: this may cause both, but they don't influence each other.
- For \neg cavity: something else causes catch and/or toothache.

So we have:

$$P(\text{Toothache}, \text{Catch} | \text{cavity}) = P(\text{Toothache} | \text{cavity}) \cdot P(\text{Catch} | \text{cavity})$$

$$P(\text{Toothache}, \text{Catch} | \neg \text{cavity}) = P(\text{Toothache} | \neg \text{cavity}) \cdot P(\text{Catch} | \neg \text{cavity})$$

- **Note:** The definition is symmetric regarding the roles of Z_1 and Z_2 : Toothache is **conditionally independent** of Cavity.
- But there may be dependencies *within* Z_1 or Z_2 , e.g.
 $Z_2 = \{\text{Toothache}, \text{Sleeplessness}\}$.

Conditional Independence, ctd.

- ▶ Assertion If Z_1 and Z_2 are conditionally independent given Z , then $P(Z_1|Z_2, Z) = P(Z_1|Z)$.

▶ Proof:

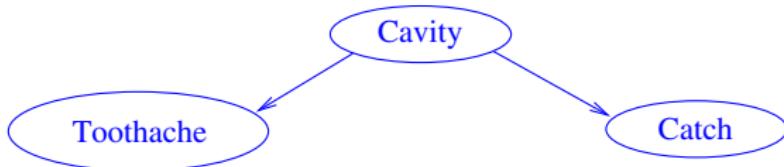
1. By definition, $P(Z_1|Z_2, Z) = \frac{P(Z_1, Z_2, Z)}{P(Z_2, Z)}$
2. which by product rule is equal to $\frac{P(Z_1, Z_2|Z) \cdot P(Z)}{P(Z_2, Z)}$
3. which by conditional independence is equal to $\frac{P(Z_1|Z) \cdot P(Z_2|Z) \cdot P(Z)}{P(Z_2, Z)}$.
4. Since $\frac{P(Z_2|Z) \cdot P(Z)}{P(Z_2, Z)} = 1$ this proves the claim.

□

- ▶ Example 7.4. Using $\{\text{Toothache}\}$ as Z_1 , $\{\text{Catch}\}$ as Z_2 , and $\{\text{Cavity}\}$ as Z : $P(\text{Toothache}|\text{Catch}, \text{Cavity}) = P(\text{Toothache}|\text{Cavity})$.
- ▶ In the presence of conditional independence, we can drop variables from the right-hand side of conditional probabilities.
- ▶ Third of the four basic techniques in Bayesian networks.
- ▶ Last missing technique: “Capture variable dependencies in a graph”; illustration see next slide, details see

Exploiting Conditional Independence: Overview

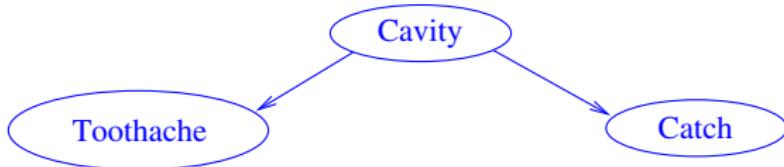
- 1. Graph captures variable dependencies: (Variables X_1, \dots, X_n)



- Given evidence e , want to know $P(X|e)$.
- Remaining vars: Y .

Exploiting Conditional Independence: Overview

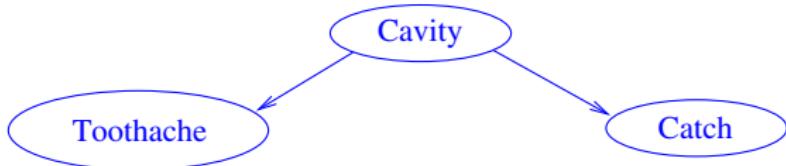
- 1. Graph captures variable dependencies: (Variables X_1, \dots, X_n)



- Given evidence e , want to know $P(X|e)$.
- Remaining vars: Y .
- 2. Normalization+Marginalization:
$$P(X|e) = \alpha \cdot P(X, e); \text{ if } Y \neq \emptyset \text{ then } P(X|e) = \alpha \cdot (\sum_{y \in Y} P(X, e, y))$$
- A sum over atomic events!

Exploiting Conditional Independence: Overview

- 1. Graph captures variable dependencies: (Variables X_1, \dots, X_n)



- Given evidence e , want to know $P(X|e)$.
- Remaining vars: Y .

- 2. Normalization+Marginalization:

$$P(X|e) = \alpha \cdot P(X, e); \text{ if } Y \neq \emptyset \text{ then } P(X|e) = \alpha \cdot (\sum_{y \in Y} P(X, e, y))$$

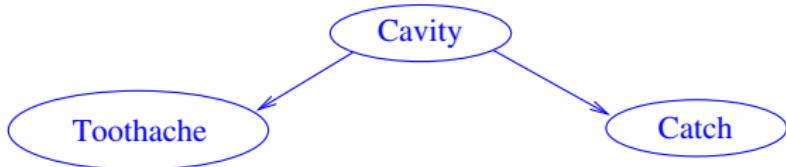
- A sum over atomic events!

- 3. Chain rule: Order X_1, \dots, X_n consistently with dependency graph.

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

Exploiting Conditional Independence: Overview

- 1. Graph captures variable dependencies: (Variables X_1, \dots, X_n)



- Given evidence e , want to know $P(X|e)$.
- Remaining vars: Y .

- 2. Normalization+Marginalization:

$$P(X|e) = \alpha \cdot P(X, e); \text{ if } Y \neq \emptyset \text{ then } P(X|e) = \alpha \cdot (\sum_{y \in Y} P(X, e, y))$$

- A sum over atomic events!

- 3. Chain rule: Order X_1, \dots, X_n consistently with dependency graph.

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

- 4. Exploit Conditional Independence: Instead of $P(X_i | X_{i-1}, \dots, X_1)$, with previous slide we can use $P(X_i | \text{Parents}(X_i))$.

- Bayesian networks!

Exploiting Conditional Independence: Example

- ▶ 1. Graph captures variable dependencies: (See previous slide.)
- ▶ Given toothache, catch, want $P(\text{Cavity}|\text{toothache}, \text{catch})$. Remaining vars: \emptyset .

Exploiting Conditional Independence: Example

- ▶ 1. Graph captures variable dependencies: (See previous slide.)
 - ▶ Given toothache, catch, want $P(\text{Cavity}|\text{toothache}, \text{catch})$. Remaining vars: \emptyset .
- ▶ 2. Normalization+Marginalization:

$$P(\text{Cavity}|\text{toothache}, \text{catch}) = \alpha \cdot P(\text{Cavity}, \text{toothache}, \text{catch})$$

Exploiting Conditional Independence: Example

- ▶ 1. Graph captures variable dependencies: (See previous slide.)
 - ▶ Given toothache, catch, want $P(\text{Cavity}|\text{toothache}, \text{catch})$. Remaining vars: \emptyset .
- ▶ 2. Normalization+Marginalization:
$$P(\text{Cavity}|\text{toothache}, \text{catch}) = \alpha \cdot P(\text{Cavity}, \text{toothache}, \text{catch})$$
- ▶ 3. Chain rule: Order $X_1 = \text{Cavity}$, $X_2 = \text{Toothache}$, $X_3 = \text{Catch}$.
$$P(\text{Cavity}, \text{toothache}, \text{catch}) =$$
$$P(\text{catch}|\text{toothache}, \text{Cavity}) \cdot P(\text{toothache}|\text{Cavity}) \cdot P(\text{Cavity})$$

Exploiting Conditional Independence: Example

- ▶ 1. Graph captures variable dependencies: (See previous slide.)
 - ▶ Given toothache, catch, want $P(\text{Cavity}|\text{toothache}, \text{catch})$. Remaining vars: \emptyset .
- ▶ 2. Normalization+Marginalization:
$$P(\text{Cavity}|\text{toothache}, \text{catch}) = \alpha \cdot P(\text{Cavity}, \text{toothache}, \text{catch})$$
- ▶ 3. Chain rule: Order $X_1 = \text{Cavity}$, $X_2 = \text{Toothache}$, $X_3 = \text{Catch}$.
$$P(\text{Cavity}, \text{toothache}, \text{catch}) =$$
$$P(\text{catch}|\text{toothache}, \text{Cavity}) \cdot P(\text{toothache}|\text{Cavity}) \cdot P(\text{Cavity})$$
- ▶ 4. Exploit Conditional independence:

Instead of $P(\text{catch}|\text{toothache}, \text{Cavity})$ use $P(\text{catch}|\text{Cavity})$.

Exploiting Conditional Independence: Example

- ▶ 1. Graph captures variable dependencies: (See previous slide.)
 - ▶ Given toothache, catch, want $P(\text{Cavity}|\text{toothache}, \text{catch})$. Remaining vars: \emptyset .
- ▶ 2. Normalization+Marginalization:
$$P(\text{Cavity}|\text{toothache}, \text{catch}) = \alpha \cdot P(\text{Cavity}, \text{toothache}, \text{catch})$$
- ▶ 3. Chain rule: Order $X_1 = \text{Cavity}$, $X_2 = \text{Toothache}$, $X_3 = \text{Catch}$.
$$P(\text{Cavity}, \text{toothache}, \text{catch}) =$$
$$P(\text{catch}|\text{toothache}, \text{Cavity}) \cdot P(\text{toothache}|\text{Cavity}) \cdot P(\text{Cavity})$$
- ▶ 4. Exploit Conditional independence:
Instead of $P(\text{catch}|\text{toothache}, \text{Cavity})$ use $P(\text{catch}|\text{Cavity})$.
- ▶ Thus:

$$\begin{aligned} & P(\text{Cavity}|\text{toothache}, \text{catch}) \\ &= \alpha \cdot P(\text{catch}|\text{Cavity}) \cdot P(\text{toothache}|\text{Cavity}) \cdot P(\text{Cavity}) \\ &= \alpha \cdot \langle 0.9 \cdot 0.6 \cdot 0.2, 0.2 \cdot 0.1 \cdot 0.8 \rangle \\ &= \alpha \cdot \langle 0.108, 0.016 \rangle \end{aligned}$$

Exploiting Conditional Independence: Example

- ▶ 1. Graph captures variable dependencies: (See previous slide.)
 - ▶ Given toothache, catch, want $P(\text{Cavity}|\text{toothache}, \text{catch})$. Remaining vars: \emptyset .

- ▶ 2. Normalization+Marginalization:

$$P(\text{Cavity}|\text{toothache}, \text{catch}) = \alpha \cdot P(\text{Cavity}, \text{toothache}, \text{catch})$$

- ▶ 3. Chain rule: Order $X_1 = \text{Cavity}$, $X_2 = \text{Toothache}$, $X_3 = \text{Catch}$.

$$P(\text{Cavity}, \text{toothache}, \text{catch}) =$$

$$P(\text{catch}|\text{toothache}, \text{Cavity}) \cdot P(\text{toothache}|\text{Cavity}) \cdot P(\text{Cavity})$$

- ▶ 4. Exploit Conditional independence:

Instead of $P(\text{catch}|\text{toothache}, \text{Cavity})$ use $P(\text{catch}|\text{Cavity})$.

- ▶ Thus:

$$P(\text{Cavity}|\text{toothache}, \text{catch})$$

$$= \alpha \cdot P(\text{catch}|\text{Cavity}) \cdot P(\text{toothache}|\text{Cavity}) \cdot P(\text{Cavity})$$

$$= \alpha \cdot \langle 0.9 \cdot 0.6 \cdot 0.2, 0.2 \cdot 0.1 \cdot 0.8 \rangle$$

$$= \alpha \cdot \langle 0.108, 0.016 \rangle$$

- ▶ So: $\alpha \approx 8.06$ and $P(\text{cavity}|(\text{toothache} \wedge \text{catch})) \approx 0.87$.

- ▶ **Definition 7.5.** A Bayesian network in which a single cause directly influences a number of effects, all of which are conditionally independent, given the cause is called a **naive Bayes model** or **Bayesian classifier**.
- ▶ **Observation 7.6.** In a naive Bayes model, the full joint probability distribution can be written as

$$P(\text{cause} | \text{effect}_1, \dots, \text{effect}_n) = P(\text{cause}) \cdot \prod_i P(\text{effect}_i | \text{cause})$$

- ▶ **Note:** This kind of model is called “naive” since it is often used as a simplifying model if the effects are not conditionally independent after all.
- ▶ It is also called **idiot Bayes model** by Bayesian fundamentalists.
- ▶ In practice, naive Bayes models can work surprisingly well, even when the conditional independence assumption is not true.
- ▶ **Example 7.7.** The dentistry example is a (true) naive Bayes model.

8 The Wumpus World Revisited

Wumpus World Revisited

- ▶ Example 8.1 (The Wumpus is Back).
- ▶ We have a maze where
 - ▶ pits cause a breeze in neighboring squares
 - ▶ Every square except (1,1) has a 20% pit probability.
(unfair otherwise)
 - ▶ we forget wumpus and gold for now
- ▶ Where does the agent should go, if there is breeze at (1,2) and (2,1)?
- ▶ Pure logical inference can conclude nothing about which square is most likely to be safe!
- ▶ Idea: Let's evaluate our probabilistic reasoning machinery, if that can help!

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 B OK	2,2	3,2	4,2
1,1	2,1 B OK	3,1	4,1

Wumpus: Probabilistic Model

► Boolean random variables

(only for the observed squares)

- $P_{i,j}$: pit at square (i,j)
- $B_{i,j}$: breeze at square (i,j)

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 B OK	2,2	3,2	4,2
1,1 OK	2,1 B OK	3,1	4,1

► Full joint probability distribution

1. $P(P_{1,1}, \dots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1}) = P(B_{1,1}, B_{1,2}, B_{2,1} | P_{1,1}, \dots, P_{4,4}) \cdot P(P_{1,1}, \dots, P_{4,4})$
(Product Rule)
2. $P(P_{1,1}, \dots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} P(P_{i,j})$
(pits are spread independently)
3. For a particular configuration $p_{1,1}, \dots, p_{4,4}$ with $p_{i,j} \in \{T, F\}$, n pits, and $P(p_{i,j}) = 0.2$ we have $P(p_{1,1}, \dots, p_{4,4}) = 0.2^n \cdot 0.8^{(16-n)}$

Wumpus: Query and Simple Reasoning

We have evidence in our example:

► ► $b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$ and

► $\kappa = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$

We are interested in answering queries such as

$P(P_{1,3} | \kappa, b)$. (pit in (1, 3) given evidence)

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 B OK	2,2	3,2	4,2
1,1 OK	2,1 B OK	3,1	4,1

► **Observation:** The answer can be computed by enumeration of the full joint probability distribution.

► **Standard Approach:** Let U be the variables $P_{i,j}$ except $P_{1,3}$ and κ , then

$$P(P_{1,3} | \kappa, b) = \sum_{u \in U} P(P_{1,3}, u, \kappa, b)$$

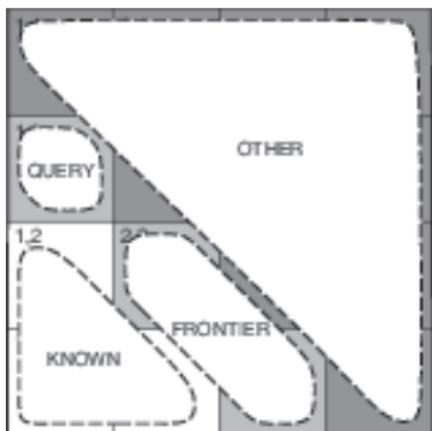
► **Problem:** Need to explore all possible values of variables in U ($2^{12} = 4096$ terms!)

► Can we do better? (faster; with less computation)

Wumpus: Conditional Independence

► *Observation 8.2.*

The observed breezes are **conditionally independent** of the other variables given the known, frontier, and query variables.



- We split the set of hidden variables into fringe and other variables: $U = F \cup O$ where F is the fringe and O the rest.
- **Corollary 8.3.** $P(b|P_{1,3}, \kappa, U) = P(b|P_{1,3}, \kappa, F)$ (**by conditional independence**)
- **Now:** let us exploit this formula.

Wumpus: Reasoning

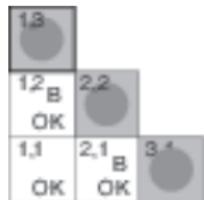
- We calculate:

$$\begin{aligned} P(P_{1,3}|\kappa, b) &= \alpha \left(\sum_{u \in U} P(P_{1,3}, u, \kappa, b) \right) \\ &= \alpha \left(\sum_{u \in U} P(b|P_{1,3}, \kappa, u) \cdot P(P_{1,3}, \kappa, u) \right) \\ &= \alpha \left(\sum_{f \in F} \sum_{o \in O} P(b|P_{1,3}, \kappa, f, o) \cdot P(P_{1,3}, \kappa, f, o) \right) \\ &= \alpha \left(\sum_{f \in F} P(b|P_{1,3}, \kappa, f) \cdot \left(\sum_{o \in O} P(P_{1,3}, \kappa, f, o) \right) \right) \\ &= \alpha \left(\sum_{f \in F} P(b|P_{1,3}, \kappa, f) \cdot \left(\sum_{o \in O} P(P_{1,3}) \cdot P(\kappa) \cdot P(f) \cdot P(o) \right) \right) \\ &= \alpha P(P_{1,3}) P(\kappa) \left(\sum_{f \in F} P(b|P_{1,3}, \kappa, f) \cdot P(f) \cdot \left(\sum_{o \in O} P(o) \right) \right) \\ &= \alpha' P(P_{1,3}) \left(\sum_{f \in F} P(b|P_{1,3}, \kappa, f) \cdot P(f) \right) \end{aligned}$$

for $\alpha' := \alpha P(\kappa)$ as $\sum_{o \in O} P(o) = 1$.

Wumpus: Solution

- We calculate using the **product rule** and **conditional independence** (see above)
 $P(P_{1,3}|\kappa, b) = \alpha' \cdot P(P_{1,3}) \cdot (\sum_{f \in F} P(b|P_{1,3}, \kappa, f) \cdot P(f))$
- Let us explore possible models (values) of Fringe that are F compatible with observation b .

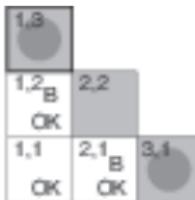


$$0.2 \times 0.2 = 0.04$$



$$0.2 \times 0.8 = 0.16$$

(a)



$$0.8 \times 0.2 = 0.16$$



$$0.2 \times 0.2 = 0.04$$



$$0.2 \times 0.8 = 0.16$$

(b)

- $P(P_{1,3}|\kappa, b) = \alpha' \cdot \langle 0.2 \cdot (0.04 + 0.16 + 0.16), 0.8 \cdot (0.04 + 0.16) \rangle = \langle 0.31, 0.69 \rangle$
- $P(P_{3,1}|\kappa, b) = \langle 0.31, 0.69 \rangle$ by symmetry
- $P(P_{2,2}|\kappa, b) = \langle 0.86, 0.14 \rangle$ (definitely avoid)

9 Conclusion

- ▶ **Uncertainty** is unavoidable in many environments, namely whenever agents do not have perfect knowledge.
- ▶ **Probabilities** express the degree of belief of an agent, given its knowledge, into an event.
- ▶ **Conditional probabilities** express the likelihood of an **event** given observed evidence.
- ▶ **Assessing** a probability $\hat{=}$ use statistics to approximate the likelihood of an **event**.
- ▶ **Bayes' rule** allows us to derive, from probabilities that are easy to assess, probabilities that aren't easy to **assess**.
- ▶ Given **multiple evidence**, we can exploit **conditional independence**.
- ▶ **Bayesian networks** (up next) do this, in a comprehensive, computational manner.

Chapter 21 Probabilistic Reasoning: Bayesian Networks

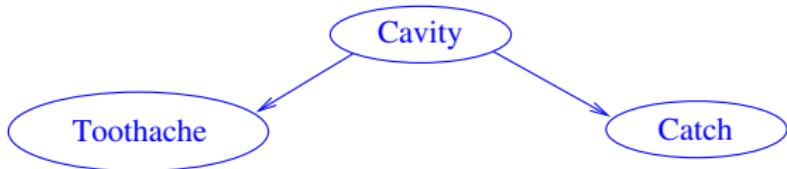
1 Introduction

Reminder: Our Agenda for This Topic

- ▶ Our treatment of the topic “Probabilistic Reasoning” consists of this and last document.
- ▶ Last chapter: All the basic machinery at use in [Bayesian networks](#).
- ▶ [This document: Bayesian networks](#): What they are, how to build them, how to use them.
- ▶ The most wide-spread and successful practical framework for probabilistic reasoning.

Reminder: Our Machinery

1. Graph captures variable dependencies: (Variables X_1, \dots, X_n)



- ▶ Given evidence e , want to know $P(X|e)$. Remaining vars: Y .

2. Normalization+Marginalization:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_{y \in Y} P(X, e, y)$$

- ▶ A sum over atomic events!

3. Chain rule: X_1, \dots, X_n consistently with dependency graph.

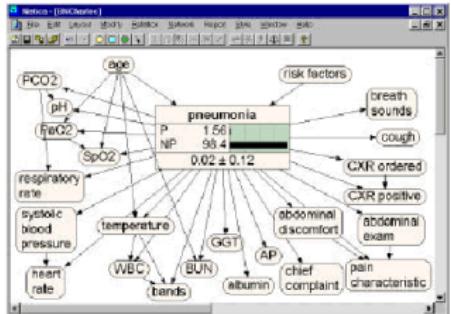
$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

4. Exploit conditional independence: Instead of $P(X_i | X_{i-1}, \dots, X_1)$, we can use $P(X_i | \text{Parents}(X_i))$.

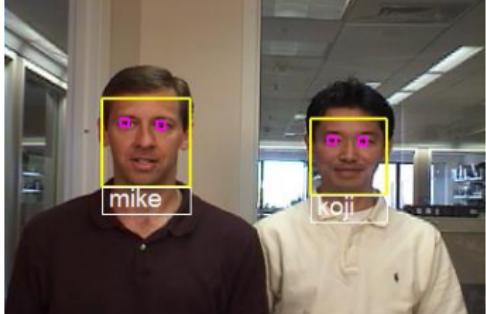
- ▶ Bayesian networks!

Some Applications

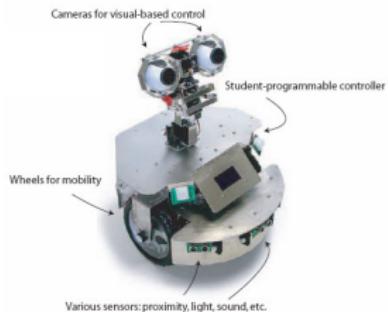
- A ubiquitous problem: Observe “symptoms”, need to infer “causes”.
Medical Diagnosis



Face Recognition



Self-Localization



Nuclear Test Ban



Our Agenda for This Chapter

- ▶ What is a Bayesian Network?: i.e. What is the syntax?
 - ▶ Tells you what Bayesian networks look like.
- ▶ What is the Meaning of a Bayesian Network?: What is the semantics?
 - ▶ Makes the intuitive meaning precise.
- ▶ Constructing Bayesian Networks: How do we design these networks? What effect do our choices have on their size?
 - ▶ Before you can start doing inference, you need to model your domain.
- ▶ Inference in Bayesian Networks: How do we use these networks? What is the associated complexity?
 - ▶ Inference is our primary purpose. It is important to understand its complexities and how it can be improved.

2 What is a Bayesian Network?

What is a Bayesian Network? (Short: BN)

- ▶ What do the others say?
 - ▶ “A *Bayesian network* is a methodology for representing the *full joint probability distribution*. In some cases, that representation is compact.”
 - ▶ “A *Bayesian network* is a graph whose nodes are *random variables* X_i and whose edges $\langle X_j, X_i \rangle$ denote a direct influence of X_j on X_i . Each node X_i is associated with a *conditional probability table (CPT)*, specifying $P(X_i | \text{Parents}(X_i))$.”
 - ▶ “A *Bayesian network* is a graphical way to depict conditional independence relations within a set of *random variables*.”
- ▶ A *Bayesian network* (BN) represents the structure of a given domain. Probabilistic inference exploits that structure for improved efficiency.
- ▶ BN inference: Determine the distribution of a query variable X given observed evidence e : $P(X|e)$.

- ▶ **Example 2.1 (From Russell/Norvig).**
 - ▶ I got very valuable stuff at home. So I bought an alarm. Unfortunately, the alarm just rings at home, doesn't call me on my mobile.
 - ▶ I've got two neighbors, Mary and John, who'll call me if they hear the alarm.
 - ▶ The problem is that, sometimes, the alarm is caused by an earthquake.
 - ▶ Also, John might confuse the alarm with his telephone, and Maria might miss the alarm altogether because she typically listens to loud music.
- ▶ **Question:** Given that both John and Mary call me, what is the probability of a burglary?

John, Mary, and My Alarm: Designing the Network

► Cooking Recipe:

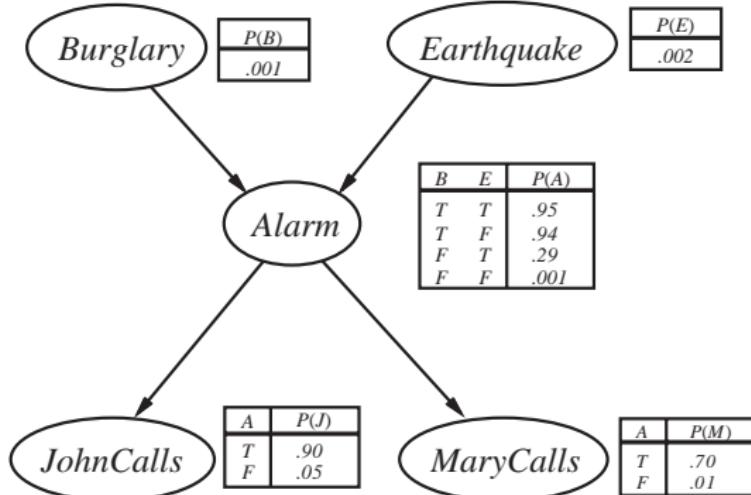
- (1) Design the random variables X_1, \dots, X_n ;
- (2) Identify their dependencies;
- (3) Insert the conditional probability tables $P(X_i | \text{Parents}(X_i))$.

► Example 2.2 (Let's cook!).

- (1) Random variables: Burglary, Earthquake, Alarm, JohnCalls, MaryCalls.
- (2) Dependencies: Burglaries and earthquakes are independent (this is actually debatable \leadsto design decision!) the alarm might be activated by either. John and Mary call if and only if they hear the alarm (they don't care about earthquakes)
- (3) Conditional probability tables: Assess the probabilities, see next slide.

John, Mary, and My Alarm: The Bayesian network

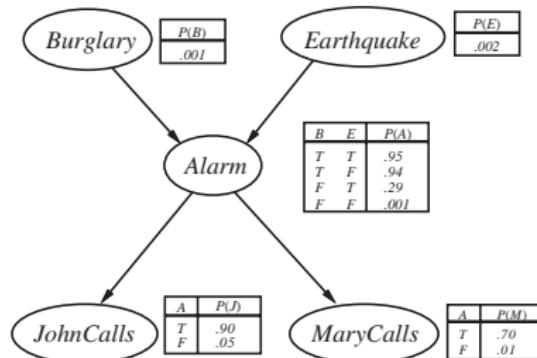
► Example 2.3.



- Note: In each $P(X_i|\text{Parents}(X_i))$, we show only $P(X_i = \text{T}|\text{Parents}(X_i))$. We don't show $P(X_i = \text{F}|\text{Parents}(X_i))$ which is $1 - P(X_i = \text{T}|\text{Parents}(X_i))$.

The Syntax of Bayesian Networks

► Definition 2.4 (Bayesian Network).



Given random variables X_1, \dots, X_n with finite domains D_1, \dots, D_n , a **Bayesian network** (also **belief network** or **probabilistic network**) is a **DAG** $\mathcal{B} := \langle \{X_1, \dots, X_n\}, E \rangle$. We denote $\text{Parents}(X_i) := \{X_j | (X_j, X_i) \in E\}$. Each X_i is associated with a function

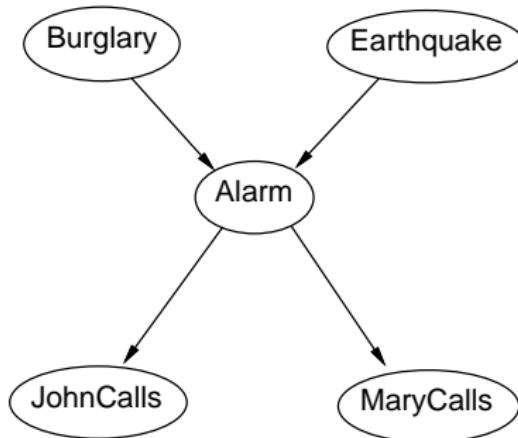
$$\text{CPT}(X_i) : D_i \times \prod_{X_j \in \text{Parents}(X_i)} D_j \rightarrow [0,1]$$

the **conditional probability table**.

► Definition 2.5. Related formalisms summed up under the term **graphical models**.

3 What is the Meaning of a Bayesian Network?

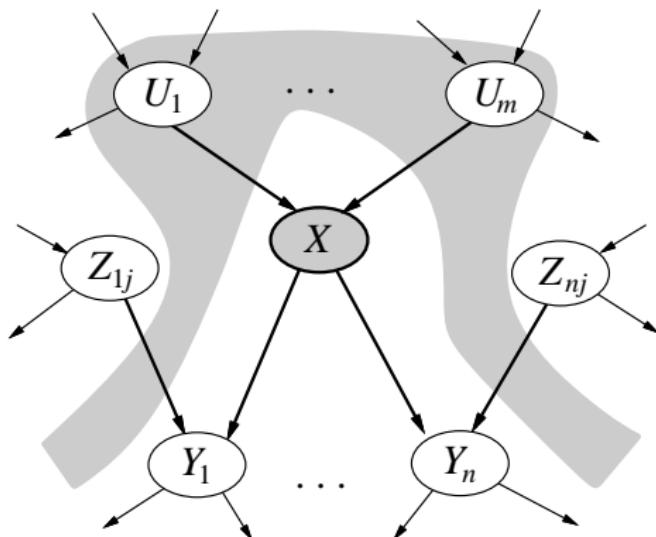
The Semantics of Bayesian Networks: Illustration



- ▶ Alarm depends on Burglary and Earthquake.
- ▶ MaryCalls only depends on Alarm.
 $P(\text{MaryCalls}|\text{Alarm}, \text{Burglary}) = P(\text{MaryCalls}|\text{Alarm})$
- ▶ Bayesian networks represent sets of independence assumptions.

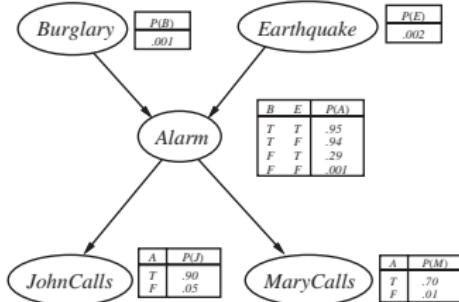
The Semantics of Bayesian Networks: Illustration, ctd.

- ▶ **Observation 3.1.** Each node X in a BN is conditionally independent of its non-descendants given its parents $\text{Parents}(X)$.



- ▶ **Question:** Why non-descendants of X ?
- ▶ **Intuition:** Given that BNs are acyclic, these are exactly those nodes that could have an edge into X .

The Semantics of Bayesian Networks: Formal



- ▶ **Definition 3.2.** Let $\langle \mathcal{X}, E \rangle$ be a Bayesian network, $X \in \mathcal{X}$, and E^* the transitive reflexive closure of E , then $\text{NonDesc}(X) := \{Y | (X, Y) \notin E^*\} \setminus \text{Parents}(X)$ is the set of non-descendants of X .
- ▶ **Definition 3.3.** Given a Bayesian network $\mathcal{B} := \langle \mathcal{X}, E \rangle$, we identify \mathcal{B} with the following two assumptions:
 - (A) $X \in \mathcal{X}$ is conditionally independent of $\text{NonDesc}(X)$ given $\text{Parents}(X)$.
 - (B) For all values x of $X \in \mathcal{X}$, and all value combinations of $\text{Parents}(X)$, we have $P(x | \text{Parents}(X)) = \text{CPT}(x, \text{Parents}(X))$.

Recovering the Full Joint Probability Distribution

- ▶ Intuition: A Bayesian network is a methodology for representing the full joint probability distribution.
- ▶ Problem: How to recover the full joint probability distribution $P(X_1, \dots, X_n)$ from $\mathcal{B} := \langle \{X_1, \dots, X_n\}, E \rangle$?

Recovering the Full Joint Probability Distribution

- ▶ Intuition: A Bayesian network is a methodology for representing the full joint probability distribution.
- ▶ Problem: How to recover the full joint probability distribution $P(X_1, \dots, X_n)$ from $\mathcal{B} := \langle \{X_1, \dots, X_n\}, E \rangle$?
- ▶ Chain Rule: For any ordering X_1, \dots, X_n , we have:

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

Choose X_1, \dots, X_n consistent with \mathcal{B} : $X_j \in \text{Parents}(X_i) \rightsquigarrow j < i$.

Recovering the Full Joint Probability Distribution

- ▶ Intuition: A Bayesian network is a methodology for representing the full joint probability distribution.
- ▶ Problem: How to recover the full joint probability distribution $P(X_1, \dots, X_n)$ from $\mathcal{B} := \langle \{X_1, \dots, X_n\}, E \rangle$?
- ▶ Chain Rule: For any ordering X_1, \dots, X_n , we have:

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

Choose X_1, \dots, X_n consistent with \mathcal{B} : $X_j \in \text{Parents}(X_i) \rightsquigarrow j < i$.

- ▶ ***Observation 3.4 (Exploiting Conditional Independence).***

With 3.3 (A), we can use $P(X_i | \text{Parents}(X_i))$ instead of $P(X_i | X_{i-1}, \dots, X_1)$:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | \text{Parents}(X_i))$$

The distributions $P(X_i | \text{Parents}(X_i))$ are given by 3.3 (B).

- ▶ Same for atomic events $P(x_1, \dots, x_n)$.

Recovering the Full Joint Probability Distribution

- ▶ Intuition: A Bayesian network is a methodology for representing the full joint probability distribution.
- ▶ Problem: How to recover the full joint probability distribution $P(X_1, \dots, X_n)$ from $\mathcal{B} := \langle \{X_1, \dots, X_n\}, E \rangle$?
- ▶ Chain Rule: For any ordering X_1, \dots, X_n , we have:

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

Choose X_1, \dots, X_n consistent with \mathcal{B} : $X_j \in \text{Parents}(X_i) \rightsquigarrow j < i$.

- ▶ **Observation 3.4 (Exploiting Conditional Independence).**

With 3.3 (A), we can use $P(X_i | \text{Parents}(X_i))$ instead of $P(X_i | X_{i-1}, \dots, X_1)$:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | \text{Parents}(X_i))$$

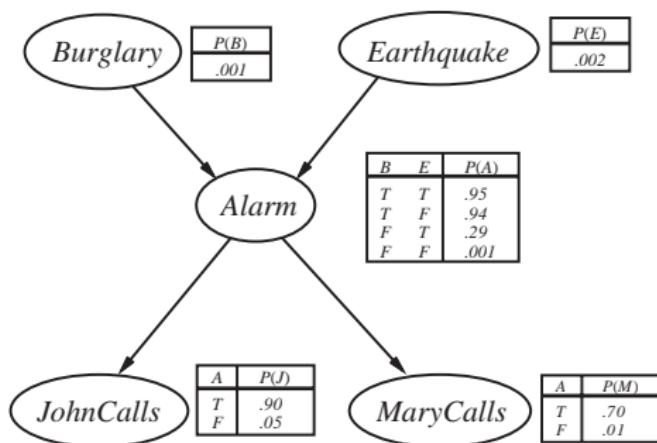
The distributions $P(X_i | \text{Parents}(X_i))$ are given by 3.3 (B).

- ▶ Same for atomic events $P(x_1, \dots, x_n)$.
- ▶ **Observation 3.5 (Why “acyclic”?).** for cyclic \mathcal{B} , this does NOT hold, indeed cyclic BNs may be self-contradictory. (need a consistent ordering)

Recovering a Probability for John, Mary, and the Alarm

- **Example 3.6.** John and Mary called because there was an alarm, but no earthquake or burglary

$$\begin{aligned} P(j, m, a, \neg b, (\neg e)) &= P(j|a) \cdot P(m|a) \cdot P(a|\neg b, \neg e) \cdot P(\neg b) \cdot P(\neg e) \\ &= 0.9 * 0.7 * 0.001 * 0.999 * 0.998 \\ &= 0.00062 \end{aligned}$$



4 Constructing Bayesian Networks

Constructing Bayesian Networks

► BN construction algorithm:

1. Initialize $BN := \langle \{X_1, \dots, X_n\}, E \rangle$ where $E = \emptyset$.
2. Fix any order of the variables, X_1, \dots, X_n .
3. **for** $i := 1, \dots, n$ **do**
 - a. Choose a minimal set $\text{Parents}(X_i) \subseteq \{X_1, \dots, X_{i-1}\}$ so that
$$P(X_i | X_{i-1}, \dots, X_1) = P(X_i | \text{Parents}(X_i))$$

- b. For each $X_j \in \text{Parents}(X_i)$, insert (X_j, X_i) into E .
- c. Associate X_i with $CPT(X_i)$ corresponding to $P(X_i | \text{Parents}(X_i))$.

► **Attention:** Which variables we need to include into $\text{Parents}(X_i)$ depends on what " $\{X_1, \dots, X_{i-1}\}$ " is . . . !

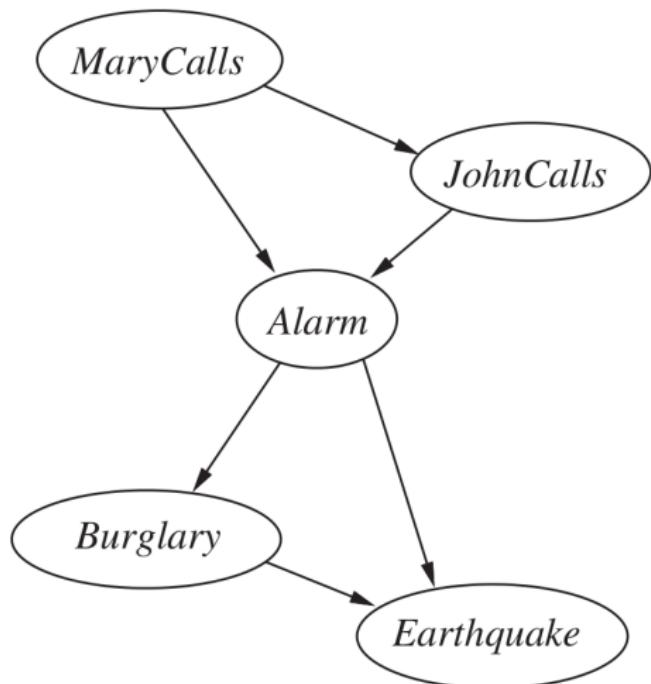
- The size of the resulting BN depends on the chosen order X_1, \dots, X_n .
- The size of a Bayesian network is *not* a fixed property of the domain. It depends on the skill of the designer.

John and Mary Depend on the Variable Order!

- ▶ **Example 4.1.** MaryCalls, JohnCalls, Alarm, Burglary, Earthquake.

John and Mary Depend on the Variable Order!

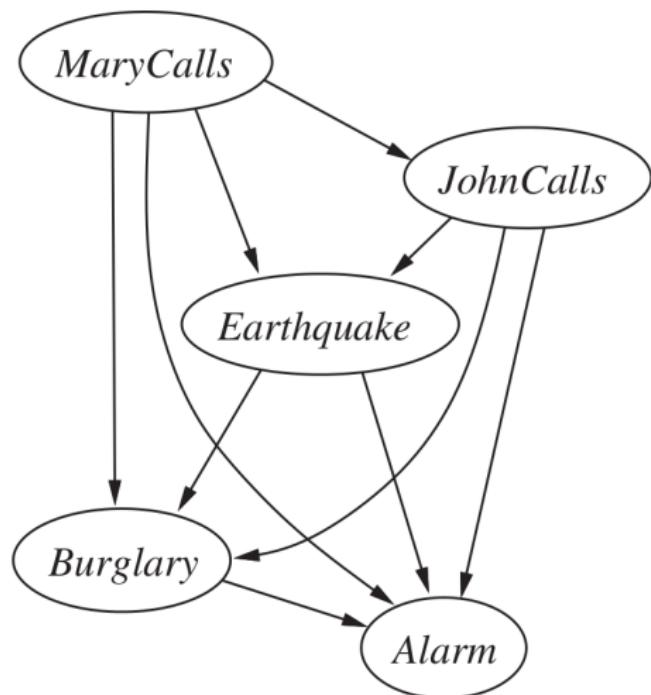
- ▶ **Example 4.1.** MaryCalls, JohnCalls, Alarm, Burglary, Earthquake.



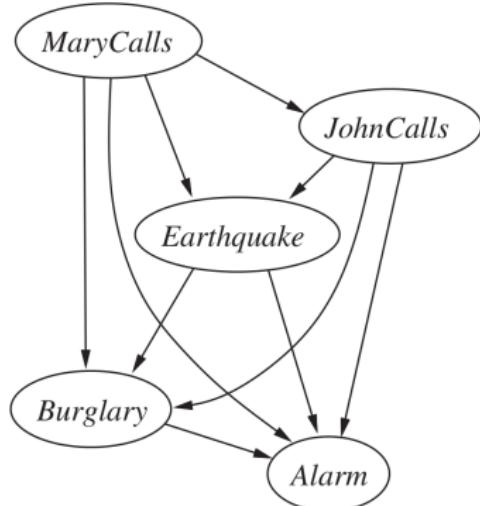
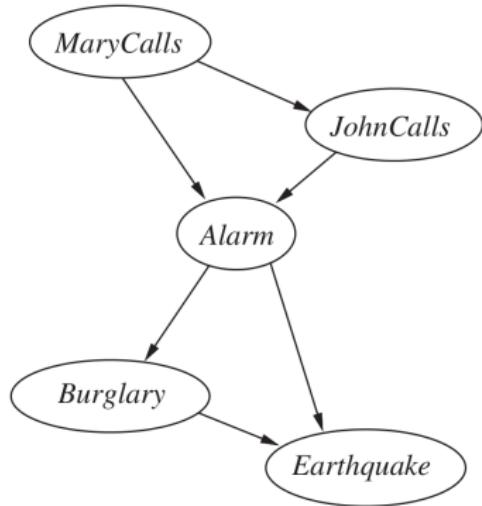
- ▶ **Example 4.2.** MaryCalls, JohnCalls, Earthquake, Burglary, Alarm.

John and Mary Depend on the Variable Order! Ctd.

- Example 4.2. MaryCalls, JohnCalls, Earthquake, Burglary, Alarm.

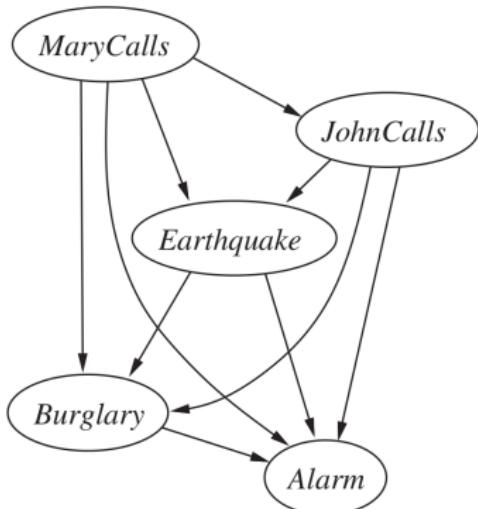
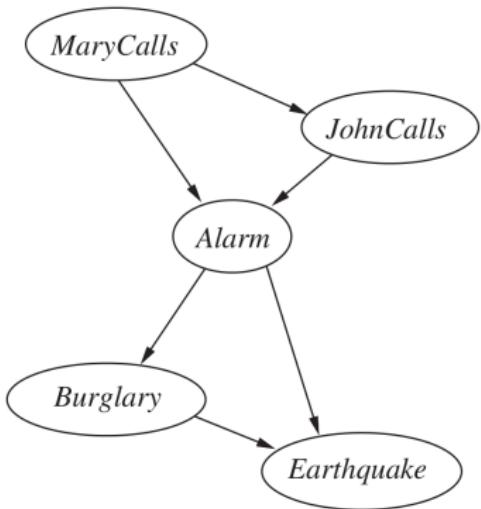


John and Mary, What Went Wrong?



- ▶ Intuition: These BNs link from symptoms to causes! $(P(\text{Cavity}|\text{Toothache}))$
Even though M and J are conditionally independent given A , they are *not* independent without any additional evidence; thus we don't "see" their conditional independence unless we ordered A before M and J ! \leadsto We organized the domain in the wrong way here.
We fail to identify many conditional independence relations (e.g., get dependencies between conditionally independent symptoms).

John and Mary, What Went Wrong?



- ▶ Intuition: These BNs link from symptoms to causes! ($P(\text{Cavity}|\text{Toothache})$)
- ▶ Also recall: Conditional probabilities $P(\text{Symptom}|\text{Cause})$ are more robust and often easier to assess than $P(\text{Cause}|\text{Symptom})$.
- ▶ Rule of Thumb: We should order causes before symptoms.

Compactness of Bayesian Networks

- ▶ **Definition 4.3.** Given random variables X_1, \dots, X_n with finite domains D_1, \dots, D_n , the size of $\mathcal{B} := \langle \{X_1, \dots, X_n\}, E \rangle$ is defined as

$$\text{size}(\mathcal{B}) := \sum_{i=1}^n \#(D_i) \cdot \prod_{X_j \in \text{Parents}(X_i)} \#(D_j)$$

- ▶ = The total number of entries in the CPTs.
- ▶ Smaller BN \leadsto need to assess less probabilities, more efficient inference.
- ▶ Explicit full joint probability distribution has size $\prod_{i=1}^n \#(D_i)$.
- ▶ If $\#(\text{Parents}(X_i)) \leq k$ for every X_i , and D_{\max} is the largest variable domain, then $\text{size}(\mathcal{B}) \leq n \#(D_{\max})^{k+1}$.
- ▶ For $\#(D_{\max}) = 2$, $n = 20$, $k = 4$ we have $2^{20} = 1048576$ probabilities, but a Bayesian network of size $\leq 20 \cdot 2^5 = 640 \dots !$
- ▶ In the worst case, $\text{size}(\mathcal{B}) = n \cdot \prod_{i=1}^n \#(D_i)$, namely if every variable depends on all its predecessors in the chosen order.
- ▶ BNs are compact if each variable is directly influenced only by few of its predecessor variables.

Representing Conditional Distributions: Deterministic Nodes

- ▶ Problem: Even if $\max(\text{Parents})$ is small, the CPT has 2^k entries. (worst-case)
- ▶ only need to determine pattern and some values.

Representing Conditional Distributions: Deterministic Nodes

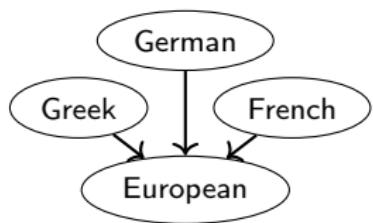
- ▶ **Problem:** Even if $\max(\text{Parents})$ is small, the **CPT** has 2^k entries. (worst-case)
- ▶ **Idea:** Usually **CPTs** follow standard patterns called **canonical distributions**.
- ▶ only need to determine pattern and some values.
- ▶ **Definition 4.4.** A node X in a **Bayesian network** is called **deterministic**, if its value is completely determined by the values of $\text{Parents}(X)$.

Representing Conditional Distributions: Deterministic Nodes

- ▶ Problem: Even if $\max(\text{Parents})$ is small, the CPT has 2^k entries. (worst-case)
- ▶ Idea: Usually CPTs follow standard patterns called **canonical distributions**.
- ▶ only need to determine pattern and some values.
- ▶ **Definition 4.4.** A node X in a Bayesian network is called **deterministic**, if its value is completely determined by the values of $\text{Parents}(X)$.
- ▶ **Example 4.5 (Logical Dependencies).**

In the network on the right, the node *European* is **deterministic**, the CPT corresponds to a logical disjunction, i.e.

$$P(\text{european}) = P(\text{greek} \vee \text{german} \vee \text{french}).$$



Representing Conditional Distributions: Deterministic Nodes

- ▶ Problem: Even if $\max(\text{Parents})$ is small, the CPT has 2^k entries. (worst-case)
- ▶ Idea: Usually CPTs follow standard patterns called **canonical distributions**.
- ▶ only need to determine pattern and some values.
- ▶ **Definition 4.4.** A node X in a Bayesian network is called **deterministic**, if its value is completely determined by the values of $\text{Parents}(X)$.
- ▶ **Example 4.5 (Logical Dependencies).**

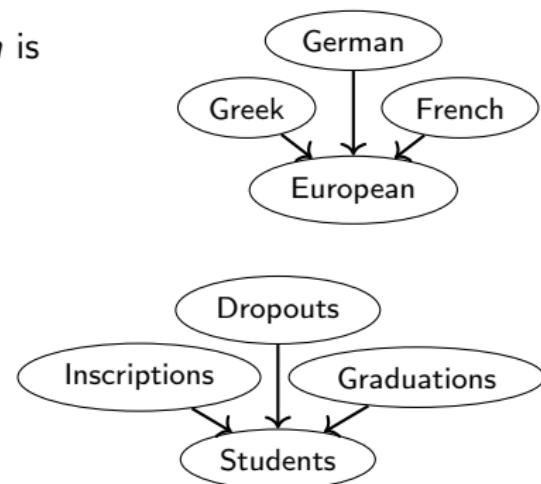
In the network on the right, the node *European* is **deterministic**, the CPT corresponds to a logical disjunction, i.e.

$$P(\text{european}) = P(\text{greek} \vee \text{german} \vee \text{french}).$$

- ▶ **Example 4.6 (Numerical Dependencies).**

In the network on the right, the node *Students* is **deterministic**, the CPT corresponds to a sum, i.e.

$$P(S = i - d - g) = P(I = i) + P(D = d) + P(G = g).$$

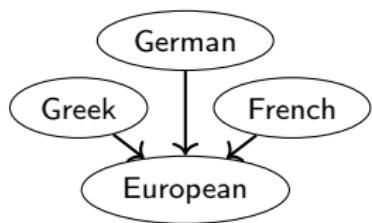


Representing Conditional Distributions: Deterministic Nodes

- ▶ Problem: Even if $\max(\text{Parents})$ is small, the CPT has 2^k entries. (worst-case)
- ▶ Idea: Usually CPTs follow standard patterns called **canonical distributions**.
- ▶ only need to determine pattern and some values.
- ▶ **Definition 4.4.** A node X in a Bayesian network is called **deterministic**, if its value is completely determined by the values of $\text{Parents}(X)$.
- ▶ **Example 4.5 (Logical Dependencies).**

In the network on the right, the node *European* is **deterministic**, the CPT corresponds to a logical disjunction, i.e.

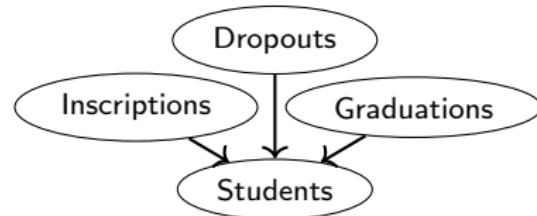
$$P(\text{european}) = P(\text{greek} \vee \text{german} \vee \text{french}).$$



- ▶ **Example 4.6 (Numerical Dependencies).**

In the network on the right, the node *Students* is **deterministic**, the CPT corresponds to a sum, i.e.

$$P(S = i - d - g) = P(I = i) + P(D = d) + P(G = g).$$



- ▶ Intuition: Deterministic nodes model direct, causal relationships.

Representing Conditional Distributions: Noisy Nodes

- **Problem:** Sometimes, values of nodes are only “almost deterministic”.
(uncertain, but mostly logical)

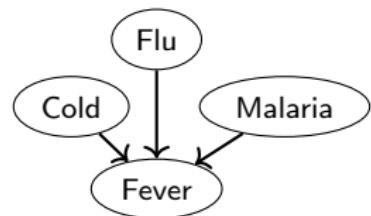
Representing Conditional Distributions: Noisy Nodes

- ▶ **Problem:** Sometimes, values of nodes are only “almost deterministic”.
(uncertain, but mostly logical)
- ▶ **Idea:** Use “noisy” logical relationships. *(generalize logical ones softly to [0,1])*

Representing Conditional Distributions: Noisy Nodes

- ▶ **Problem:** Sometimes, values of nodes are only “almost deterministic”.
(uncertain, but mostly logical)
- ▶ **Idea:** Use “noisy” logical relationships. *(generalize logical ones softly to [0,1])*
- ▶ **Example 4.7 (Inhibited Causal Dependencies).**

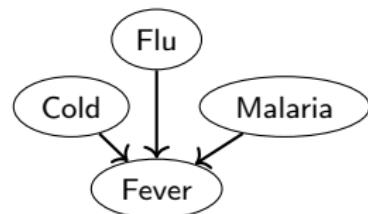
In the network on the right, deterministic disjunction for the node Fever is incorrect, since the diseases sometimes fail to develop fever. The causal relation between parent and child is **inhibited**.



Representing Conditional Distributions: Noisy Nodes

- ▶ **Problem:** Sometimes, values of nodes are only “almost deterministic”.
(uncertain, but mostly logical)
- ▶ **Idea:** Use “noisy” logical relationships. *(generalize logical ones softly to [0,1])*
- ▶ **Example 4.7 (Inhibited Causal Dependencies).**

In the network on the right, deterministic disjunction for the node Fever is incorrect, since the diseases sometimes fail to develop fever. The causal relation between parent and child is **inhibited**.



- ▶ **Assumptions:** We make the following assumptions for modeling 4.7:
 1. Cold, Flu, and Malaria is a complete list of fever causes *(add a leak node for the others otherwise)*.
 2. Inhibitions of the **parents** are independent

Thus we can model the inhibitions by individual inhibition factors q_j .

- ▶ **Definition 4.8.** The **CPT** of a **noisy disjunction node** X in a **Bayesian network** is given by $P(x_i | \text{Parents}(X_i)) = \prod_{\{j | X_j=T\}} q_j$, where the q_j are the inhibition factors of $X_i \in \text{Parents}(X)$.

Representing Conditional Distributions: Noisy Nodes

- **Example 4.9.** We have the following inhibition factors for 4.7:

$$q_{\text{cold}} = P(\neg \text{fever} | \text{cold}, \neg \text{flu}, \neg \text{malaria}) = 0.6$$

$$q_{\text{flu}} = P(\neg \text{fever} | \neg \text{cold}, \text{flu}, \neg \text{malaria}) = 0.2$$

$$q_{\text{malaria}} = P(\neg \text{fever} | \neg \text{cold}, \neg \text{flu}, \text{malaria}) = 0.1$$

If we model Fever as a **noisy disjunction node**, then the general rule $P(x_i | \text{Parents}(X_i)) = \prod_{\{j | X_j = T\}} q_j$ for the **CPT** gives the following table:

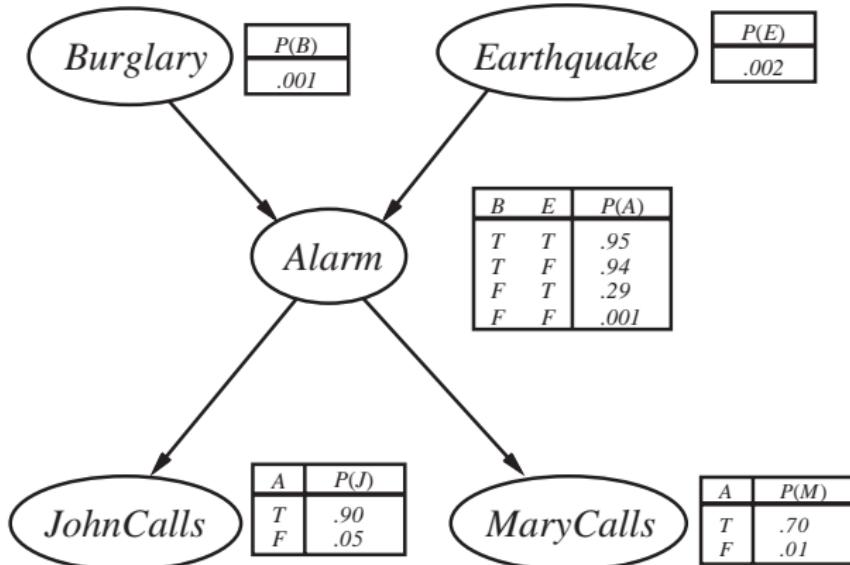
Cold	Flu	Malaria	$P(\text{Fever})$	$P(\neg \text{Fever})$
F	F	F	0.0	1.0
F	F	T	0.9	0.1
F	T	F	0.8	0.2
F	T	T	0.98	$0.02 = 0.2 \cdot 0.1$
T	F	F	0.4	0.6
T	F	T	0.94	$0.06 = 0.6 \cdot 0.1$
T	T	F	0.88	$0.12 = 0.6 \cdot 0.2$
T	T	T	0.988	$0.012 = 0.6 \cdot 0.2 \cdot 0.1$

- ▶ **Observation 4.10.** In general, noisy logical relationships in which a variable depends on k parents can be described by $\mathcal{O}(k)$ parameters instead of $\mathcal{O}(2^k)$ for the full conditional probability table. This can make assessment (and learning) tractable.
- ▶ **Example 4.11.** The CPCS network [Pra+94] uses noisy-OR and noisy-MAX distributions to model relationships among diseases and symptoms in internal medicine. With 448 nodes and 906 links, it requires only 8,254 values instead of 133,931,430 for a network with full CPTs.

5 Inference in Bayesian Networks

Inference for Mary and John

- ▶ Intuition: Observe **evidence variables** and draw conclusions on **query variables**.
- ▶ Example 5.1.



- ▶ What is $P(\text{Burglary}|\text{johncalls})$?
- ▶ What is $P(\text{Burglary}|\text{johncalls}, \text{marycalls})$?

- ▶ **Definition 5.2 (Probabilistic Inference Task).** Given random variables X_1, \dots, X_n , a probabilistic inference task consists of a set $X \subseteq \{X_1, \dots, X_n\}$ of query variables, a set $E \subseteq \{X_1, \dots, X_n\}$ of evidence variables, and an event e that assigns values to E . We wish to compute the conditional probability distribution $P(X|e)$.
 $Y := \{X_1, \dots, X_n\} \setminus X \cup E$ are the hidden variables.
- ▶ Notes:
 - ▶ We assume that a \mathcal{B} for X_1, \dots, X_n is given.
 - ▶ In the remainder, for simplicity, $X = \{X\}$ is a singleton.
- ▶ **Example 5.3.** In $P(\text{Burglary}|\text{johncalls}, \text{marycalls})$, $X = \text{Burglary}$, $e = \text{johncalls}, \text{marycalls}$, and $Y = \{\text{Alarm}, \text{EarthQuake}\}$.

Inference by Enumeration: The Principle (A Reminder!)

- ▶ **Problem:** Given evidence e , want to know $P(X|e)$.
Hidden variables: Y .

Inference by Enumeration: The Principle (A Reminder!)

- ▶ **Problem:** Given evidence e , want to know $P(X|e)$.
Hidden variables: Y .
- ▶ **1. Bayesian network:** Construct a **Bayesian network \mathcal{B}** that captures variable dependencies.

Inference by Enumeration: The Principle (A Reminder!)

- ▶ **Problem:** Given evidence e , want to know $P(X|e)$.
Hidden variables: Y .
- ▶ 1. **Bayesian network:** Construct a **Bayesian network \mathcal{B}** that captures variable dependencies.
- ▶ 2. **Normalization+Marginalization:**
$$P(X|e) = \alpha P(X, e); \text{ if } Y \neq \emptyset \text{ then } P(X|e) = \alpha(\sum_{y \in Y} P(X, e, y))$$

Inference by Enumeration: The Principle (A Reminder!)

- ▶ **Problem:** Given evidence e , want to know $P(X|e)$.
Hidden variables: Y .
- ▶ 1. **Bayesian network:** Construct a Bayesian network \mathcal{B} that captures variable dependencies.
- ▶ 2. **Normalization+Marginalization:**
$$P(X|e) = \alpha P(X, e); \text{ if } Y \neq \emptyset \text{ then } P(X|e) = \alpha(\sum_{y \in Y} P(X, e, y))$$
- ▶ Recover the summed-up probabilities $P(X, e, y)$ from \mathcal{B} !
- ▶ 3. **Chain Rule:** Order X_1, \dots, X_n consistent with \mathcal{B} .

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

Inference by Enumeration: The Principle (A Reminder!)

- ▶ **Problem:** Given evidence e , want to know $P(X|e)$.
Hidden variables: Y .
- ▶ 1. **Bayesian network:** Construct a Bayesian network \mathcal{B} that captures variable dependencies.
- ▶ 2. **Normalization+Marginalization:**
$$P(X|e) = \alpha P(X, e); \text{ if } Y \neq \emptyset \text{ then } P(X|e) = \alpha(\sum_{y \in Y} P(X, e, y))$$
- ▶ Recover the summed-up probabilities $P(X, e, y)$ from \mathcal{B} !
- ▶ 3. **Chain Rule:** Order X_1, \dots, X_n consistent with \mathcal{B} .
$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$
- ▶ 4. **Exploit conditional independence:** Instead of $P(X_i | X_{i-1}, \dots, X_1)$, use $P(X_i | \text{Parents}(X_i))$.

Inference by Enumeration: The Principle (A Reminder!)

- ▶ **Problem:** Given evidence e , want to know $P(X|e)$.
Hidden variables: Y .
- ▶ 1. **Bayesian network:** Construct a Bayesian network \mathcal{B} that captures variable dependencies.
- ▶ 2. **Normalization+Marginalization:**

$$P(X|e) = \alpha P(X, e); \text{ if } Y \neq \emptyset \text{ then } P(X|e) = \alpha(\sum_{y \in Y} P(X, e, y))$$

- ▶ Recover the summed-up probabilities $P(X, e, y)$ from \mathcal{B} !
- ▶ 3. **Chain Rule:** Order X_1, \dots, X_n consistent with \mathcal{B} .

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

- ▶ 4. **Exploit conditional independence:** Instead of $P(X_i | X_{i-1}, \dots, X_1)$, use $P(X_i | \text{Parents}(X_i))$.
- ▶ Given a Bayesian network \mathcal{B} , probabilistic inference tasks can be solved as sums of products of conditional probabilities from \mathcal{B} .

Inference by Enumeration: The Principle (A Reminder!)

- ▶ **Problem:** Given evidence e , want to know $P(X|e)$.
Hidden variables: Y .
- ▶ 1. **Bayesian network:** Construct a Bayesian network \mathcal{B} that captures variable dependencies.
- ▶ 2. **Normalization+Marginalization:**

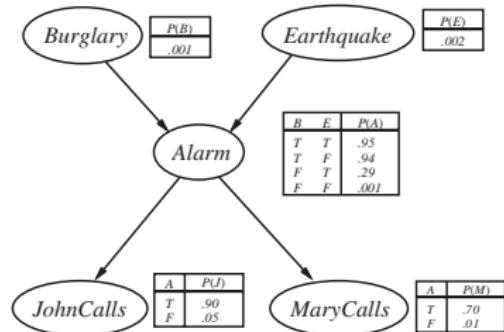
$$P(X|e) = \alpha P(X, e); \text{ if } Y \neq \emptyset \text{ then } P(X|e) = \alpha(\sum_{y \in Y} P(X, e, y))$$

- ▶ Recover the summed-up probabilities $P(X, e, y)$ from \mathcal{B} !
- ▶ 3. **Chain Rule:** Order X_1, \dots, X_n consistent with \mathcal{B} .

$$P(X_1, \dots, X_n) = P(X_n | X_{n-1}, \dots, X_1) \cdot P(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot P(X_1)$$

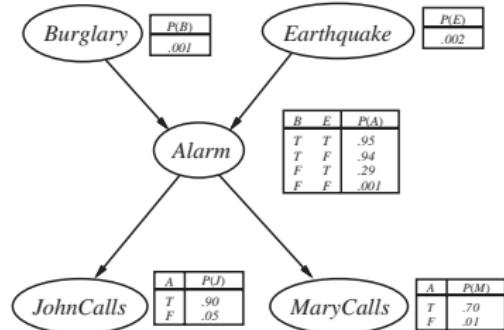
- ▶ 4. **Exploit conditional independence:** Instead of $P(X_i | X_{i-1}, \dots, X_1)$, use $P(X_i | \text{Parents}(X_i))$.
- ▶ Given a Bayesian network \mathcal{B} , probabilistic inference tasks can be solved as sums of products of conditional probabilities from \mathcal{B} .
- ▶ Sum over all value combinations of hidden variables.

Inference by Enumeration: John and Mary



- Want: $P(\text{Burglary}|\text{johncalls}, \text{marycalls})$.
Hidden variables: $Y = \{\text{Earthquake}, \text{Alarm}\}$.

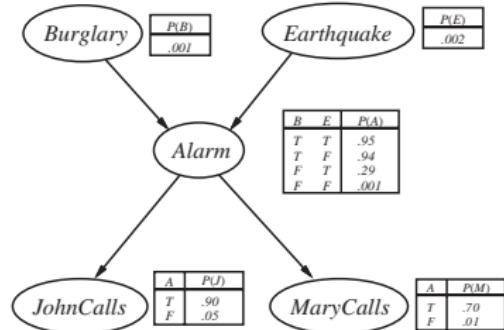
Inference by Enumeration: John and Mary



- Want: $P(\text{Burglary} | \text{johncalls}, \text{marycalls})$.
Hidden variables: $Y = \{\text{Earthquake}, \text{Alarm}\}$.
- Normalization+Marginalization:

$$P(B|j, m) = \alpha P(B, j, m) = \alpha \left(\sum_{v_E} \sum_{v_A} P(B, j, m, v_E, v_A) \right)$$

Inference by Enumeration: John and Mary



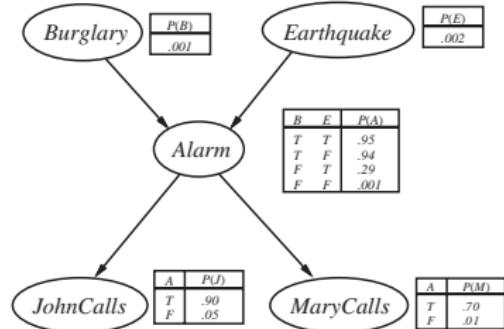
- Want: $P(\text{Burglary} | \text{johncalls}, \text{marycalls})$.
Hidden variables: $Y = \{\text{Earthquake}, \text{Alarm}\}$.

- Normalization+Marginalization:

$$P(B|j, m) = \alpha P(B, j, m) = \alpha \left(\sum_{v_E} \sum_{v_A} P(B, j, m, v_E, v_A) \right)$$

- Order: $X_1 = B, X_2 = E, X_3 = A, X_4 = J, X_5 = M$.

Inference by Enumeration: John and Mary



- Want: $P(\text{Burglary} | \text{johncalls}, \text{marycalls})$.
Hidden variables: $Y = \{\text{Earthquake}, \text{Alarm}\}$.

- Normalization+Marginalization:

$$P(B|j, m) = \alpha P(B, j, m) = \alpha \left(\sum_{v_E} \sum_{v_A} P(B, j, m, v_E, v_A) \right)$$

- Order: $X_1 = B, X_2 = E, X_3 = A, X_4 = J, X_5 = M$.

- Chain rule and conditional independence:

$$P(B|j, m) = \alpha \left(\sum_{v_E} \sum_{v_A} P(B) \cdot P(v_E) \cdot P(v_A|B, v_E) \cdot P(j|v_A) \cdot P(m|v_A) \right)$$

Inference by Enumeration: John and Mary, ctd.

- Move variables outwards: (until we hit the first parent):

$$P(B|j, m) = \alpha \cdot P(B) \cdot \left(\sum_{v_E} P(v_E) \cdot \left(\sum_{v_A} P(v_A|B, v_E) \cdot P(j|v_A) \cdot P(m|v_A) \right) \right)$$

Note: This step *is* actually done by the pseudo-code, implicitly in the sense that in the recursive calls to enumerate-all we multiply our own prob with all the rest. That is valid because, the variable ordering being consistent, all our **parents** are already here which is just another way of saying “my own prob does not depend on the variables in the rest of the order”.

- The probabilities of the outside-variables multiply the entire “rest of the sum”

Inference by Enumeration: John and Mary, ctd.

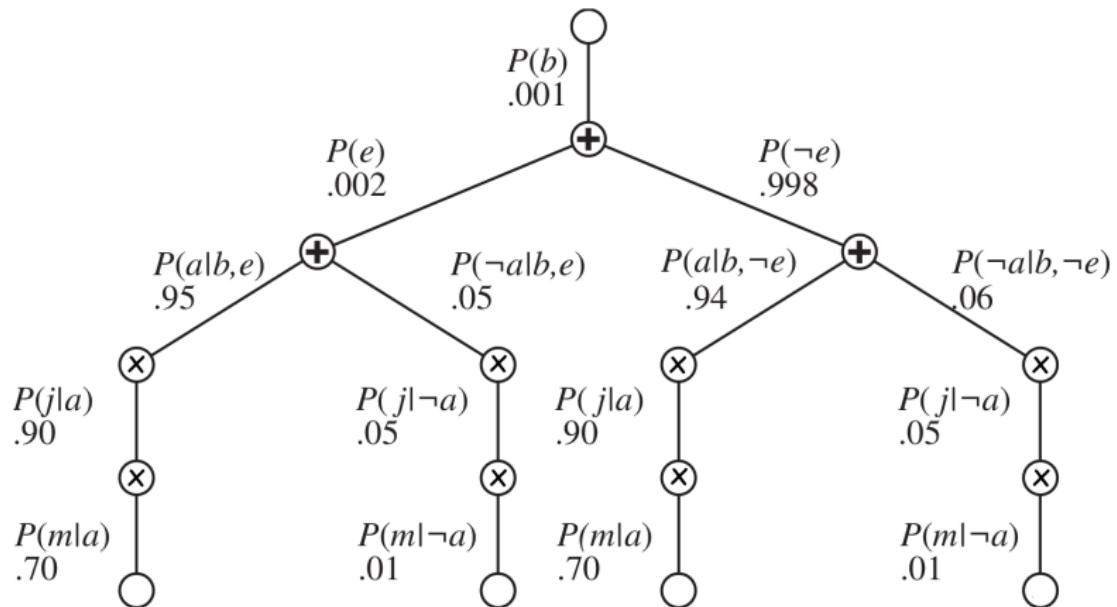
- Move variables outwards: (until we hit the first parent):

$$P(B|j, m) = \alpha \cdot P(B) \cdot (\sum_{v_E} P(v_E) \cdot (\sum_{v_A} P(v_A|B, v_E) \cdot P(j|v_A) \cdot P(m|v_A)))$$

- The probabilities of the outside-variables multiply the entire “rest of the sum”
- Chain rule and conditional independence, ctd.:

$$\begin{aligned} & P(B|j, m) \\ &= \alpha P(B) (\sum_{v_E} P(v_E) (\sum_{v_A} P(v_A|B, v_E) P(j|v_A) P(m|v_A))) \\ &= \alpha \cdot P(b) \cdot \left(P(e) \cdot \left(+ \underbrace{\frac{P(a|b, e) P(j|a) P(m|a)}{P(\neg a|b, e) P(j|\neg a) P(m|\neg a)}}_e \right) e \right. \\ &\quad \left. + P(\neg e) \cdot \left(+ \underbrace{\frac{P(a|b, \neg e) P(j|a) P(m|a)}{P(\neg a|b, \neg e) P(j|\neg a) P(m|\neg a)}}_{\neg a} \right) \neg e \right) \\ &= \alpha \langle 0.00059224, 0.0014919 \rangle \approx \langle 0.284, 0.716 \rangle \end{aligned}$$

The Evaluation of $P(b|j, m)$, as a “Search Tree”



- Inference by enumeration = a tree with “sum nodes” branching over values of hidden variables, and with non-branching “multiplication nodes”.

- ▶ Inference by Enumeration:
 - ▶ Evaluates the tree in a depth-first manner.

- ▶ Inference by Enumeration:
 - ▶ Evaluates the tree in a depth-first manner.
 - ▶ space complexity: linear in the number of variables.
 - ▶ time complexity: exponential in the number of hidden variables, e.g. $\mathcal{O}(2^{\#(Y)})$ in case these variables are Boolean.
- ▶ Can we do better than this?
- ▶ **Definition 5.4.** Variable elimination is a BNI algorithm that avoids
 - ▶ repeated computation, and (see below)
 - ▶ irrelevant computation. (see below)
- ▶ In some special cases, variable elimination runs in polynomial time.

Variable Elimination: Sketch of Ideas

- ▶ **Avoiding repeated computation:** Evaluate expressions from right to left, storing all intermediate results.
- ▶ For query $P(B|j, m)$:
 1. CPTs of BN yield **factors** (probability tables):

$$P(B|j, m) = \alpha \cdot \underbrace{P(B)}_{f_1(B)} \cdot (\sum_{v_E} \underbrace{P(v_E)}_{f_2(E)}) \sum_{v_A} \underbrace{P(v_A|B, v_E)}_{f_3(A, B, E)} \cdot \underbrace{P(j|v_A)}_{f_4(A)} \cdot \underbrace{P(m|v_A)}_{f_5(A)}$$

2. Then the computation is performed in terms of **factor product** and **summing out variables** from factors:

$$P(B|j, m) = \alpha \cdot f_1(B) \cdot (\sum_{v_E} f_2(E) \cdot (\sum_{v_A} f_3(A, B, E) \cdot f_4(A) \cdot f_5(A)))$$

Variable Elimination: Sketch of Ideas

- ▶ **Avoiding repeated computation:** Evaluate expressions from right to left, storing all intermediate results.
- ▶ For query $P(B|j, m)$:
 1. CPTs of BN yield **factors** (probability tables):

$$P(B|j, m) = \alpha \cdot \underbrace{P(B)}_{f_1(B)} \cdot (\sum_{v_E} \underbrace{P(v_E)}_{f_2(E)}) \sum_{v_A} \underbrace{P(v_A|B, v_E)}_{f_3(A, B, E)} \cdot \underbrace{P(j|v_A)}_{f_4(A)} \cdot \underbrace{P(m|v_A)}_{f_5(A)}$$

2. Then the computation is performed in terms of **factor product** and **summing out variables** from factors:

$$P(B|j, m) = \alpha \cdot f_1(B) \cdot (\sum_{v_E} f_2(E) \cdot (\sum_{v_A} f_3(A, B, E) \cdot f_4(A) \cdot f_5(A)))$$

- ▶ **Avoiding irrelevant computation:** Repeatedly remove hidden variables that are leaf nodes.
- ▶ For query $P(\text{JohnCalls}|\text{burglary})$:

$$P(J|b) = \alpha \cdot P(b) \cdot (\sum_{v_E} P(v_E) \cdot (\sum_{v_A} P(v_A|b, v_E) \cdot P(J|v_A) \cdot (\sum_{v_M} P(v_M|v_A))))$$

- ▶ The rightmost sum equals 1 and can be dropped.

- ▶ **Definition 5.5.** A graph G is called **singly connected**, or a **polytree** (otherwise **multiply connected**), if there is at most one **undirected path** between any two **nodes** in G .
- ▶ **Theorem 5.6 (Good News).** On **singly connected Bayesian networks**, variable elimination runs in **polynomial time**.

- ▶ **Definition 5.5.** A graph G is called **singly connected**, or a **polytree** (otherwise **multiply connected**), if there is at most one **undirected path** between any two **nodes** in G .
- ▶ **Theorem 5.6 (Good News).** On **singly connected Bayesian networks**, variable elimination runs in **polynomial time**.
- ▶ Is our BN for Mary & John a **polytree**? (Yes.)

- ▶ **Definition 5.5.** A graph G is called **singly connected**, or a **polytree** (otherwise **multiply connected**), if there is at most one **undirected path** between any two **nodes** in G .
- ▶ **Theorem 5.6 (Good News).** On **singly connected Bayesian networks**, variable elimination runs in **polynomial time**.
- ▶ Is our BN for Mary & John a **polytree**? (Yes.)
- ▶ **Theorem 5.7 (Bad News).** For **multiply connected Bayesian networks**, probabilistic inference is **#P-hard**. (#P is harder than NP, i.e. $NP \subseteq \#P$)
- ▶ **So?:** Life goes on ... In the hard cases, if need be we can throw exactitude to the winds and approximate.
- ▶ **Example 5.8.** Sampling techniques as in **MCTS**.

6 Conclusion

- ▶ Bayesian networks (BN) are a wide-spread tool to model uncertainty, and to reason about it. A BN represents conditional independence relations between random variables. It consists of a graph encoding the variable dependencies, and of conditional probability tables (CPTs).
- ▶ Given a variable order, the BN is small if every variable depends on only a few of its predecessors.
- ▶ Probabilistic inference requires to compute the probability distribution of a set of query variables, given a set of evidence variables whose values we know. The remaining variables are hidden.
- ▶ Inference by enumeration takes a BN as input, then applies Normalization+Marginalization, the chain rule, and exploits conditional independence. This can be viewed as a tree search that branches over all values of the hidden variables.
- ▶ Variable elimination avoids unnecessary computation. It runs in polynomial time for poly-tree BNs. In general, exact probabilistic inference is #P-hard. Approximate probabilistic inference methods exist.

Topics We Didn't Cover Here

- ▶ **Inference by sampling:** A whole zoo of methods for doing this exists.

Topics We Didn't Cover Here

- ▶ **Inference by sampling:** A whole zoo of methods for doing this exists.
- ▶ **Clustering:** Pre-combining subsets of variables to reduce the runtime of inference.

Topics We Didn't Cover Here

- ▶ **Inference by sampling:** A whole zoo of methods for doing this exists.
- ▶ **Clustering:** Pre-combining subsets of variables to reduce the runtime of inference.
- ▶ **Compilation to SAT:** More precisely, to “weighted model counting” in CNF formulas. Model counting extends DPLL with the ability to determine the number of satisfying interpretations. Weighted model counting allows to define a mass for each such interpretation (= the probability of an **atomic event**).

Topics We Didn't Cover Here

- ▶ **Inference by sampling:** A whole zoo of methods for doing this exists.
- ▶ **Clustering:** Pre-combining subsets of variables to reduce the runtime of inference.
- ▶ **Compilation to SAT:** More precisely, to “weighted model counting” in CNF formulas. Model counting extends DPLL with the ability to determine the number of satisfying interpretations. Weighted model counting allows to define a mass for each such interpretation (= the probability of an **atomic event**).
- ▶ **Dynamic BN:** BN with one slice of variables at each “time step”, encoding probabilistic behavior over time.

Topics We Didn't Cover Here

- ▶ **Inference by sampling:** A whole zoo of methods for doing this exists.
- ▶ **Clustering:** Pre-combining subsets of variables to reduce the runtime of inference.
- ▶ **Compilation to SAT:** More precisely, to “weighted model counting” in CNF formulas. Model counting extends DPLL with the ability to determine the number of satisfying interpretations. Weighted model counting allows to define a mass for each such interpretation (= the probability of an **atomic event**).
- ▶ **Dynamic BN:** BN with one slice of variables at each “time step”, encoding probabilistic behavior over time.
- ▶ **Relational BN:** BN with predicates and object variables.

Topics We Didn't Cover Here

- ▶ **Inference by sampling:** A whole zoo of methods for doing this exists.
- ▶ **Clustering:** Pre-combining subsets of variables to reduce the runtime of inference.
- ▶ **Compilation to SAT:** More precisely, to “weighted model counting” in CNF formulas. Model counting extends DPLL with the ability to determine the number of satisfying interpretations. Weighted model counting allows to define a mass for each such interpretation (= the probability of an **atomic event**).
- ▶ **Dynamic BN:** BN with one slice of variables at each “time step”, encoding probabilistic behavior over time.
- ▶ **Relational BN:** BN with predicates and object variables.
- ▶ **First-order BN:** Relational BN with quantification, i.e. probabilistic logic. E.g., the BLOG language developed by Stuart Russel and co-workers.

Chapter 22 Making Simple Decisions Rationally

1 Introduction

- ▶ **Definition 1.1.** Decision theory investigates **decision problems**, i.e. how an agent a deals with choosing among **actions** based on the desirability of their outcomes given by a **utility function** on **state**.
- ▶ **Wait:** Isn't that what we did in *Problem Solving*?
- ▶ **Yes, but:** Now we do it for **stochastic** (i.e. non-deterministic), **partially observable environments**.
- ▶ **Recall:** We call the **environment** of an **agent A**
 - ▶ **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**.
 - ▶ **deterministic**, iff the next state of the environment is completely determined by the current state and A 's action, else **stochastic**.
 - ▶ **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**.
- ▶ **For now:** We restrict ourselves to **episodic decision theory**, which deals with choosing among **actions** based on the desirability of their *immediate* outcomes. (**no need to treat time explicitly**)
- ▶ **Later:** We will study **sequential decision problems**, where the **agent**'s utility depends on a sequence of decisions.

Preview: Episodic Decision Theory

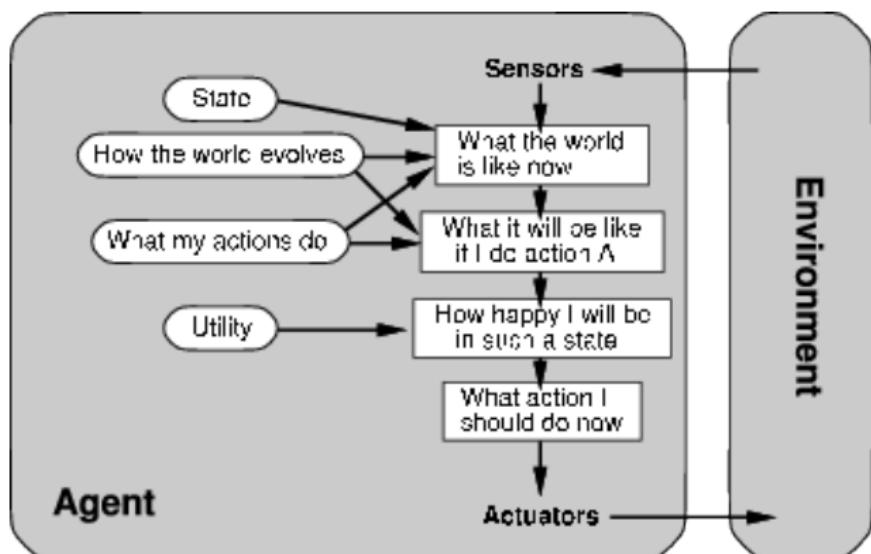
- ▶ **Problem:** The environment is partially observable, so we do not know the “current state”.
- ▶ **Idea: Rational** decisions $\hat{=}$ choose actions that maximize **expected utility(MEU)**:
 - ▶ Treat the result of an action a as a random variable R_a whose values are the possible states.
 - ▶ Study $P(R_a = s' | a, e)$ given evidence observations e .
 - ▶ Capture the agent's preferences in a **utility function** U from states to \mathbb{R}_0^+ .
 - ▶ The **expected utility** $EU(a)$ of an action a is then

$$EU(a) = \sum_{s'} P(R_a = s' | a) \cdot U(s')$$

- ▶ **Intuitively:** A formalization of what it means to “do the right thing”.
- ▶ **Hooray:** This solves all of the AI problem. (in principle)
- ▶ **Problem:** There is a long long way towards an operationalization. (do that now)

Utility-based agents

- ▶ **Definition 1.2.** A **utility based agent** uses a **world model** along with a **utility function** that influences its **preferences** among the **states** of that world. It chooses the **action** that leads to the best **expected utility**, which is computed by averaging over all possible outcome **states**, weighted by the probability of the outcome.
- ▶ **Agent Schema:**



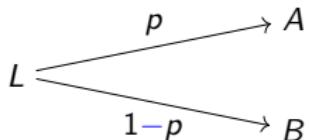
- ▶ Rational preferences
- ▶ Utilities and Money
- ▶ Multiattribute utilities
- ▶ Decision networks
- ▶ Value of information

2 Rational Preferences

Preferences in Deterministic Environments

- ▶ **Problem:** We cannot directly measure **utility** of (or satisfaction/happiness in) a state.
- ▶ **Example 2.1.** I have to decide whether to go to class today (or sleep in).
What is the **utility** of this lecture. (obviously 42)
- ▶ **Idea:** We can let people/agents choose between two **states!** (subjective preference)
- ▶ **Example 2.2.** *Give me your phone or I will give you a bloody nose.* \rightsquigarrow
To make a decision in a **deterministic** environment, the agent must determine whether it prefers a **state** without phone to one with a bloody nose?
- ▶ **Definition 2.3.** Given **states** A and B (we call them **prizes**) and agent can express **preferences** of the form
 - ▶ $A \succ B$ A **preferred** over B
 - ▶ $A \sim B$ **indifference** between A and B
 - ▶ $A \succeq B$ B not **preferred** over A

- ▶ **Problem:** In non-deterministic environments we do not have full information about the states we choose between.
- ▶ **Example 2.4 (Airline Food).** *Do you want chicken or pasta* (but we cannot see through the tin foil)
- ▶ **Definition 2.5.**
Given prizes A_i and probabilities p_i with $\sum_{i=1}^n p_i = 1$, a **lottery** $[p_1, A_1; \dots; p_n, A_n]$ represents the result of a non-deterministic action that can have outcomes A_i with probability p_i . For the binary case, we use $[p, A; (1-p), B]$.
- ▶ We extend **preferences** to include **lotteries** for non-deterministic environments.



- ▶ Idea: Preferences of a rational agent must obey constraints:
Rational preferences \rightsquigarrow behavior describable as maximization of expected utility.
- ▶ **Definition 2.6.** We call a set \succ of preferences rational, iff the following constraints hold:

Orderability

$$A \succ B \vee B \succ A \vee A \sim B$$

Transitivity

$$A \succ B \wedge B \succ C \Rightarrow A \succ C$$

Continuity

$$A \succ B \succ C \Rightarrow (\exists p. [p, A; (1-p), C] \sim B)$$

Substitutability

$$A \sim B \Rightarrow [p, A; (1-p), C] \sim [p, B; (1-p), C]$$

Monotonicity

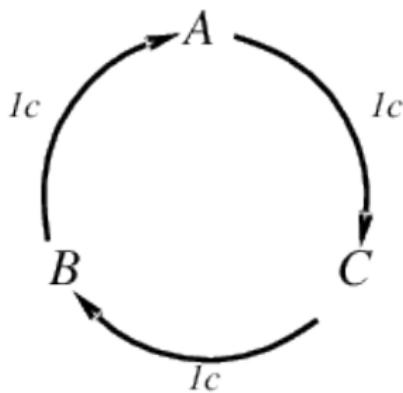
$$A \succ B \Rightarrow (p > q) \Leftrightarrow [p, A; (1-p), B] \succeq [q, A; (1-q), B]$$

Decomposability

$$[p, A; (1-p), [q, B; (1-q), C]] \sim [p, A; (1-pq), B; (1-p(1-q)), C]$$

Rational preferences contd.

- ▶ Violating the constraints leads to self-evident irrationality
- ▶ **Example 2.7.** An agent with intransitive preferences can be induced to give away all its money:
 - ▶ If $B \succ C$, then an agent who has C would pay (say) 1 cent to get B
 - ▶ If $A \succ B$, then an agent who has B would pay (say) 1 cent to get A
 - ▶ If $C \succ A$, then an agent who has A would pay (say) 1 cent to get C



3 Utilities and Money

Ramseys Theorem and Value Functions

- **Theorem 3.1.** (Ramsey, 1931; von Neumann and Morgenstern, 1944)
Given a rational set of preferences there exists a real-valued function U such that

$$U(A) \geq U(B), \text{ iff } A \succeq B \text{ and } U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$

- These are existence theorems, uniqueness not guaranteed.
- Note: Agent behavior is invariant w.r.t. positive linear transformation, i.e.

$$U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0$$

behaves exactly like U .

- Observation: With deterministic prizes only (no lottery choices), only a total ordering on prizes can be determined.
- Definition 3.2. We call a total ordering on states a value function or ordinal utility function.

Maximizing Expected Utility (Ideas)

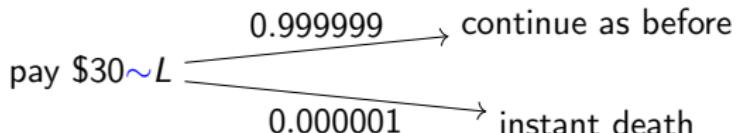
- ▶ **Definition 3.3 (MEU principle).** We call an **action rational** if it **maximizes expected utility (MEU)**. An **agent** is called **rational**, iff it always chooses a rational action.
- ▶ **Note:** An **agent** can be entirely **rational** (consistent with **MEU**) without ever representing or manipulating **utilities** and probabilities.
- ▶ **Example 3.4.** A lookup table for perfect tic tac toe.
- ▶ But an observer can construct a **value function** V by observing the **agent's preferences**.
(even if the agent does not know V)

Maximizing Expected Utility (Definitions)

- ▶ We first formalize the notion of expectation of a random variable.
- ▶ **Definition 3.5.** Given a probability model $\langle \Omega, P \rangle$ and a $X : \Omega \rightarrow \mathbb{R}_0^+$ a random variable, then $E(X) := \sum_{x \in \Omega} P(X = x) \cdot x$ is called the **expected-value** (or **expectation**) of X .
- ▶ Idea: Apply this idea to get the **expected utility** of an action, this is stochastic:
 - ▶ In **partially observable environments**, we do not know the current state.
 - ▶ In **nondeterministic environments**, we cannot be sure of the result of an action.
- ▶ **Definition 3.6.** Let \mathcal{A} be an agent with a set Ω of states and a utility function $U : \Omega \rightarrow \mathbb{R}_0^+$, then for each action a , we define a random variable R_a whose values are the results of performing a in the current state.
- ▶ **Definition 3.7.** The **expected utility** $EU(a|e)$ of an action a (given evidence e) is

$$EU(a|e) := \sum_{s \in \Omega} P(R_a = s | a, e) \cdot U(s)$$

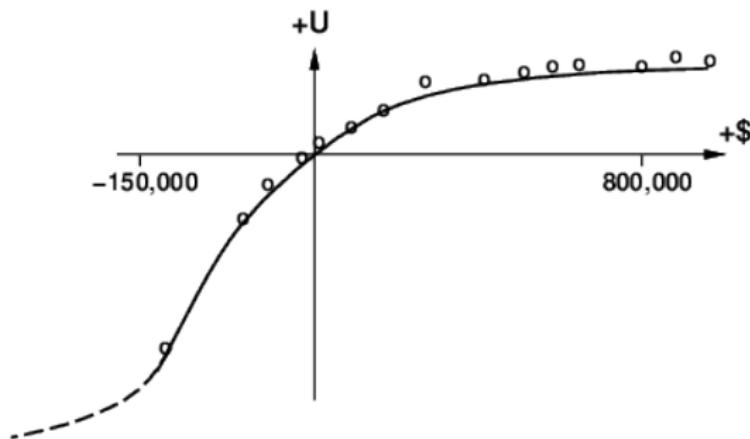
- ▶ Utilities map states to real numbers. Which numbers?
- ▶ **Definition 3.8 (Standard approach to assessment of human utilities).**
Compare a given state A to a standard lottery L_p that has
 - ▶ “best possible prize” u_T with probability p
 - ▶ “worst possible catastrophe” u_L with probability $1 - p$adjust lottery probability p until $A \sim L_p$. Then $U(A) = p$.
- ▶ **Example 3.9.** Choose $u_T \hat{=} \text{current state}$, $u_L \hat{=} \text{instant death}$



- ▶ **Definition 3.10.** **Normalized utilities:** $u_{\top} = 1$, $u_{\perp} = 0$.
- ▶ **Definition 3.11.** **Micromorts:** one-millionth chance of instant death.
- ▶ **Micromorts** are useful for Russian roulette, paying to reduce product risks, etc.
- ▶ **Problem:** What is the value of a **micromort**?
- ▶ **Ask them directly:** What would you pay to avoid playing Russian roulette with a million-barrelled revolver? (**very large numbers**)
- ▶ **But their behavior suggests a lower price:**
 - ▶ Driving in a car for 370km incurs a risk of one **micromort**;
 - ▶ Over the life of your car – say, 150,000km that's 400 **micromorts**.
 - ▶ People appear to be willing to pay about 10,000€ more for a safer car that halves the risk of death. ($\sim 25\text{€}$ per **micromort**)
- ▶ This figure has been confirmed across many individuals and risk types.
- ▶ Of course, this argument holds only for small risks. Most people won't agree to kill themselves for 25M€.
- ▶ **Definition 3.12.** **QALYs: quality-adjusted life years**
- ▶ **Application:** **QALYs** are useful for medical decisions involving substantial risk.

Money vs. Utility

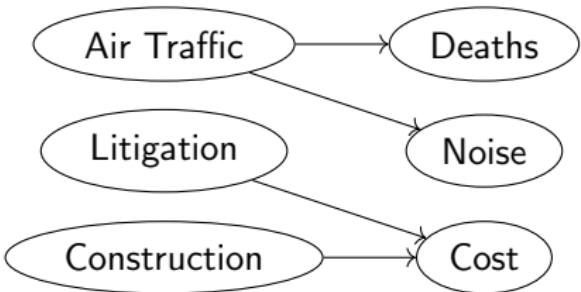
- ▶ Money does *not* behave as a utility function should.
- ▶ Given a lottery L with expected monetary value $\text{EMV}(L)$, usually $U(L) < U(\text{EMV}(L))$, i.e., people are **risk averse**.
- ▶ **Utility curve:** For what probability p am I indifferent between a prize x and a lottery $[p, M\$; (1-p), 0\$]$ for large M ?
- ▶ Typical empirical data, extrapolated with **risk prone** behavior for debtors:



- ▶ **Empirically:** comes close to the logarithm on the positive numbers.

4 Multi-Attribute Utility

- ▶ How can we handle utility functions of many variables $X_1 \dots X_n$?
- ▶ Example 4.1 (Assessing an Airport Site).

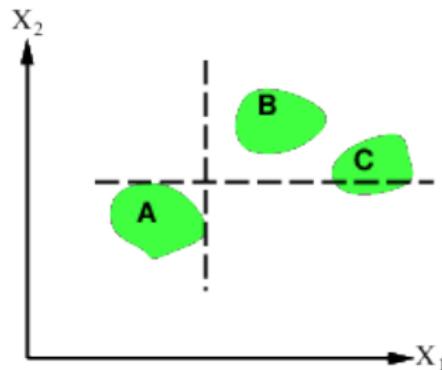
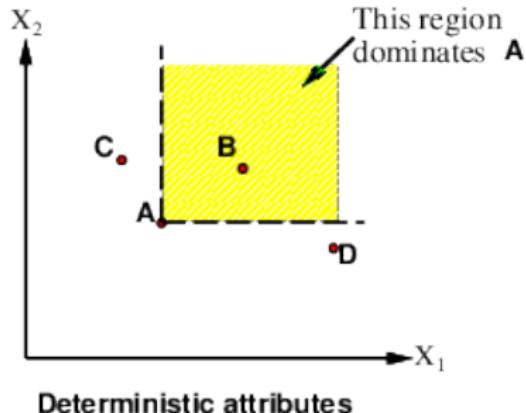


what is
 $U(Deaths, Noise, Cost)$ for
a projected airport?

- ▶ How can complex utility functions be assessed from preference behaviour?
- ▶ Idea 1: Identify conditions under which decisions can be made without complete identification of $U(x_1, \dots, x_n)$.
- ▶ Idea 2: Identify various types of independence in preferences and derive consequent canonical forms for $U(x_1, \dots, x_n)$.

Strict Dominance

- ▶ Typically define attributes such that U is monotonic in each argument. (wlog. growing)
- ▶ **Definition 4.2.** Choice B **strictly dominates** choice A iff $X_i(B) \geq X_i(A)$ for all i (and hence $U(B) \geq U(A)$)



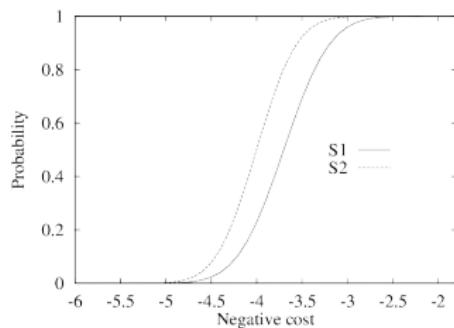
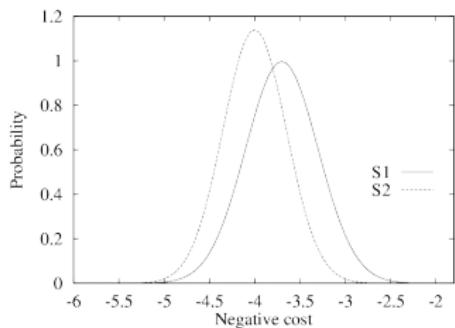
- ▶ **Observation:** Strict dominance seldom holds in practice (life is difficult) but is useful for narrowing down the field of contenders.
- ▶ For uncertain attributes strict dominance is even more unlikely.

Stochastic Dominance

- ▶ **Definition 4.3.** Distribution p_2 **stochastically dominates** distribution p_1 iff the **cummulative distribution** of p_2 **strictly dominates** that for p_1 for all t , i.e.

$$\int_t^{-\infty} p_1(x) dx \leq \int_t^{-\infty} p_2(x) dx$$

- ▶ **Example 4.4.**

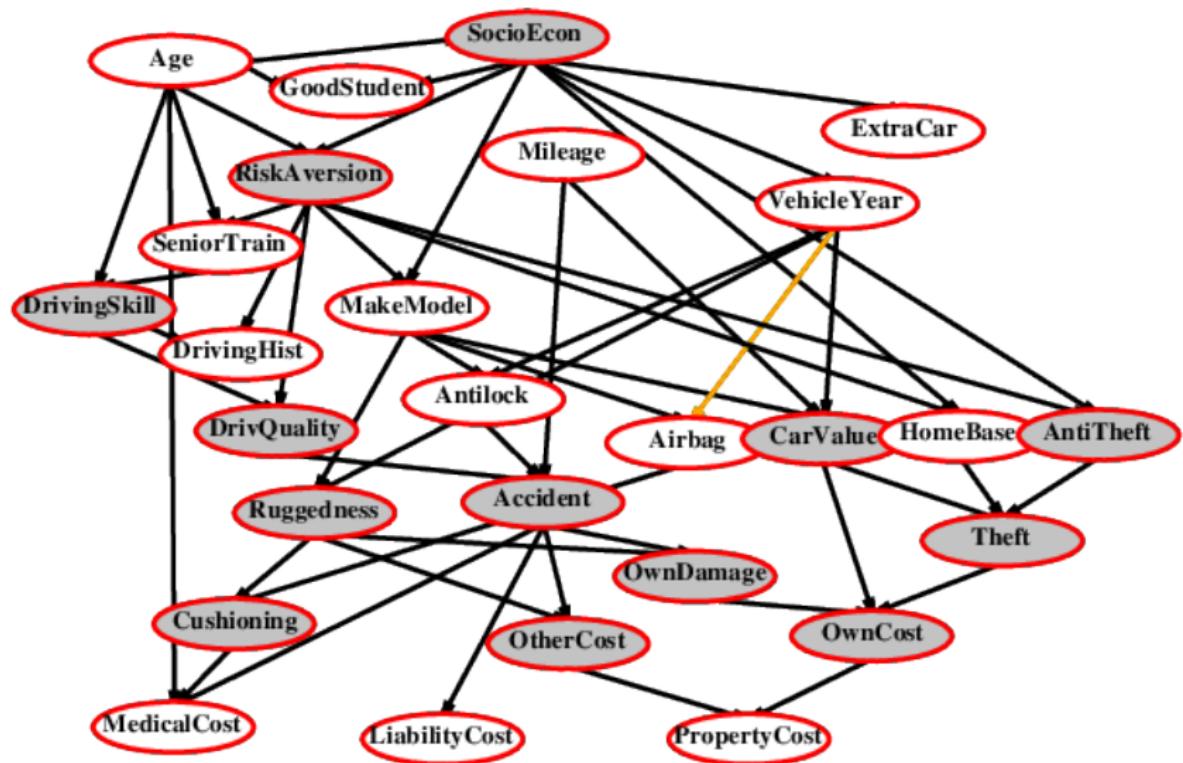


- ▶ **Observation 4.5.** If U is monotonic in x , then A_1 with outcome distribution p_1 stochastically dominates A_2 with outcome distribution p_2 :

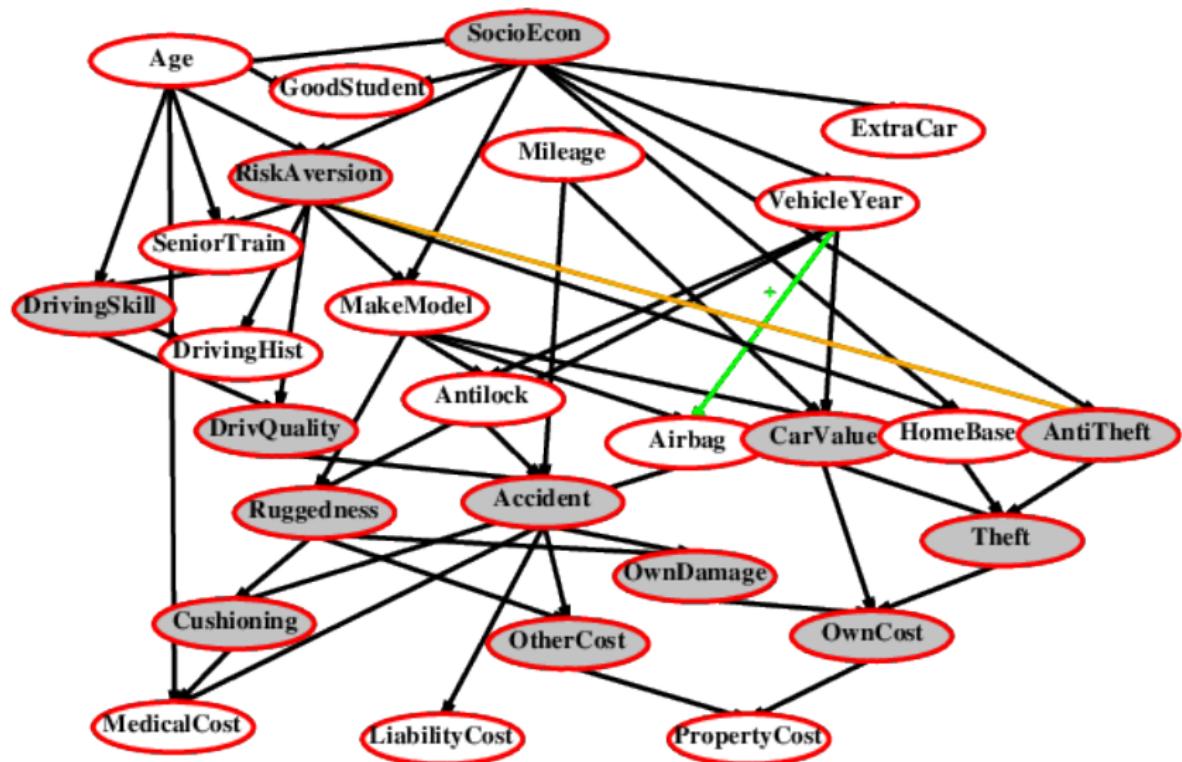
$$\int_{-\infty}^{\infty} (p_1(x) U(x)) dx \geq \int_{-\infty}^{\infty} (p_2(x) U(x)) dx$$

- ▶ Multiattribute case: stochastic dominance on all attributes \leadsto optimal
- ▶ Stochastic dominance can often be determined without exact distributions using qualitative reasoning
- ▶ **Example 4.6.** Construction cost increases with distance from city S_1 is closer to the city than $S_2 \leadsto S_1$ stochastically dominates S_2 on cost
- ▶ **Example 4.7.** Injury increases with collision speed
- ▶ Idea: Annotate Bayesian networks with stochastic dominance information.
- ▶ **Definition 4.8.** $X \xrightarrow{+} Y$ (X positively influences Y) means that $P(Y|x_1, z)$ stochastically dominates $P(Y|x_2, z)$ for every value z of Y 's other parents Z and all x_1 and x_2 with $x_1 \geq x_2$.

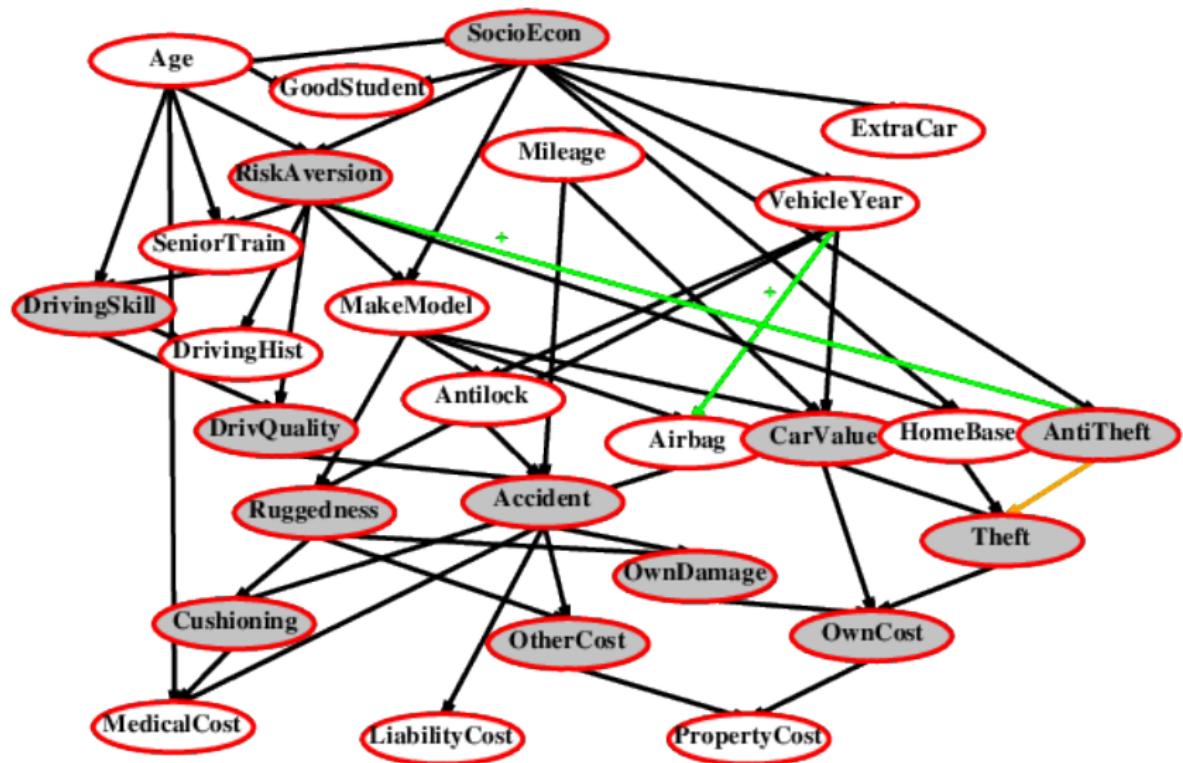
Label the arcs + or - for influence in a Bayesian Network



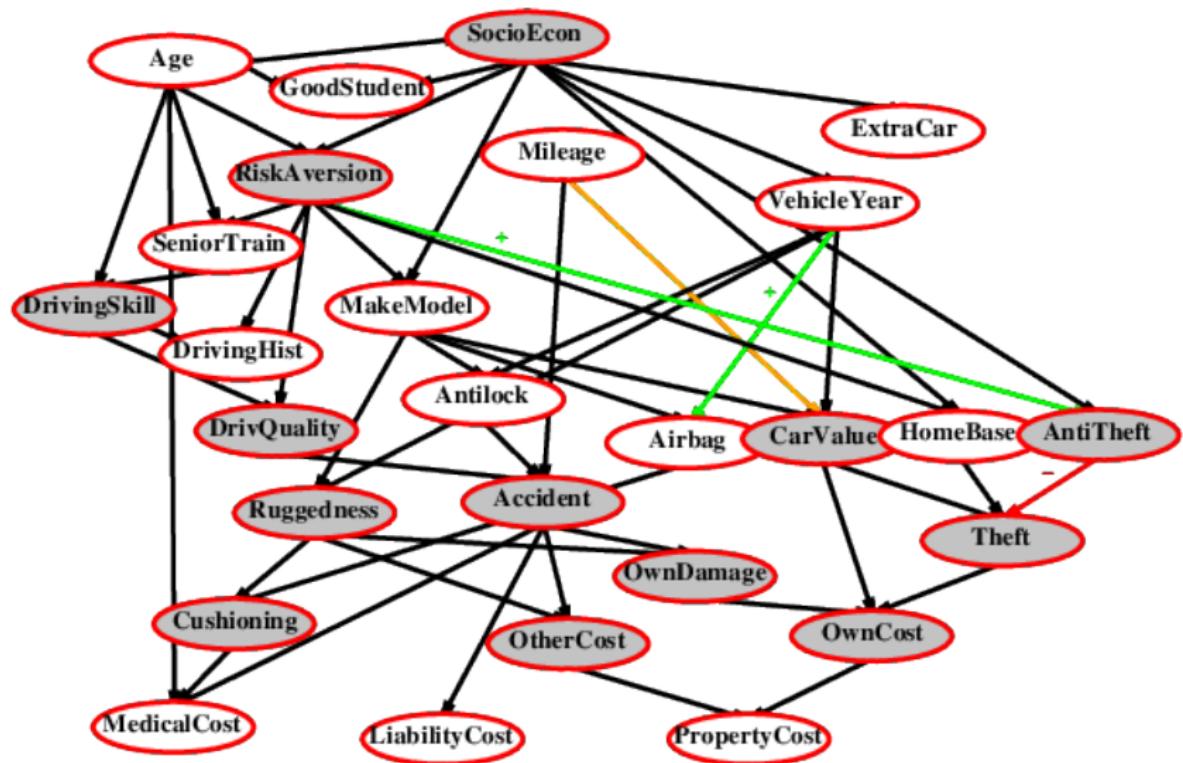
Label the arcs + or - for influence in a Bayesian Network



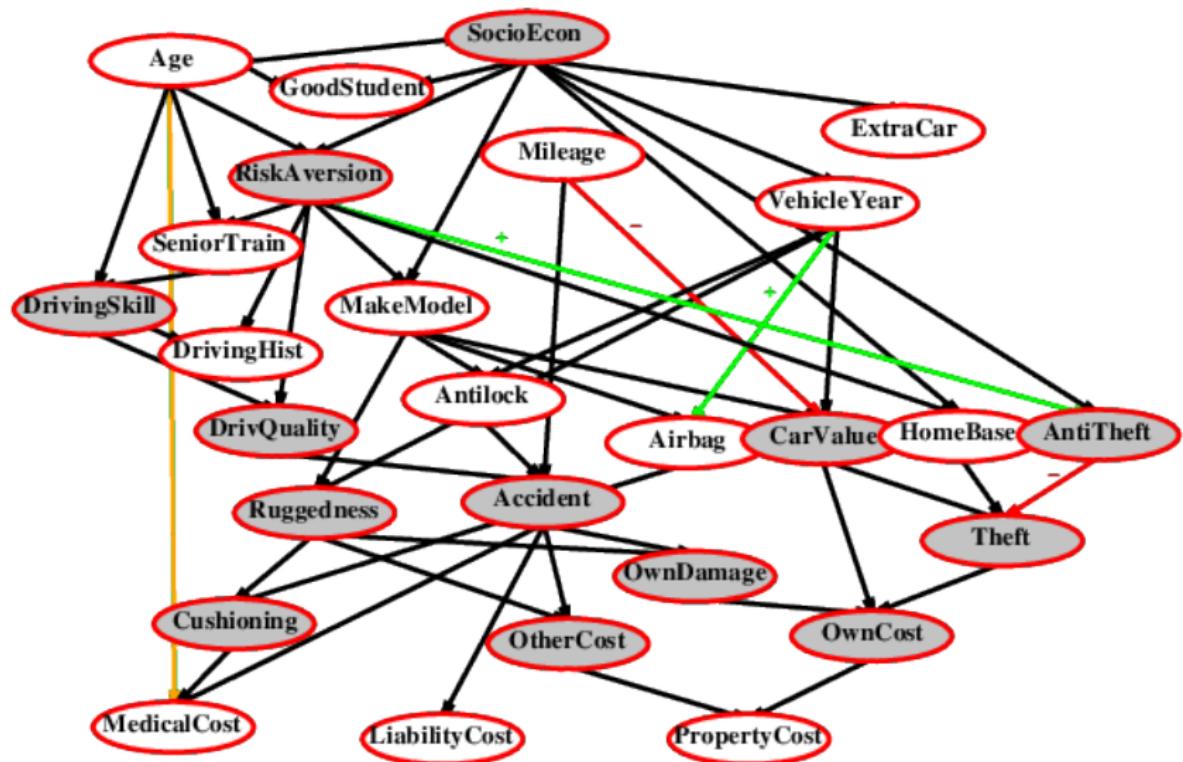
Label the arcs + or - for influence in a Bayesian Network



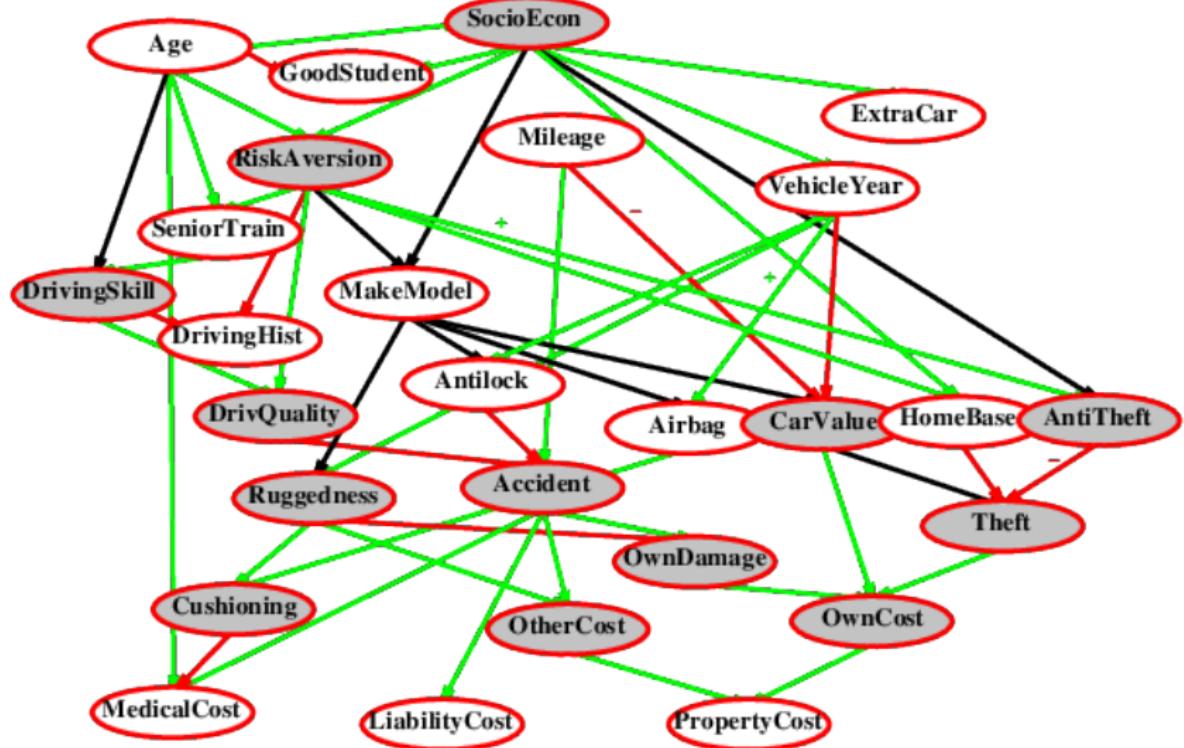
Label the arcs + or - for influence in a Bayesian Network



Label the arcs + or - for influence in a Bayesian Network



Label the arcs + or - for influence in a Bayesian Network



- ▶ **Observation 4.9.** n attributes with d values each \leadsto need d^n values to determine utility function $U(x_1, \dots, x_n)$. (worst case)

▶ Assumption: Preferences of real agents have much more structure.

▶ Approach: Identify regularities and prove representation theorems based on these:

$$U(x_1, \dots, x_n) = F(f_1(x_1), \dots, f_n(x_n))$$

where F is simple, e.g. addition.

- ▶ Note the similarity to Bayesian networks that decompose the full joint probability distribution.

Preference structure: Deterministic

- ▶ Recall: In deterministic environments an agent has a value function.
- ▶ **Definition 4.10.** X_1 and X_2 **preferentially independent** of X_3 iff preference between $\langle x_1, x_2, x_3 \rangle$ and $\langle x'_1, x'_2, x_3 \rangle$ does not depend on x_3 .
- ▶ **Example 4.11.** E.g., $\langle \text{Noise}, \text{Cost}, \text{Safety} \rangle$: are preferentially independent $\langle 20,000 \text{ suffer}, 4.6 \text{ G\$}, 0.06 \text{ deaths/mpm} \rangle$ vs. $\langle 70,000 \text{ suffer}, 4.2 \text{ G\$}, 0.06 \text{ deaths/mpm} \rangle$
- ▶ **Theorem 4.12 (Leontief, 1947).** If every pair of attributes is preferentially independent of its complement, then every subset of attributes is preferentially independent of its complement: **mutual preferential independence**.
- ▶ **Theorem 4.13 (Debreu, 1960).** Mutual preferential independence implies that there is an **additive value function**: $V(S) = \sum_i V_i(X_i(S))$, where V_i is a **value function** referencing just one variable X_i .
- ▶ Hence assess n single-attribute functions. (often a good approximation)
- ▶ **Example 4.14.** The **value function** for the airport decision might be

$$V(\text{noise}, \text{cost}, \text{deaths}) = -\text{noise} \cdot 10^4 - \text{cost} - \text{deaths} \cdot 10^{12}$$

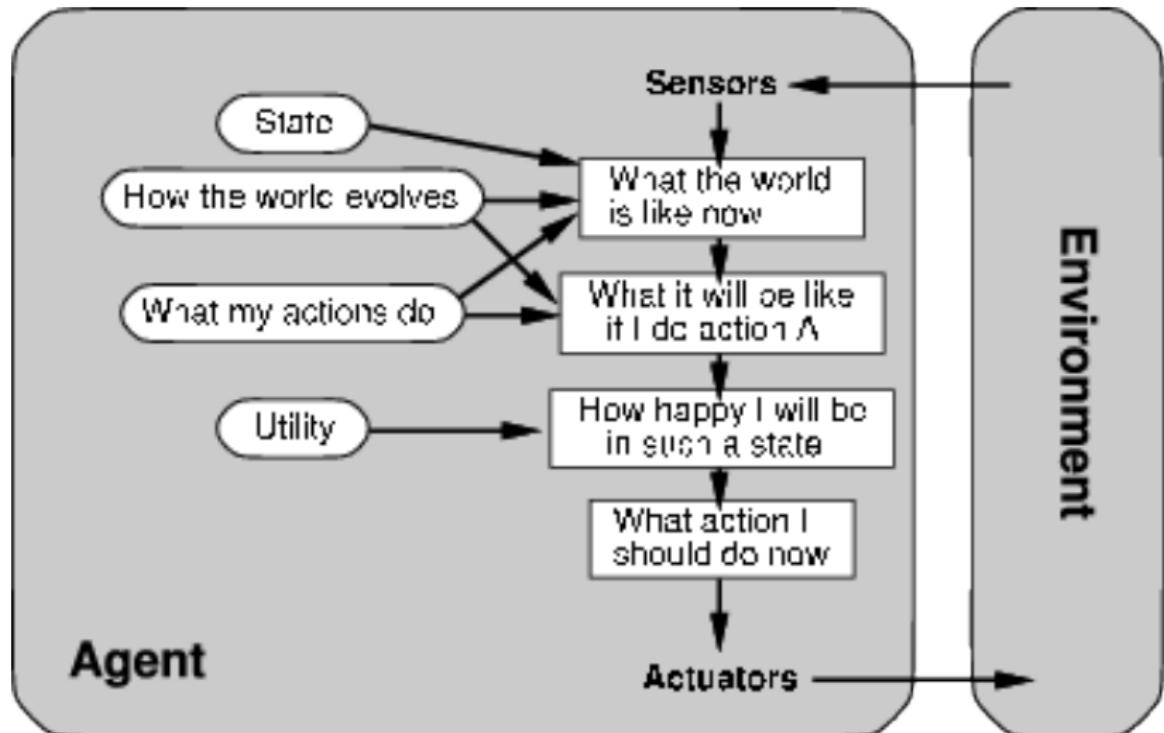
- ▶ Need to consider **preferences** over **lotteries** and real **utility functions** (not just **value functions**)
- ▶ **Definition 4.15.** X is **utility independent** of Y iff preferences over **lotteries** in X do not depend on particular values in Y.
- ▶ **Definition 4.16.** A set X is **mutually utility independent**, iff each subset is **utility independent** of its complement.
- ▶ **Theorem 4.17.** For **mutually utility independent** sets there is a multiplicative utility function: [Kee74]

$$U = k_1 U_1 + k_2 U_2 + k_3 U_3 + k_1 k_2 U_1 U_2 + k_2 k_3 U_2 U_3 + k_3 k_1 U_3 U_1 + k_1 k_2 k_3 U_1 U_2 U_3$$

- ▶ **System Support:** Routine procedures and software packages for generating preference tests to identify various canonical families of **utility functions**.

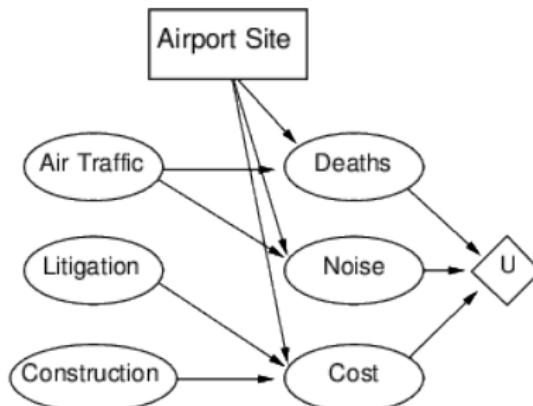
5 Decision Networks

Utility-Based Agents (Recap)



Decision networks

- ▶ **Definition 5.1.** A **decision network** is a Bayesian network with added **action nodes** and **utility nodes** (also called **value nodes**) that enable **rational** decision making.
- ▶ **Example 5.2 (Choosing an Airport Site).**



- ▶ **Algorithm:**

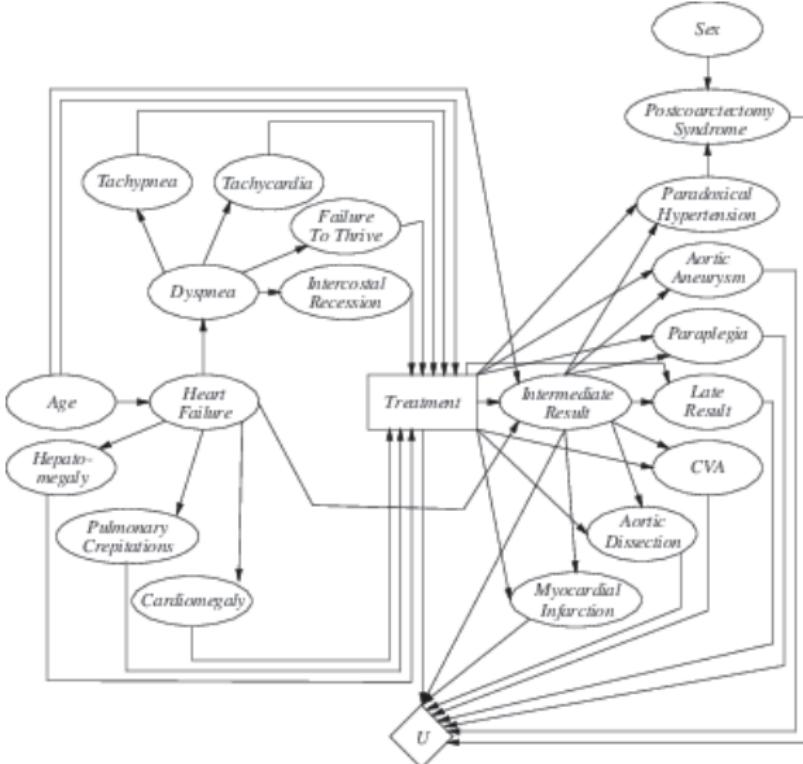
For each value of action node

compute expected value of utility node given action, evidence

Return MEU action (via argmax)

Decision Networks: Example

- Example 5.3 (A Decision-Network for Aortic Coarctation). from [Luc96]



- ▶ **Question:** How do you create a model like the one from 5.3?
- ▶ **Answer:** By a systematic process of the form: (after [Luc96])
 1. **Create a causal model:** a graph with nodes for symptoms, disorders, treatments, outcomes, and their influences (edges)
 2. **Simplify to a qualitative decision model:** remove vars not involved in treatment decisions
 3. **Assign probabilities** (↔ Bayesian network)
e.g. from patient databases, literature studies, or the expert's subjective assessments
 4. **Assign utilities:** (e.g. in QUALYs or micromorts)
 5. **Verify and refine the model** wrt. a gold standard given by experts
e.g. refine by "running the model backwards" and compare with the literature
 6. **Perform sensitivity analysis:** (important step in practice)
 - ▶ is the optimal treatment decision robust against small changes in the parameters? (if yes
↔ great! if not, collect better data)

6 The Value of Information

What if we do not have all information we need?

- ▶ **It is Well-Known:** One of the most important parts of decision making is knowing what questions to ask.
- ▶ **Example 6.1 (Medical Diagnosis).**
 - ▶ We do not expect a doctor to already know the results of the diagnostic tests when the patient comes in.
 - ▶ Tests are often expensive, and sometimes hazardous (directly or by delaying treatment)
 - ▶ **Therefore:** Only test, if
 - ▶ knowing the results lead to a significantly better treatment plan,
 - ▶ information from test results is not drowned out by a-priori likelihood.
- ▶ **Definition 6.2. Information value theory** enables the **agent** to make decisions on **information gathering** rationally.
- ▶ **Intuition:** Simple form of **sequential decision making**. (action only impacts belief state).
- ▶ **Intuition:** With the information, we can change the action to the **actual** information, rather than the average.

Value of Information by Example

- ▶ **Idea:** Compute value of acquiring each possible piece of evidence.
- ▶ **We will see:** This can be done directly from a **decision network**.
- ▶ **Example 6.3 (Buying Oil Drilling Rights).**
There are n blocks of rights, exactly one has oil, worth $k\text{€}$, in particular
 - ▶ Prior probabilities $1/n$ each, mutually exclusive.
 - ▶ Current price of each block is $(k/n)\text{€}$.
 - ▶ "Consultant" offers accurate survey of block 3. What's a fair price?
- ▶ **Solution:** Compute expected value of information $\hat{=}$ expected value of best action given the information minus expected value of best action without information.
- ▶ **Example 6.4 (Oil Drilling Rights contd.).**
 - ▶ Survey may say *oil in block 3 with probability $1/n$* \leadsto buy block 3 for $(k/n)\text{€}$ make profit of $(k - k/n)\text{€}$.
 - ▶ Survey may say *no oil in block 3 with probability $(n-1)/n$* \leadsto buy another block, make profit of $k/(n-1) - k/n\text{€}$.
 - ▶ Expected profit is $\frac{1}{n} \cdot \frac{(n-1)k}{n} + \frac{n-1}{n} \cdot \frac{k}{n(n-1)} = \frac{k}{n}$.
 - ▶ So, we should pay up to $(k/n)\text{€}$ for the information. (as much as block 3 is worth)

General formula (VPI)

- Given current evidence E , possible actions $a \in A$ with outcomes in S_a , and current best action α

$$\text{EU}(\alpha|E) = \max_{a \in A} \left(\sum_{s \in S_a} U(s) \cdot P(s|E, a) \right)$$

- Suppose we knew $F = f$ (new evidence), then we would choose α_f s.t.

$$\text{EU}(\alpha_f|E, F = f) = \max_{a \in A} \left(\sum_{s \in S_a} U(s) \cdot P(s|E, a, F = f) \right)$$

here, F is a random variable with domain D whose value is currently unknown.

- Idea: So we must compute the expected gain over all possible values $f \in D$.
- Definition 6.5.** Let F be a random variable with domain D , then the value of perfect information (VPI) on F given evidence E is defined as

$$\text{VPI}_E(F) := \left(\sum_{f \in D} P(F = f|E) \cdot \text{EU}(\alpha_f|E, F = f) \right) - \text{EU}(\alpha|E)$$

where $\alpha_f = \underset{a \in A}{\operatorname{argmax}} \text{EU}(a|E, F = f)$ and A the set of possible actions.

- ▶ Non-negative: in *expectation*, not *post hoc*: $\text{VPI}_E(F) \geq 0$ for all j and E
- ▶ Non-additive: $\text{VPI}_E(F, G) \neq \text{VPI}_E(F) + \text{VPI}_E(G)$ (consider, e.g., obtaining F twice)
- ▶ Order-independent:

$$\text{VPI}_E(F, G) = \text{VPI}_E(F) + \text{VPI}_{E,F}(G) = \text{VPI}_E(G) + \text{VPI}_{E,G}(F)$$

- ▶ Note: when more than one piece of evidence can be gathered, maximizing VPI for each to select one is not always optimal
~ evidence-gathering becomes a sequential decision problem.

A simple Information-Gathering Agent

- ▶ **Definition 6.6.** A simple **information gathering agent**. (gathers info before acting)

```
function Information—Gathering—Agent (percept) returns an action
```

```
    persistent:  $D$ , a decision network
```

```
    integrate percept into  $D$ 
```

```
     $j := \operatorname{argmax}_k \text{VPI}_E(E_k)/\text{Cost}(E_k)$ 
```

```
    if  $\text{VPI}_E(E_j) > \text{Cost}(E_j)$  return Request( $E_j$ )
```

```
    else return the best action from  $D$ 
```

The next percept after $\text{Request}(E_j)$ provides a value for E_j .

- ▶ **Problem:** The **information gathering** implemented here is **myopic**, i.e. calculating **VPI** as if only a single evidence variable will be acquired. (cf. **greedy search**)
- ▶ But it works relatively well in practice. (e.g. outperforms humans for selecting diagnostic tests)

Chapter 23 Temporal Probability Models

- ▶ Modeling time and uncertainty for sequential environments.
- ▶ Markov inference: Filtering, prediction, smoothing, and most likely explanation.
- ▶ Hidden Markov models
- ▶ Dynamic Bayesian networks
- ▶ Particle filtering?
- ▶ Further Algorithms and Topics?

1 Modeling Time and Uncertainty

- ▶ **Observation 1.1.** The world changes; we need to track and predict it!
- ▶ **Example 1.2.** Consider the following decision problems:
 - ▶ Vehicle diagnosis: car state constant during diagnosis \leadsto episodic!
 - ▶ Diabetes management: patient state can quickly deteriorate \leadsto sequential!
- ▶ Here we lay the mathematical foundations for the latter.
- ▶ **Definition 1.3.** A **temporal probability model** is a **probability model**, where possible worlds are indexed by a **time structure** $\langle S, \preceq \rangle$.
- ▶ We restrict ourselves to linear, discrete time structures, i.e. $\langle S, \preceq \rangle = \langle \mathbb{N}, \leq \rangle$.
(Step size irrelevant for theory, depends on problem in practice)
- ▶ **Definition 1.4 (Basic Setup).** A **temporal probability model** has two sets of random variables indexed by \mathbb{N} .
 - ▶ $X_t \triangleq$ set of (unobservable) **state variables** at time $t \geq 0$
e.g., BloodSugar_t , StomachContents_t , etc.
 - ▶ $E_t \triangleq$ set of (observable) **evidence variables** at time $t > 0$
e.g., $\text{MeasuredBloodSugar}_t$, PulseRate_t , FoodEaten_t
- ▶ **Notation:** $X_{a:b} = X_a, X_{a+1}, \dots, X_{b-1}, X_b$

- ▶ **Example 1.5 (Umbrellas).** You are a security guard in a secret underground facility, want to know if it is raining outside. Your only source of information is whether the director comes in with an umbrella.
 - ▶ State variables: R_0, R_1, R_2, \dots ,
 - ▶ Observations (evidence variables): U_1, U_2, U_3, \dots

Markov Processes

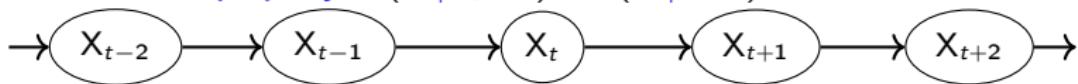
- Idea: Construct a Bayesian network from these variables. (parents?)
- Intuition: Increasing the order adds “memory” to the process, Markov chains have none.

Markov Processes

- ▶ Idea: Construct a Bayesian network from these variables. (parents?)
 - ▶ **Definition 1.6.** **Markov property**: X_t only depends on a bounded subset of $X_{0:t-1}$. (in particular not on $E_{1:t}$)
 - ▶ **Definition 1.7.** A (discrete-time) **Markov process** is a sequence of random variables with the Markov property.
-
- ▶ Intuition: Increasing the order adds “memory” to the process, Markov chains have none.

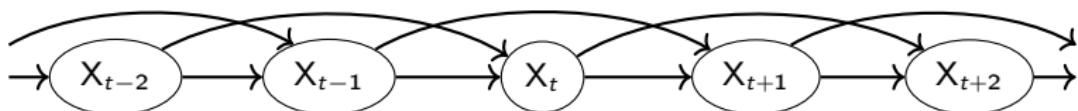
Markov Processes

- Idea: Construct a Bayesian network from these variables. (parents?)
- **Definition 1.6.** **Markov property:** X_t only depends on a bounded subset of $X_{0:t-1}$. (in particular not on $E_{1:t}$)
- **Definition 1.7.** A (discrete-time) **Markov process** is a sequence of random variables with the **Markov property**.
- **Definition 1.8.** We say that a **Markov process** has the *n*th-order **Markov property** for $n \in \mathbb{N}^+$, iff $P(X_t | X_{0:t-1}) = P(X_t | X_{t-n:t-1})$. Special Cases
 - **First order Markov property:** $P(X_t | X_{0:t-1}) = P(X_t | X_{t-1})$



A first-order Markov process is called a **Markov chain**.

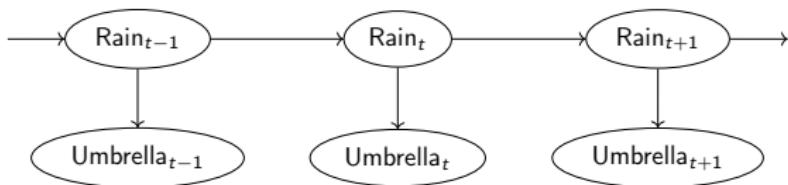
- **Second order Markov property:** $P(X_t | X_{0:t-1}) = P(X_t | X_{t-2}, X_{t-1})$



- Intuition: Increasing the order adds “memory” to the process, **Markov chains** have none.
- Preview: We will use **Markov processes** to model **sequential environments**.

Markov Process Example: The Umbrella

- ▶ **Example 1.9 (Umbrellas continued).** We model the situation in a [Bayesian network](#):

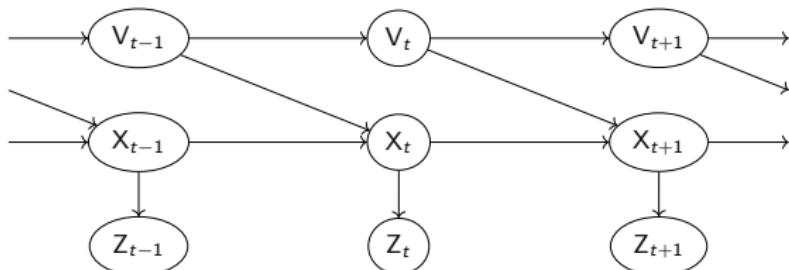


- ▶ Problem: First order Markov property not exactly true in real world!
- ▶ Possible fixes:
 1. Increase the [order](#) of the Markov process. (more dependencies)
 2. Add [state variables](#), e.g., add Temp_t , Pressure_t . (more information sources)

We will see the second in another example: tracking robot motion.

Markov Process Example: Robot Motion

- ▶ **Example 1.10 (Random Robot Motion).** To track a robot wandering randomly on the X/Y plane, use the following **Markov chain**

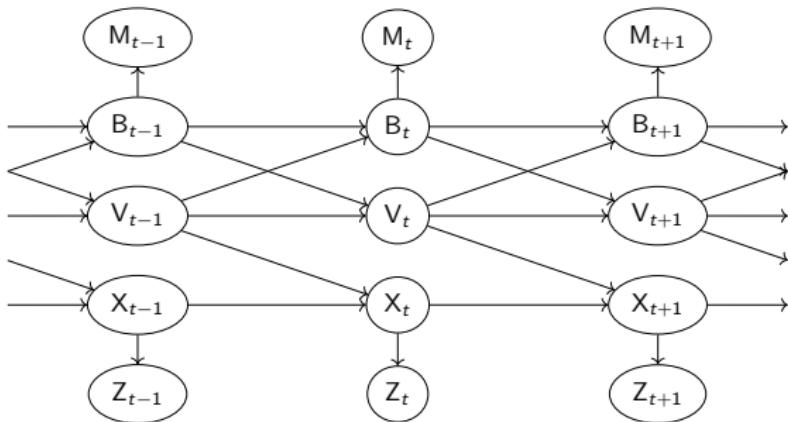


We use Newton's laws to calculate the new position

- ▶ the velocity V_i may change unpredictably.
 - ▶ the position X_i depends on previous position X_{i-1} and velocity V_{i-1}
 - ▶ the position X_i influences the observed position Z_i .
- ▶ **Example 1.11 (Battery Powered Robot).** **Markov property** violated!
 - ▶ Battery exhaustion has a systematic effect on the change in velocity.
 - ▶ This depends on how much power was used by all previous manoeuvres.
 - ▶ Idea: We can restore the **Markov property** by including a **state variable** for the charge level B_t .
(Better still: Battery level sensor)

Markov Process Example: Robot Motion

► Example 1.12 (Battery Powered Robot Motion).



- Battery level B_i is influenced by previous level B_{i-1} and velocity V_{i-1} .
- Velocity V_i is influenced by previous level B_{i-1} and velocity V_{i-1} as well.
- Battery meter M_i is only influenced by Battery level B_i .

Stationary Markov Processes as Transition Models

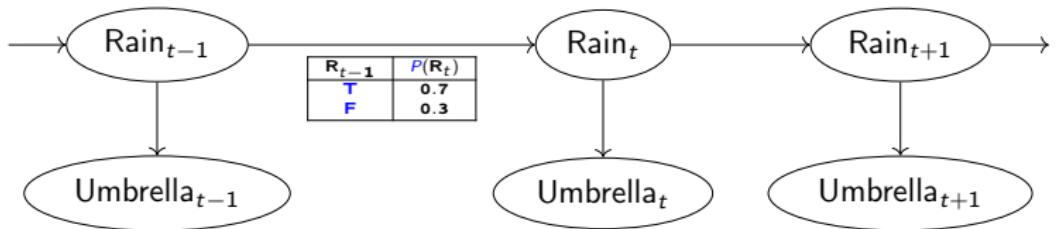
- ▶ **Theorem 1.13.** Let M be a Markov chain with state variables X_t evidence variables E_t ; then $P(X_t|X_{t-1})$ is the transition model and $P(E_t|X_{0:t}, E_{1:t-1})$ the sensor model of M .
- ▶ Problem: Even with Markov property the transition model is infinite. ($t \in \mathbb{N}$)

Stationary Markov Processes as Transition Models

- ▶ **Theorem 1.13.** Let M be a **Markov chain** with **state variables** X_t **evidence variables** E_t ; then $P(X_t|X_{t-1})$ is the **transition model** and $P(E_t|X_{0:t}, E_{1:t-1})$ the **sensor model** of M .
- ▶ Problem: Even with **Markov property** the **transition model** is **infinite**. ($t \in \mathbb{N}$)
- ▶ **Definition 1.14.** A **Markov chain** is called **stationary** if the **transition model** is independent of time, i.e. $P(X_t|X_{t-1})$ is the same for all t .

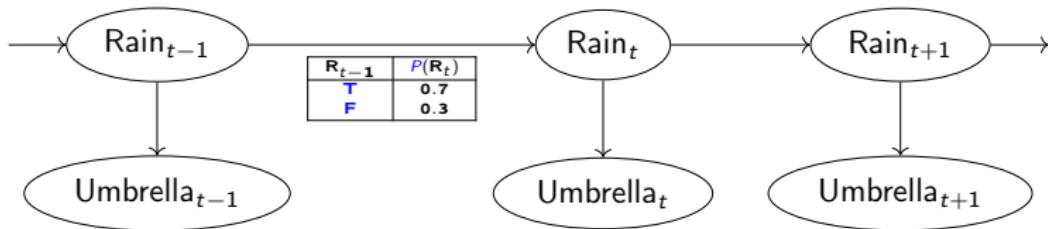
Stationary Markov Processes as Transition Models

- ▶ **Theorem 1.13.** Let M be a **Markov chain** with **state variables** X_t **evidence variables** E_t ; then $P(X_t|X_{t-1})$ is the **transition model** and $P(E_t|X_{0:t}, E_{1:t-1})$ the **sensor model** of M .
- ▶ Problem: Even with **Markov property** the **transition model** is **infinite**. ($t \in \mathbb{N}$)
- ▶ **Definition 1.14.** A **Markov chain** is called **stationary** if the **transition model** is independent of time, i.e. $P(X_t|X_{t-1})$ is the same for all t .
- ▶ **Example 1.15 (Umbrellas are stationary).** $P(R_t|R_{t-1})$ does not depend on t . (need only one table)



Stationary Markov Processes as Transition Models

- ▶ **Theorem 1.13.** Let M be a **Markov chain** with **state variables** X_t **evidence variables** E_t ; then $P(X_t|X_{t-1})$ is the **transition model** and $P(E_t|X_{0:t}, E_{1:t-1})$ the **sensor model** of M .
- ▶ Problem: Even with **Markov property** the **transition model** is **infinite**. ($t \in \mathbb{N}$)
- ▶ **Definition 1.14.** A **Markov chain** is called **stationary** if the **transition model** is independent of time, i.e. $P(X_t|X_{t-1})$ is the same for all t .
- ▶ **Example 1.15 (Umbrellas are stationary).** $P(R_t|R_{t-1})$ does not depend on t . (need only one table)



- ▶ ⚠: Don't confuse "**stationary**" (Markov processes) with "**static**" (environments).
- ▶ We restrict ourselves to **stationary Markov processes** in AI-1.

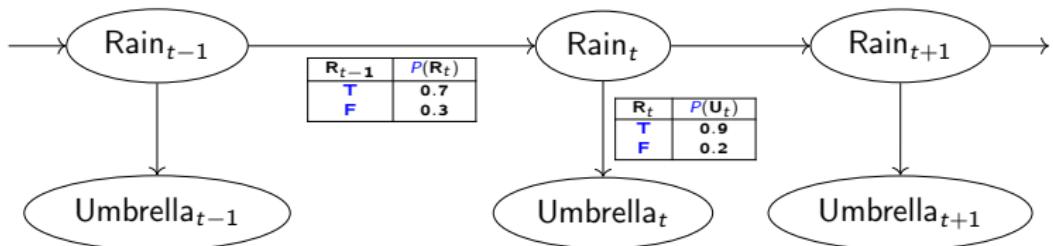
- ▶ **Recap:** The **sensor model** predicts the influence of percepts (and the world state) on the belief state. **(used during update)**
- ▶ **Problem:** The **evidence variables** E_t could depend on previous variables as well as the current state.

- ▶ **Recap:** The **sensor model** predicts the influence of percepts (and the world state) on the belief state. (used during update)
- ▶ **Problem:** The **evidence variables** E_t could depend on previous variables as well as the current state.
- ▶ We restrict dependency to current state. (otherwise state repn. deficient)
- ▶ **Definition 1.16.** We say that a **sensor model** has the **sensor Markov property**, iff $P(E_t | X_{0:t}, E_{1:t-1}) = P(E_t | X_t)$

- ▶ **Recap:** The **sensor model** predicts the influence of percepts (and the world state) on the belief state. (used during update)
- ▶ **Problem:** The **evidence variables** E_t could depend on previous variables as well as the current state.
- ▶ We restrict dependency to current state. (otherwise state repn. deficient)
- ▶ **Definition 1.16.** We say that a **sensor model** has the **sensor Markov property**, iff $P(E_t | X_{0:t}, E_{1:t-1}) = P(E_t | X_t)$
- ▶ **Assumptions on Sensor Models:** We usually assume the **sensor Markov property** and make it **stationary** as well: $P(E_t | X_t)$ is fixed for all t .

Umbrellas, the full Story

- ▶ Example 1.17 (Umbrellas, Transition & Sensor Models).



Note that influence goes from Rain_t to Umbrella_t . (causal dependency)

- ▶ **Observation 1.18.** If we additionally know the initial prior probabilities $P(X_0)$ (\equiv time $t = 0$), then we can compute the **full joint probability distribution** as

$$P(X_{0:t}, E_{1:t}) = P(X_0) \cdot \prod_{i=1}^t P(X_i | X_{i-1}) \cdot P(E_i | X_i)$$

2 Inference: Filtering, Prediction, and Smoothing

- ▶ **Definition 2.1.** The **Markov inference** tasks consist of **filtering**, **prediction**, **smoothing**, and **most likely explanation** as defined below.
- ▶ **Definition 2.2. Filtering** (or **monitoring**): $P(X_t|e_{1:t})$
computing the **belief state** – input to the decision process of a rational agent.
- ▶ **Definition 2.3. Prediction** (or **state estimation**): $P(X_{t+k}|e_{1:t})$ for $k > 0$
evaluation of possible action sequences. (\cong filtering without the evidence)
- ▶ **Definition 2.4. Smoothing** (or **hindsight**): $P(X_k|e_{1:t})$ for $0 \leq k < t$
better estimate of past states. (essential for learning)
- ▶ **Definition 2.5. Most likely explanation** $\underset{x_{1:t}}{\operatorname{argmax}} (P(x_{1:t}|e_{1:t}))$
speech recognition, decoding with a noisy channel.

- Aim: **Recursive state estimation:** $P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t}))$

Filtering

- ▶ Aim: **Recursive state estimation:** $P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t}))$
- ▶ Project the current distribution forward from t to $t + 1$:

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= P(X_{t+1}|e_{1:t}, e_{t+1}) && \text{(dividing up evidence)} \\ &= \alpha \cdot P(e_{t+1}|X_{t+1}, e_{1:t}) \cdot P(X_{t+1}|e_{1:t}) && \text{(using Bayes' rule)} \\ &= \alpha \cdot P(e_{t+1}|X_{t+1}) \cdot P(X_{t+1}|e_{1:t}) && \text{(sensor Markov property)} \end{aligned}$$

- ▶ Note: $P(e_{t+1}|X_{t+1})$ can be obtained directly from the **sensor model**.

Filtering

- ▶ Aim: **Recursive state estimation:** $P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t}))$
- ▶ Project the current distribution forward from t to $t + 1$:

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= P(X_{t+1}|e_{1:t}, e_{t+1}) && \text{(dividing up evidence)} \\ &= \alpha \cdot P(e_{t+1}|X_{t+1}, e_{1:t}) \cdot P(X_{t+1}|e_{1:t}) && \text{(using Bayes' rule)} \\ &= \alpha \cdot P(e_{t+1}|X_{t+1}) \cdot P(X_{t+1}|e_{1:t}) && \text{(sensor Markov property)} \end{aligned}$$

- ▶ Note: $P(e_{t+1}|X_{t+1})$ can be obtained directly from the sensor model.
- ▶ Continue by conditioning on the current state X_t :

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= \alpha \cdot P(e_{t+1}|X_{t+1}) \cdot (\sum_{x_t} P(X_{t+1}|x_t, e_{1:t}) \cdot P(x_t|e_{1:t})) \\ &= \alpha \cdot P(e_{t+1}|X_{t+1}) \cdot (\sum_{x_t} P(X_{t+1}|x_t) \cdot P(x_t|e_{1:t})) \end{aligned}$$

Filtering

- ▶ Aim: **Recursive state estimation:** $P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t}))$
- ▶ Project the current distribution forward from t to $t + 1$:

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= P(X_{t+1}|e_{1:t}, e_{t+1}) && \text{(dividing up evidence)} \\ &= \alpha \cdot P(e_{t+1}|X_{t+1}, e_{1:t}) \cdot P(X_{t+1}|e_{1:t}) && \text{(using Bayes' rule)} \\ &= \alpha \cdot P(e_{t+1}|X_{t+1}) \cdot P(X_{t+1}|e_{1:t}) && \text{(sensor Markov property)} \end{aligned}$$

- ▶ Note: $P(e_{t+1}|X_{t+1})$ can be obtained directly from the **sensor model**.
- ▶ Continue by conditioning on the current state X_t :

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= \alpha \cdot P(e_{t+1}|X_{t+1}) \cdot (\sum_{x_t} P(X_{t+1}|x_t, e_{1:t}) \cdot P(x_t|e_{1:t})) \\ &= \alpha \cdot P(e_{t+1}|X_{t+1}) \cdot (\sum_{x_t} P(X_{t+1}|x_t) \cdot P(x_t|e_{1:t})) \end{aligned}$$

- ▶ $P(X_{t+1}|X_t)$ is simply the **transition model**, $P(x_t|e_{1:t})$ the “recursive call”.
- ▶ So $f_{1:t+1} = \alpha \cdot \text{FORWARD}(f_{1:t}, e_{t+1})$ where $f_{1:t} = P(X_t|e_{1:t})$ and **FORWARD** is the update shown above. (Time and space *constant* (independent of t))

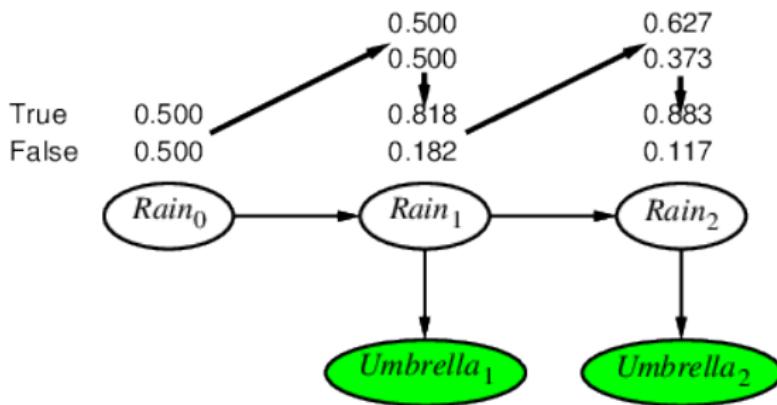
Filtering the Umbrellas

- **Example 2.6.** Say the guard believes $P(R_0) = \langle 0.5, 0.5 \rangle$. On day 1 and 2 the umbrella appears.

$$P(R_1) = \sum_{r_0} P(R_1 | r_0) \cdot P(r_0) = \langle 0.7, 0.3 \rangle \cdot 0.5 + \langle 0.3, 0.7 \rangle \cdot 0.5 = \langle 0.5, 0.5 \rangle$$

Update with evidence for $t = 1$ gives:

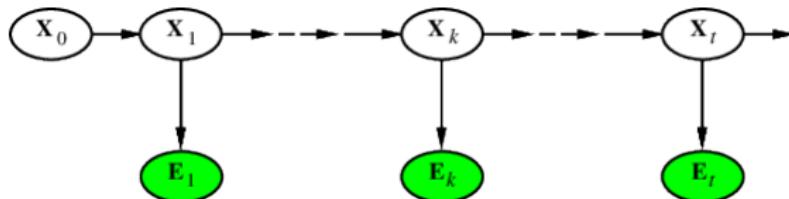
$$P(R_1 | u_1) = \alpha \cdot P(u_1 | R_1) \cdot P(R_1) = \alpha \cdot \langle 0.9, 0.2 \rangle \langle 0.5, 0.5 \rangle = \alpha \cdot \langle 0.45, 0.1 \rangle \approx \langle 0.818, 0.182 \rangle$$



- ▶ Prediction computes future $k > 0$ state distributions: $P(X_{t+k}|e_{1:t})$.
- ▶ Intuition: Prediction is filtering without new evidence.
- ▶ **Lemma 2.7.** $P(X_{t+k+1}|e_{1:t}) = \sum_{x_{t+k}} P(X_{t+k+1}|x_{t+k}) \cdot P(x_{t+k}|e_{1:t})$
- ▶ Proof Sketch: Using the same reasoning as for the FORWARD algorithm for filtering. □
- ▶ **Observation 2.8.** As $k \rightarrow \infty$, $P(x_{t+k}|e_{1:t})$ tends to the stationary distribution of the Markov chain, i.e. the a fixed point under prediction.
- ▶ The mixing time, i.e. the time until prediction reaches the stationary distribution depends on how “stochastic” the chain is.

Smoothing

- ▶ Smoothing estimates past states by computing $P(X_k | e_{1:t})$ for $0 \leq k < t$



- ▶ Divide evidence $e_{1:t}$ into $e_{1:k}$ (before k) and $e_{k+1:t}$ (after k):

$$\begin{aligned} P(X_k | e_{1:t}) &= P(X_k | e_{1:k}, e_{k+1:t}) \\ &= \alpha \cdot P(X_k | e_{1:k}) \cdot P(e_{k+1:t} | X_k, e_{1:k}) \quad (\text{Bayes Rule}) \\ &= \alpha \cdot P(X_k | e_{1:k}) \cdot P(e_{k+1:t} | X_k) \quad (\text{cond. independence}) \\ &= \alpha \cdot f_{1:k} \cdot b_{k+1:t} \end{aligned}$$

Smoothing (continued)

- ▶ Backward message $b_{k+1:t} = P(e_{k+1:t}|X_k)$ computed by a backwards recursion:

$$\begin{aligned}P(e_{k+1:t}|X_k) &= \sum_{x_{k+1}} P(e_{k+1:t}|X_k, x_{k+1}) \cdot P(x_{k+1}|X_k) \\&= \sum_{x_{k+1}} P(e_{k+1:t}|x_{k+1}) \cdot P(x_{k+1}|X_k) \\&= \sum_{x_{k+1}} P(e_{k+1}, e_{k+2:t}|x_{k+1}) \cdot P(x_{k+1}|X_k) \\&= \sum_{x_{k+1}} P(e_{k+1}|x_{k+1}) \cdot P(e_{k+2:t}|x_{k+1}) \cdot P(x_{k+1}|X_k)\end{aligned}$$

$P(e_{k+1}|x_{k+1})$ and $P(x_{k+1}|X_k)$ can be directly obtained from the model,
 $P(e_{k+2:t}|x_{k+1})$ is the “recursive call” ($b_{k+2:t}$).

- ▶ In message notation: $b_{k+1:t} = \text{BACKWARD}(b_{k+2:t}, e_{k+1:t})$ where **BACKWARD** is the update shown above. (time and space constant (independent of t))

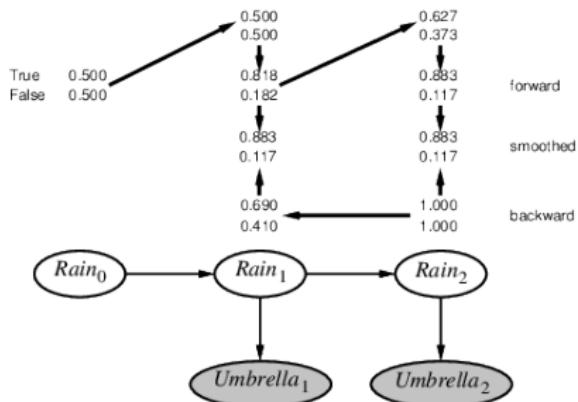
Smoothing example

- ▶ **Example 2.9 (Smoothing Umbrellas).** Umbrella appears on days 1/2.

- ▶ $P(R_1|u_1, u_2) = \alpha \cdot P(R_1|u_1) \cdot P(u_2|R_1) = \alpha \cdot \langle 0.818, 0.182 \rangle \cdot P(u_2|R_1)$
- ▶ Compute $P(u_2|R_1)$ by backwards recursion:

$$\begin{aligned} P(u_2|R_1) &= \sum_{r_2} P(u_2|r_2) \cdot P(r_2|R_1) \\ &= 0.9 \cdot 1 \cdot \langle 0.7, 0.3 \rangle + 0.2 \cdot 1 \cdot \langle 0.3, 0.7 \rangle = \langle 0.69, 0.41 \rangle \end{aligned}$$

- ▶ So $P(R_1|u_1, u_2) = \alpha \cdot \langle 0.818, 0.182 \rangle \cdot \langle 0.69, 0.41 \rangle \approx 0.883, 0.117$



Smoothing gives a higher probability for rain on day 1

- ▶ umbrella on day 2
- ▶ ↗ rain more likely on day 2
- ▶ ↗ rain more likely on day 1.

Forward/Backward Algorithm for Smoothing

- ▶ **Definition 2.10.** **Forward backward algorithm:** cache forward messages along the way:

```
function Forward–Backward (ev,prior)
    returns: a vector of probability distributions
    inputs: ev, a vector of evidence evidence values for steps 1, . . . , t
            prior, the prior distribution on the initial state,  $P(X_0)$ 
    local: fv, a vector of forward messages for steps 0, . . . , t
           b, a representation of the backward message, initially all 1s
           sv, a vector of smoothed estimates for steps 1, . . . , t
    fv[0] := prior
    for i = 1 to t do
        fv[i] := FORWARD(fv[i - 1], ev[i])
    for i = t downto 1 do
        sv[i] := NORMALIZE(fv[i]b)
        b := BACKWARD(b, ev[i])
    return sv
```

- ▶ Time complexity linear in t (polytree inference), Space complexity $\mathcal{O}(t \cdot \#(f))$.

- ▶ **Observation 2.11.** Most likely sequence \neq sequence of most likely states!
- ▶ **Example 2.12.** Suppose the umbrella sequence is T, T, F, T, T what is the most likely weather sequence?
- ▶ **Prominent Application:** In speech recognition, we want to find the most likely word sequence, given what we have heard. (can be quite noisy)
- ▶ **Idea:** Use smoothing to find posterior distribution in each time step, construct sequence of most likely states.
- ▶ **Problem:** These posterior distributions range over a single time step. (and this difference matters)

Most Likely Explanation (continued)

- Most likely path to each x_{t+1} = most likely path to *some* x_t plus one more step

$$\begin{aligned} & \max_{x_1, \dots, x_t} (P(x_1, \dots, x_t, X_{t+1} | e_{1:t+1})) \\ &= P(e_{t+1} | X_{t+1}) \cdot \max_{x_t} (P(X_{t+1} | x_t) \cdot \max_{x_1, \dots, x_{t-1}} (P(x_1, \dots, x_{t-1}, x_t | e_{1:t}))) \end{aligned}$$

- Identical to filtering, except $f_{1:t}$ replaced by

$$m_{1:t} = \max_{x_1, \dots, x_{t-1}} (P(x_1, \dots, x_{t-1}, X_t | e_{1:t}))$$

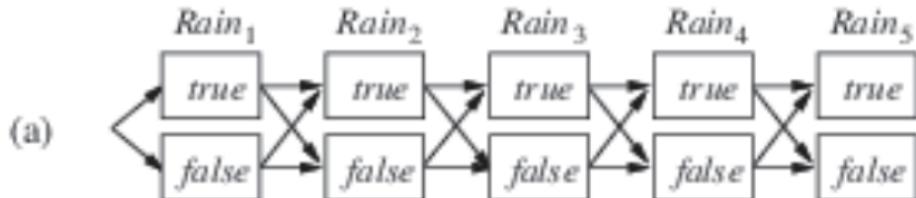
i.e., $m_{1:t}(i)$ gives the probability of the most likely path to state i .
Update has sum replaced by max, giving the **Viterbi algorithm**:

$$m_{1:t+1} = P(e_{t+1} | X_{t+1}) \cdot \max_{x_t} (P(X_{t+1} | x_t, m_{1:t}))$$

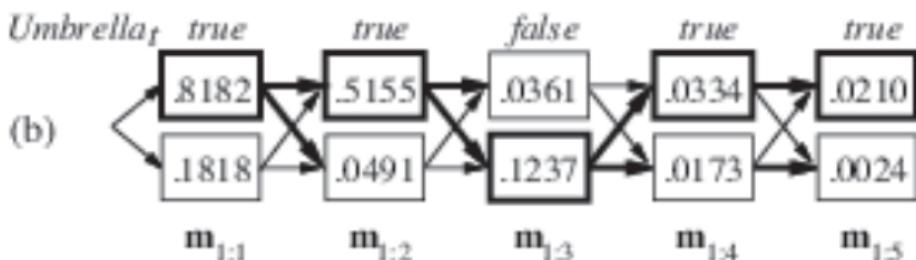
- **Observation 2.13.** Viterbi has linear time complexity (like filtering), but linear space complexity (needs to keep a pointer to most likely sequence leading to each state).

Viterbi example

- **Example 2.14 (Viterbi for Umbrellas).** View the possible state sequences for Rain_t as paths through state graph.



Operation of the Viterbi algorithm for the sequence [T, T, F, T, T]:



- values are $m_{1:t}$ (probability of best sequence reaching state at time t)
- bold arrows: best predecessor measured by “best preceding sequence probability \times transition probability”

To find “most likely sequence”, follow bold arrows back from “most likely state $m_{1:5}$.

3 Hidden Markov Models

Hidden Markov Models

- ▶ **Definition 3.1.** A **hidden Markov model (HMM)** is a **Markov chain** with a single, discrete state variable X_t with domain $\{1, \dots, S\}$ and a single, discrete evidence variable.
- ▶ **Example 3.2.** 1.5(the umbrella example) is a **HMM**.
- ▶ **Observation:** Transition model $P(X_t|X_{t-1}) \hat{=} \text{a single } S \times S \text{ matrix.}$
- ▶ **Definition 3.3.** **Transition matrix** $T_{ij} = P(X_t = j|X_{t-1} = i)$
- ▶ **Example 3.4 (Umbrellas).** $T = P(X_t|X_{t-1}) = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}.$
- ▶ **Observation:** Sensor model $P(e_t|X_t = i) \hat{=} \text{an } S\text{-vector.}$
- ▶ **Idea:** Re-cast **Markov inference** as matrix calculations.
This works best, if we make the **sensor model** into a **diagonal matrix**. (see below)
- ▶ **Definition 3.5.** **Sensor matrix** O_t for each time step $\hat{=} \text{diagonal matrix with } O_{tii} = P(e_t|X_t = i).$
- ▶ **Example 3.6 (Umbrellas).** With $U_1 = T$ and $U_3 = F$ we have

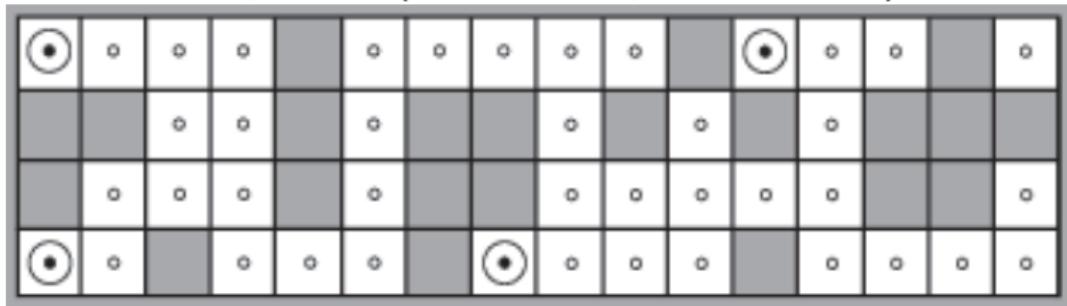
$$O_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix} \quad \text{and} \quad O_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix}$$

- ▶ Idea: Forward and backward messages are column vectors in HMMs
- ▶ Definition 3.7. Recasting the Markov inference as matrix computation, gives us two identities:
 - HMM filtering equation:** $f_{1:t+1} = \alpha \cdot O_{t+1} T^t f_{1:t}$
 - HMM smoothing equation:** $b_{k+1:t} = T O_{k+1} b_{k+2:t}$
- ▶ Observation 3.8. The forward backward algorithm for HMMs has time complexity $\mathcal{O}(S^2 t)$ and space complexity $\mathcal{O}(St)$.

Example: Robot Localization using Common Sense

- ▶ **Example 3.9 (Robot Localization in a Maze).** Robot has four sonar sensors that tell it about obstacles in four directions: N, S, W, E.
- ▶ **Notation:** We write the result where the sensor that detects obstacles in the north, south, and east as **NSE**.

- ▶ **Example 3.10 (Filter out Impossible States).**



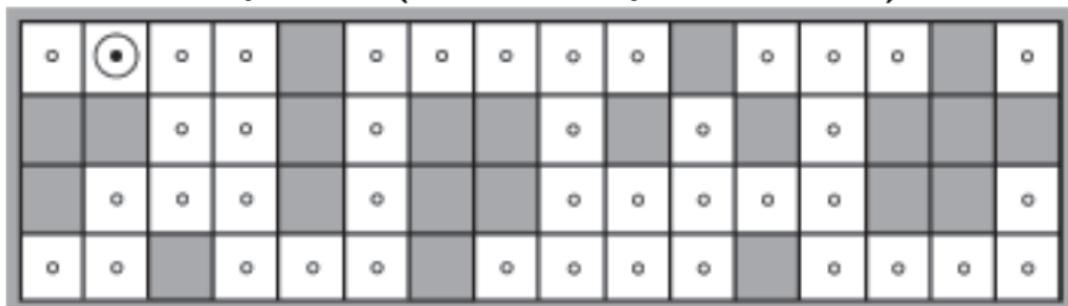
a) Possible robot locations after $e_1 = \text{NSE}$

- ▶ **Remark 3.11.** This only works for perfect sensors. (else no impossible states)

Example: Robot Localization using Common Sense

- ▶ **Example 3.9 (Robot Localization in a Maze).** Robot has four sonar sensors that tell it about obstacles in four directions: N, S, W, E.
- ▶ **Notation:** We write the result where the sensor that detects obstacles in the north, south, and east as **NSE**.

- ▶ **Example 3.10 (Filter out Impossible States).**



b) Possible robot locations after $e_1 = \text{NSE}$ and $e_2 = \text{NS}$

- ▶ **Remark 3.11.** This only works for perfect sensors. (else no impossible states)

HMM Example: Robot Localization (Modeling)

► Example 3.12 (HMM-based Robot Localization).

- Random variable X_t for robot location (domain: 42 empty squares)
- Transition matrix for the move action: (T has $42^2 = 1764$ entries)

$$P(X_{t+1} = j | X_t = i) = T_{ij} = \begin{cases} \frac{1}{\#(N(i))} & \text{if } j \in N(i) \\ 0 & \text{else} \end{cases}$$

where $N(i)$ is the set of neighboring fields of state i .

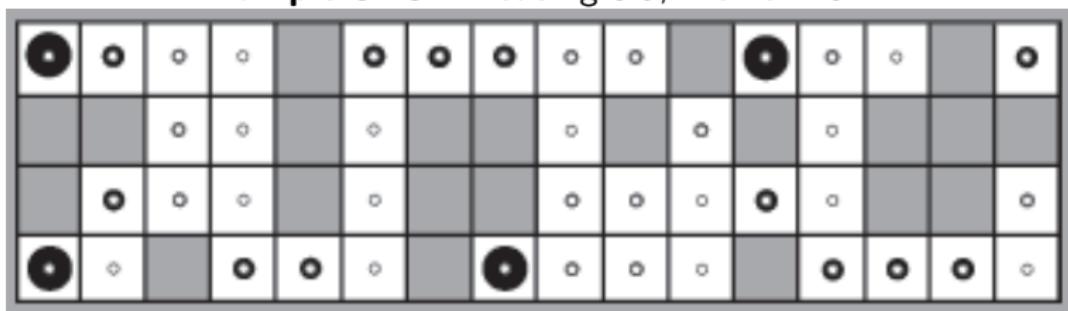
- We do not know where the robot starts: $P(X_0) = \frac{1}{n}$ (here $n = 42$)
- Evidence variable E_t : four-bit presence/absence of obstacles in N, S, W, E. Let d_{it} be the number of wrong bits and ϵ the error rate of the sensor.

$$P(E_t = e_t | X_t = i) = O_{tii} = (1 - \epsilon)^{4-d_{it}} \cdot \epsilon^{d_{it}}$$

- For instance, the probability that the sensor on a square with obstacles in north and south would produce NSE is $(1 - \epsilon)^3 \cdot \epsilon^1$.
- Idea: Use the HMM filtering equation $f_{1:t+1} = \alpha \cdot O_{t+1} T^t f_{1:t}$ for localization.
(next)

HMM Example: Robot Localization

- Idea: Use HMM filtering equation $f_{1:t+1} = \alpha \cdot O_{t+1} T^t f_{1:t}$ to compute posterior distribution over locations. (i.e. robot localization)
 - Example 3.13. Redoing 3.9, with $\epsilon = 0.2$.



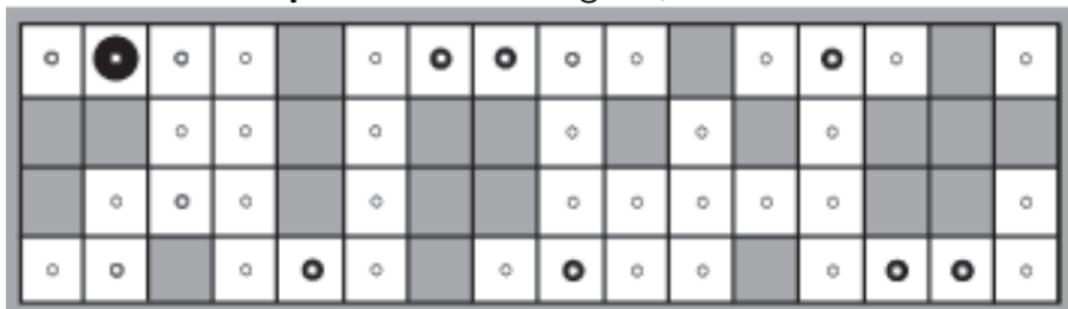
a) Posterior distribution over robot location after $E_1 = \text{NSW}$

Still the same locations as in the “perfect sensing” case, but now other locations have non-zero probability.

HMM Example: Robot Localization

- Idea: Use HMM filtering equation $f_{1:t+1} = \alpha \cdot O_{t+1} T^t f_{1:t}$ to compute posterior distribution over locations. (i.e. robot localization)

► Example 3.13. Redoing 3.9, with $\epsilon = 0.2$.

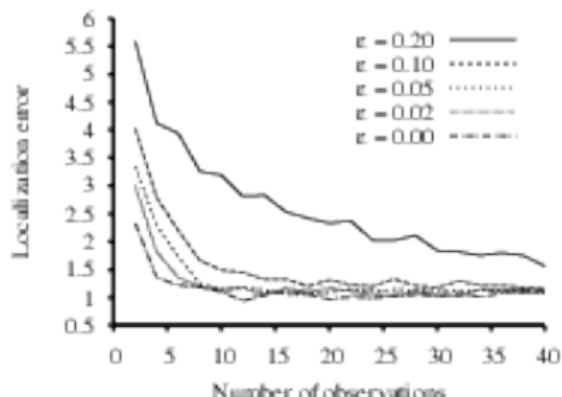


b) Posterior distribution over robot location after $E_1 = \text{NSW}$ and $E_2 = \text{NS}$

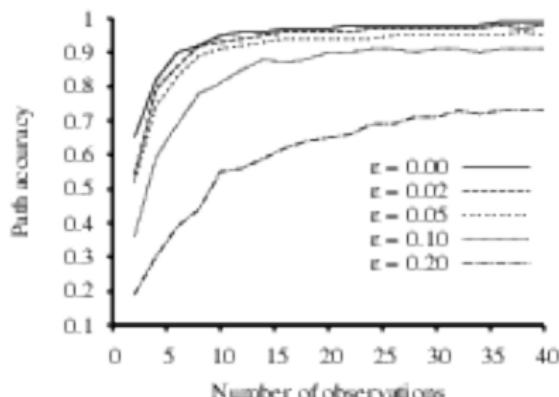
Still the same locations as in the “perfect sensing” case, but now other locations have non-zero probability.

HMM Example: Further Inference Applications

- ▶ Idea: Use smoothing: $b_{k+1:t} = \text{TO}_{k+1} b_{k+2:t}$ to find out where it started and the Viterbi algorithm to find the most likely path it took.
- ▶ **Example 3.14.** Performance of HMM localization vs. observation length (various error rates ϵ)



Localization error (Manhattan distance from true location)



Viterbi path accuracy (fraction of correct states on Viterbi path)

Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

Country dance algorithm

- **Idea:** We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned}\mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t}\end{aligned}$$

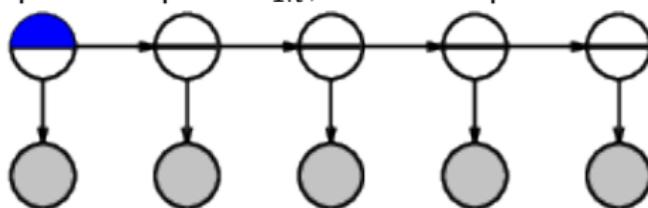
- **Algorithm:** Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

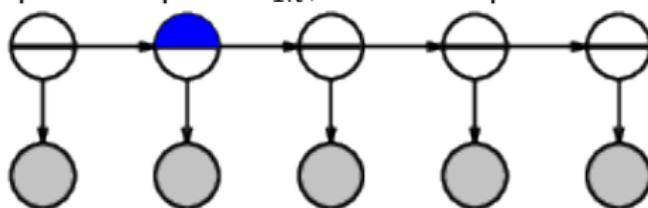


Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

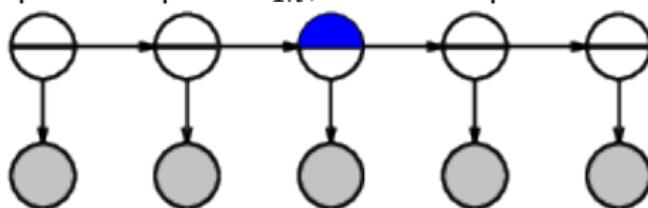


Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

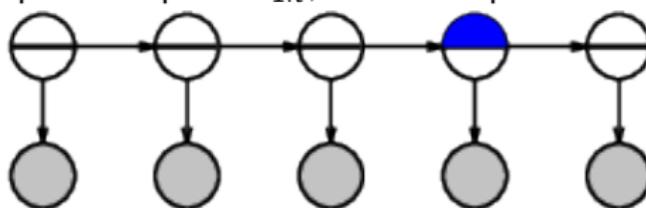


Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

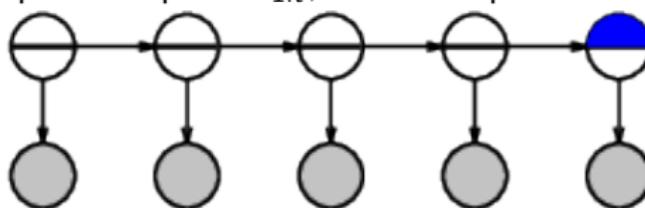


Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

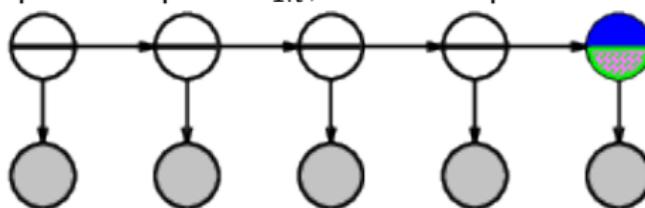


Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

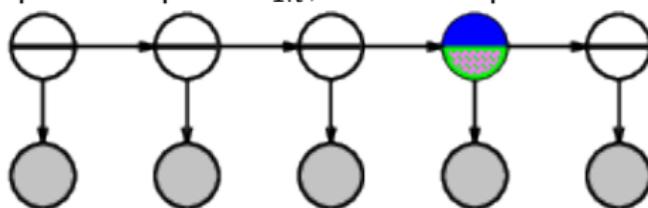


Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

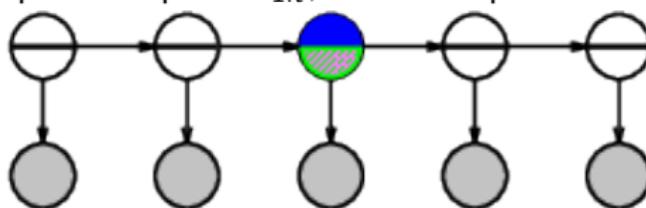


Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

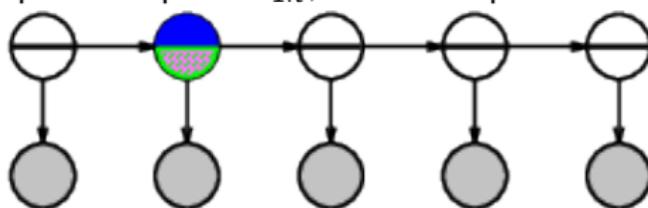


Country dance algorithm

- Idea: We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- Algorithm: Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.

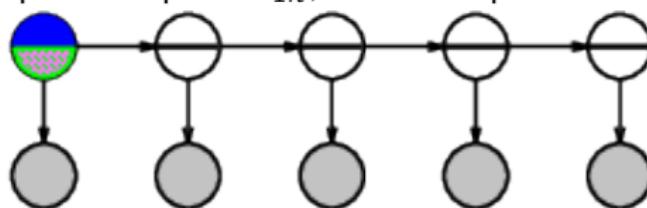


Country dance algorithm

- **Idea:** We can avoid storing all forward messages in **smoothing** by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot \mathbf{T}^t \mathbf{f}_{1:t} \\ \alpha \cdot \mathbf{T}'^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

- **Algorithm:** Forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}$, $\mathbf{b}_{t-i:t}$.



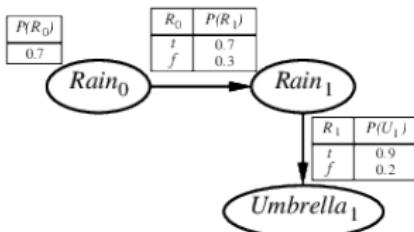
- **Observation:** Backwards pass only stores one copy of $\mathbf{f}_{1:i}$, $\mathbf{b}_{t:t-i} \rightsquigarrow$ constant space.
- **Problem:** Algorithm is severely limited: transition matrix must be invertible and sensor matrix cannot have zeroes – that is, that every observation be possible in every state.

4 Dynamic Bayesian Networks

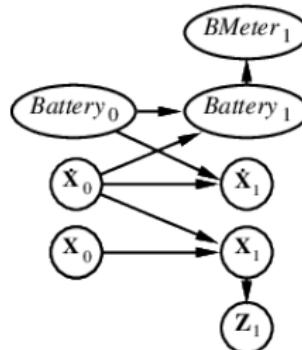
Dynamic Bayesian networks

- ▶ **Definition 4.1.** A Bayesian network \mathcal{D} is called **dynamic** (a **DBN**), iff its random variables are indexed by a **time structure**. We assume that \mathcal{D} is
 - ▶ **time sliced**, i.e. that the **time slices** \mathcal{D}_t – the subgraphs of t -indexed random variables and the edges between them – are **isomorphic**.
 - ▶ a **Markov chain**, i.e. that variables X_t can only have **parents** in \mathcal{D}_t and \mathcal{D}_{t-1} .
- ▶ X_t, E_t contain arbitrarily many variables in a replicated Bayesian network.
- ▶ **Example 4.2.**

Umbrellas

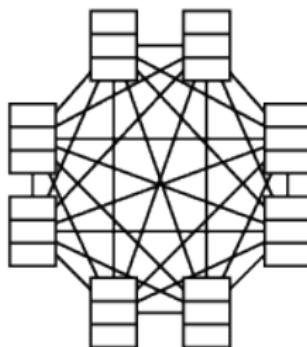
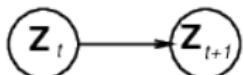
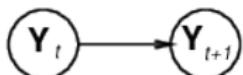
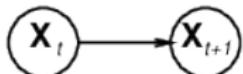


Robot Motion



► ***Observation 4.3.***

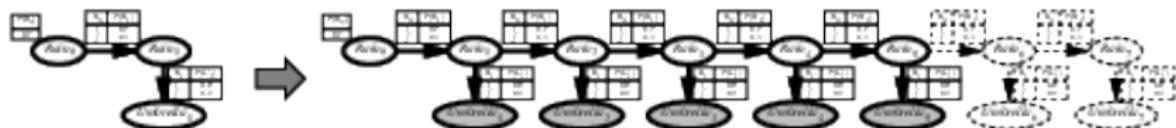
- Every HMM is a single-variable DBN. (trivially)
- Every discrete DBN is an HMM. (combine variables into tuple)
- DBNs have sparse dependencies \sim exponentially fewer parameters;



► ***Example 4.4 (Sparse Dependencies).***

With 20 Boolean state variables, three parents each, a DBN has $20 \cdot 2^3 = 160$ parameters, the corresponding HMM has $2^{20} \cdot 2^{20} \approx 10^{12}$.

- ▶ **Definition 4.5 (Naive method).** **Unroll** the network and run any exact algorithm.



- ▶ **Problem:** Inference cost for each update grows with t .
- ▶ **Definition 4.6. Rollup filtering:** add slice $t + 1$, “sum out” slice t using **variable elimination**.
- ▶ Largest factor is $\mathcal{O}(d^{n+1})$, update cost $\mathcal{O}(d^{n+2})$. (cf. HMM update cost $\mathcal{O}(d^{(2n)})$)

- ▶ Temporal probability models use state and evidence variables replicated over time.
- ▶ Markov property and stationarity assumption, so we need both
 - ▶ a transition model and $P(X_t|X_{t-1})$
 - ▶ a sensor model $P(E_t|X_t)$.
- ▶ Tasks are filtering, prediction, smoothing, most likely sequence; (all done recursively with constant cost per time step)
- ▶ Hidden Markov models have a single discrete state variable; (used for speech recognition)
- ▶ DBNs subsume HMMs, exact update intractable.
- ▶ Particle filtering is a good approximate filtering algorithm for DBNs.

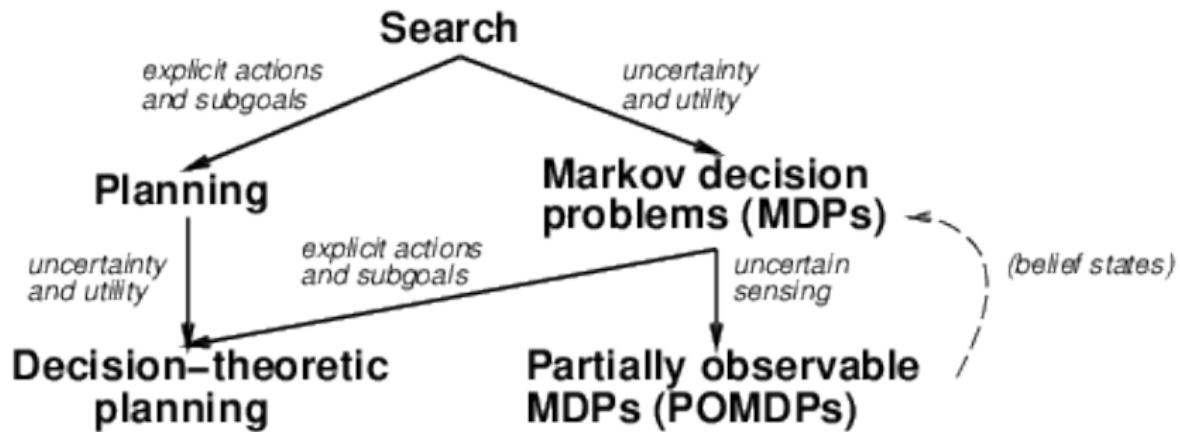
Chapter 24 Making Complex Decisions

- ▶ Markov decision processes (MDPs) for sequential environments.
- ▶ Value/policy iteration for computing utilities in MDPs.
- ▶ Partially observable MDP (POMDPs).
- ▶ Decision-theoretic agents for POMDPs.

1 Sequential Decision Problems

Sequential Decision Problems

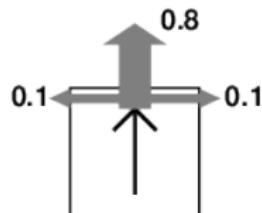
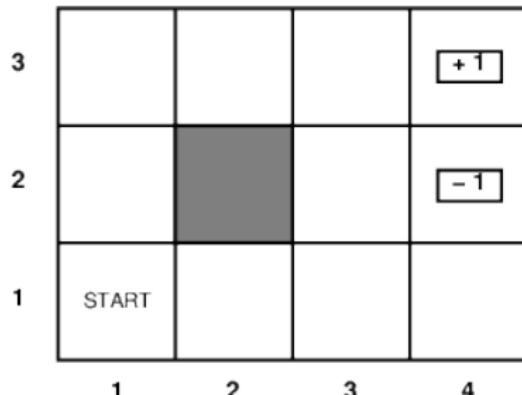
- ▶ **Definition 1.1.** In **sequential decision problems**, the **agent's utility** depends on a sequence of decisions.
- ▶ Sequential decision problems incorporate **utilities**, uncertainty, and sensing.
- ▶ Observation: Search problems and **planning tasks** are special cases.



Markov Decision Problem: Running Example

► Example 1.2 (Running Example: The 4x3 World).

A (fully observable) 4×3 environment with non-deterministic actions:



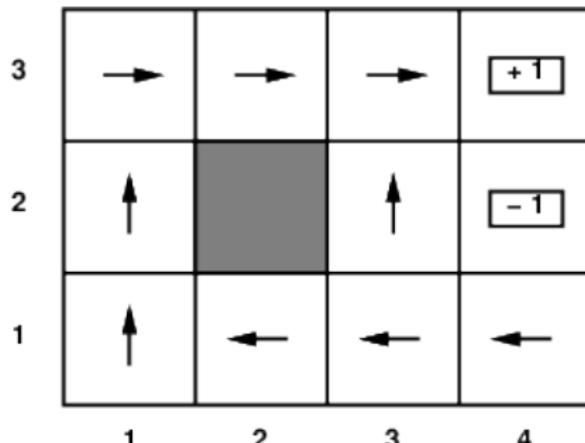
- States $s \in S$, actions $a \in A$.
- Transition model: $P(s'|s, a) \hat{=} \text{probability that } a \text{ in } s \text{ leads to } s'$.
- Reward function:

$$R(s, a, s') := \begin{cases} -0.04 & \text{if (small penalty) for nonterminal states} \\ \pm 1 & \text{if for terminal states} \end{cases}$$

- ▶ **Definition 1.3.** A sequential decision problem in a **fully observable, stochastic environment** with a **Markovian transition model** and an additive **reward function** is called a **Markov decision process (MDP)**. It consists of
 - ▶ a set of S of **states** (with initial state $s_0 \in S$),
 - ▶ sets **Actions(s)** of **actions** for each **state s** .
 - ▶ a **transition model** $P(s'|s, a)$, and
 - ▶ a **reward function** $R: S \rightarrow \mathbb{R}$ – we call $R(s)$ a **reward**.

Solving MDPs

- ▶ Recall: In search problems, the aim is to find an optimal sequence of actions.
- ▶ In MDPs, the aim is to find an optimal policy $\pi(s)$ i.e., best action for every possible state s . (because can't predict where one will end up)
- ▶ Definition 1.4. In an MDP, a policy is a mapping from states to actions. An optimal policy maximizes (say) the expected sum of rewards. (MEU)
- ▶ Example 1.5. Optimal policy when state penalty $R(s)$ is -0.04 :



Note: When you run against a wall, you stay in your square.

2 Utilities over Time

- ▶ Problem: We need to understand preferences between sequences of states.
- ▶ **Definition 2.1.** We call preferences on reward sequences stationary, iff

$$[r, r_0, r_1, r_2, \dots] \succ [r, r'_0, r'_1, r'_2, \dots] \Leftrightarrow [r_0, r_1, r_2, \dots] \succ [r'_0, r'_1, r'_2, \dots]$$

- ▶ **Theorem 2.2.** For stationary preferences, there are only two ways to combine rewards over time.
 - ▶ **additive rewards:** $U([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$
 - ▶ **discounted rewards:** $U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$ where γ is called **discount factor**.

Utilities of State Sequences

► Problem: Infinite lifetimes \leadsto additive utilities become infinite.

► Possible Solutions:

1. Finite horizon: terminate utility computation at a fixed time T

$$U([s_0, \dots, s_\infty]) = R(s_0) + \dots + R(s_T)$$

\leadsto nonstationary policy: $\pi(s)$ depends on time left.

2. If there are absorbing states: for any policy π agent eventually “dies” with probability 1 \leadsto expected utility of every state is finite.
3. Discounting: assuming $\gamma < 1$, $R(s) \leq R_{\max}$,

$$U([s_0, \dots, s_\infty]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max}/(1 - \gamma)$$

Smaller $\gamma \leadsto$ shorter horizon.

► Idea: Maximize system gain $\hat{=} \text{average reward per time step.}$

► Theorem 2.3. The optimal policy has constant gain after initial transient.

► Example 2.4. Taxi driver's daily scheme cruising for passengers.

Utility of States

- ▶ Intuition: Utility of a state $\hat{=}$ *expected (discounted) sum of rewards (until termination) assuming optimal actions.*
- ▶ **Definition 2.5.** Given a policy π , let s_t be the state the agent reaches at time t starting at state s_0 . Then the **expected utility** obtained by executing π starting in s is given by

$$U^\pi(s) := E \left[\left(\sum_{t=0}^{\infty} \gamma^t R(s_t) \right) \right]$$

we define $\pi^* := \underset{\pi}{\operatorname{argmax}} U^\pi(s)$.

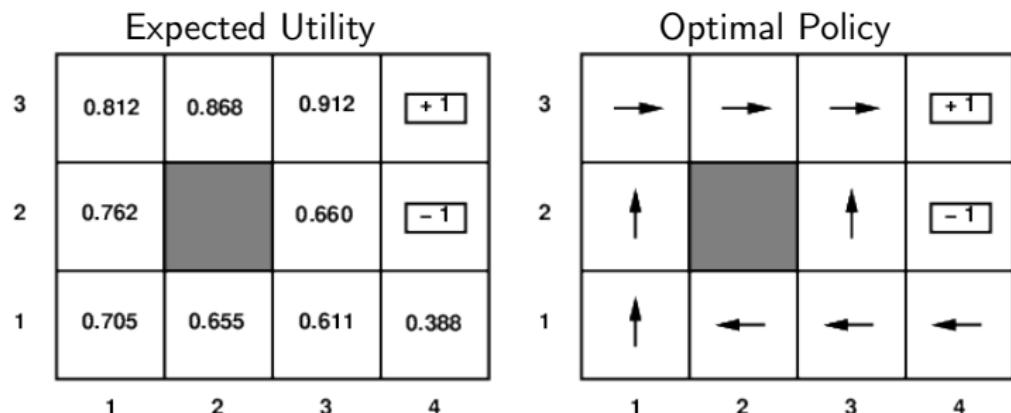
- ▶ **Observation 2.6.** π^* is independent from s .
- ▶ **Proof Sketch:** If π^*_a and π^*_b reach point c , then there is no reason to disagree – or with π^*_c □
- ▶ **Definition 2.7.** We call $\pi^* := \pi^*_s$ for some s the **optimal policy**.
- ▶ ⚠: does not hold for finite-horizon policies.
- ▶ **Definition 2.8.** The **utility** $U(s)$ of a state s is $U^{\pi^*}(s)$.

Utility of States (continued)

- ▶ Remark: $R(s) \hat{=} \text{"short-term reward"}$, whereas $U \hat{=} \text{"long-term reward"}$.
- ▶ Given the utilities of the states, choosing the best action is just **MEU**:
 - ▶ maximize the **expected utility** of the immediate successors

$$\pi^*(s) = \underset{(a \in A(s))}{\operatorname{argmax}} \left(\sum_{s'} P(s'|s, a) \cdot U(s') \right)$$

- ▶ Example 2.9 (Running Example Continued).



- ▶ Question: Why do we go left in (3, 1) and not up?

(follow the utility)

3 Value/Policy Iteration

Dynamic programming: the Bellman equation

- ▶ 2.8 leads to a simple relationship among utilities of neighboring states:
expected sum of rewards = current reward + $\gamma \cdot$ exp. reward sum after best action

- ▶ **Theorem 3.1 (Bellman equation (1957)).**

$$U(s) = R(s) + \gamma \cdot \max_{(a \in A(s))} \left(\sum_{s'} U(s') \cdot P(s'|s, a) \right)$$

We call this equation the **Bellman equation**

- ▶ **Example 3.2.** $U(1, 1) = -0.04$
 $+ \gamma \max\{0.8U(1, 2) + 0.1U(2, 1) + 0.1U(1, 1),$ up
 $0.9U(1, 1) + 0.1U(1, 2)$ left
 $0.9U(1, 1) + 0.1U(2, 1)$ down
 $0.8U(2, 1) + 0.1U(1, 2) + 0.1U(1, 1)\}$ right
- ▶ **Problem:** One equation/state $\sim n$ nonlinear (\max isn't) equations in n unknowns.
 \sim cannot use linear algebra techniques for solving them.

Value Iteration Algorithm

► Idea: We use a simple iteration scheme to find a fixpoint:

1. start with arbitrary utility values,
2. update to make them locally consistent with the Bellman equation,
3. everywhere locally consistent \leadsto global optimality.

► Definition 3.3. The **value iteration algorithm** for **utility functions** is given by

function VALUE-ITERATION (mdp, ϵ) **returns** a utility fn.

inputs: mdp , an MDP with states S , actions $A(s)$, transition model $P(s'|s, a)$, rewards $R(s)$, and discount γ

ϵ , the maximum error allowed **in** the utility of any state

local variables: U, U' , vectors of utilities **for** states **in** S , initially zero

δ , the maximum change **in** the utility of any state **in** an iteration

repeat

$U := U'; \delta := 0$

for each state s **in** S **do**

$U'[s] := R(s) + \gamma \cdot \max_a (\sum_{s'} U[s'] \cdot P(s'|s, a))$

if $|U'[s] - U[s]| > \delta$ **then** $\delta := |U'[s] - U[s]|$

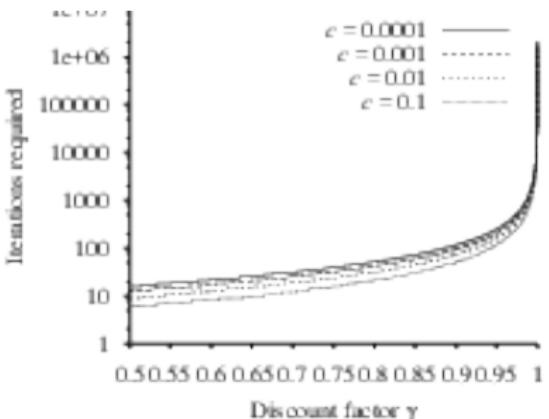
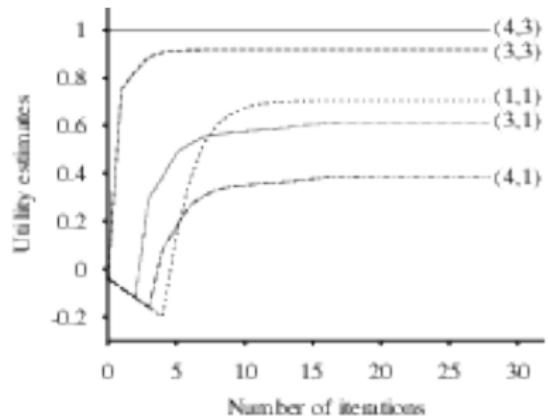
until $\delta < \epsilon(1 - \gamma)/\gamma$

return U

► Remark: Retrieve the optimal policy with $\pi[s] := \operatorname{argmax}_a (\sum_{s'} U[s'] \cdot P(s'|s, a))$

Value Iteration Algorithm (Example)

► Example 3.4 (Iteration on 4x3).



- ▶ **Definition 3.5.** The **maximum norm** $\|U\| = \max_s |\textcolor{blue}{U}(s)|$, so $\|U - V\| =$ maximum difference between U and V .
- ▶ Let U^t and U^{t+1} be successive approximations to the true utility U .
- ▶ **Theorem 3.6.** For any two approximations U^t and V^t

$$\|U^{t+1} - V^{t+1}\| \leq \gamma \|U^t - V^t\|$$

i.e., any distinct approximations must get closer to each other
so, in particular, any approximation must get closer to the true U
and **value iteration** converges to a unique, stable, optimal solution.

- ▶ **Theorem 3.7.** If $\|U^{t+1} - U^t\| < \epsilon$, then $\|U^{t+1} - U\| < 2\epsilon\gamma/1 - \gamma$
i.e., once the change in U^t becomes small, we are almost done.
- ▶ **MEU policy** using U^t may be optimal long before convergence of values.

- ▶ Recap: Value iteration computes utilities \rightsquigarrow optimal policy by MEU.
- ▶ This even works if the utility estimate is inaccurate. (\rightsquigarrow policy loss small)
- ▶ Idea: search for optimal policy and utility values simultaneously [How60]: Iterate
 - ▶ **policy evaluation**: given policy π_i , calculate $U_i = U^{\pi_i}$, the utility of each state were π_i to be executed.
 - ▶ **policy improvement**: calculate a new MEU policy π_{i+1} using 1-lookahead
Terminate if policy improvement yields no change in utilities.
- ▶ **Observation 3.8.** Upon termination U_i is a fixed point of Bellman update
 \rightsquigarrow Solution to Bellman equation $\rightsquigarrow \pi_i$ is an optimal policy.
- ▶ **Observation 3.9.** Policy improvement improves policy and policy space is finite \rightsquigarrow termination.

Policy Iteration Algorithm

- **Definition 3.10.** The **policy iteration algorithm** is given by the following pseudocode:

```
function POLICY-ITERATION(mdp) returns a policy
    inputs: mdp, and MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s'|s, a)$ 
    local variables:  $U$  a vector of utilities for states in  $S$ , initially zero
                     $\pi$  a policy indexed by state, initially random,
    repeat
         $U := \text{POLICY-EVALUATION}(\pi, U, mdp)$ 
        unchanged? := true
        foreach state  $s$  in  $X$  do
            if  $\max_{a \in A(s)} (\sum_{s'} P(s'|s, a) \cdot U(s')) > \sum_{s'} P(s'|s, \pi[s']) \cdot U(s')$  then do
                 $\pi[s] := \operatorname{argmax}_{(b \in A(s))} (\sum_{s'} P(s'|s, b) \cdot U(s'))$ 
                unchanged? := false
        until unchanged?
    return  $\pi$ 
```

- ▶ **Problem:** How to implement the POLICY-EVALUATION algorithm?
- ▶ **Solution:** To compute utilities given a fixed π : For all s we have

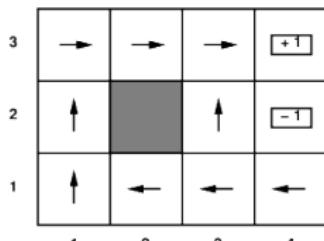
$$U(s) = R(s) + \gamma \left(\sum_{s'} U(s') \cdot P(s'|s, \pi(s)) \right)$$

- ▶ **Example 3.11 (Simplified Bellman Equations for π).**

$$U_i(1,1) = -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)$$

$$U_i(1,2) = -0.04 + 0.8U_i(1,3) + 0.1U_i(1,2)$$

⋮



- ▶ **Observation 3.12.** n simultaneous linear equations in n unknowns, solve in $\mathcal{O}(n^3)$ with standard linear algebra methods.

- ▶ Policy iteration often converges in few iterations, but each is expensive.
- ▶ Idea: Use a few steps of value iteration (but with π fixed) starting from the value function produced the last time to produce an approximate value determination step.
- ▶ Often converges much faster than pure VI or PI.
- ▶ Leads to much more general algorithms where Bellman value updates and Howard policy updates can be performed locally in any order.
- ▶ Reinforcement learning algorithms operate by performing such updates based on the observed transitions made in an initially unknown environment.

4 Partially Observable MDPs

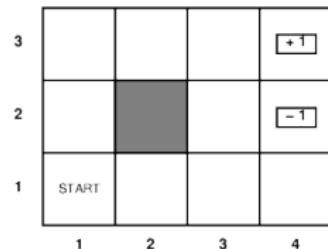
Partial Observability

- **Definition 4.1.** A **partially observable MDP** (a **POMDP** for short) is a **MDP** together with an **sensor model O** that has the **sensor Markov property** and is **stationary**: $O(s, e) = P(e|s)$.

Example 4.2 (Noisy 4x3 World).

Add a partial and/or noisy sensor.

e.g. count number of adjacent walls ($1 \leq w \leq 2$) with 0.1 error (noise) If sensor reports 1, we are in (3,?) (probably)



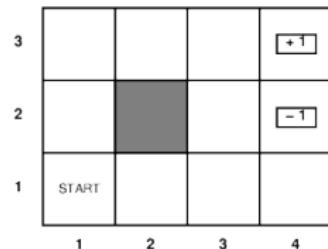
Partial Observability

- ▶ **Definition 4.1.** A **partially observable MDP** (a **POMDP** for short) is a **MDP** together with an **sensor model O** that has the **sensor Markov property** and is **stationary**: $O(s, e) = P(e|s)$.

Example 4.2 (Noisy 4x3 World).

Add a partial and/or noisy sensor.

e.g. count number of adjacent walls ($1 \leq w \leq 2$) with 0.1 error (noise) If sensor reports 1, we are in (3,?) (probably)



- ▶
- ▶ **Problem:** Agent does not know which state it is in \leadsto makes no sense to talk about **policy $\pi(s)$** !

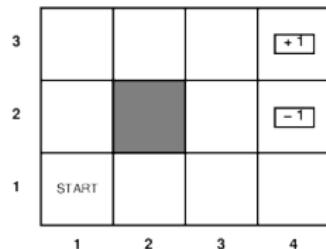
Partial Observability

- ▶ **Definition 4.1.** A **partially observable MDP** (a **POMDP** for short) is a **MDP** together with an **sensor model O** that has the **sensor Markov property** and is **stationary**: $O(s, e) = P(e|s)$.

Example 4.2 (Noisy 4x3 World).

Add a partial and/or noisy sensor.

e.g. count number of adjacent walls ($1 \leq w \leq 2$) with 0.1 error (noise) If sensor reports 1, we are in $(3, ?)$ (probably)



- ▶
- ▶ **Problem:** Agent does not know which state it is in \leadsto makes no sense to talk about **policy $\pi(s)$** !
- ▶ **Theorem 4.3 (Astrom 1965).** The **optimal policy** in a **POMDP** is a function $\pi(b)$ where b is the **belief state** (probability distribution over states).
- ▶ **Idea:** Convert a **POMDP** into an **MDP** in **belief state** space, where $T(b, a, b')$ is the probability that the new **belief state** is b' given that the current **belief state** is b and the **agent** does a . I.e., essentially a filtering update step

POMDP: Filtering at the Belief State Level

- ▶ Recap: Filtering updates the belief state for new evidence.
- ▶ For POMDPs, we also need to consider actions. (but the effect is the same)
- ▶ If $b(s)$ is the previous belief state and agent does action a and then perceives e , then the new belief state is

$$b'(s') = \alpha \cdot P(e|s') \cdot \left(\sum_s P(s'|s, a) \cdot b(s) \right)$$

We write $b' = \text{FORWARD}(b, a, e)$ in analogy to recursive state estimation.

- ▶ Fundamental Insight for POMDPs: The optimal action only depends on the agent's current belief state. (good, it does not know the state!)
- ▶ Consequence: The optimal policy can be written as a function $\pi^*(b)$ from belief states to actions.
- ▶ Definition 4.4. The POMDP decision cycle is to iterate over
 1. Given the current belief state b , execute the action $a = \pi^*(b)$
 2. Receive percept e .
 3. Set the current belief state to $\text{FORWARD}(b, a, e)$ and repeat.
- ▶ Intuition: POMDP decision cycle is search in belief state space.

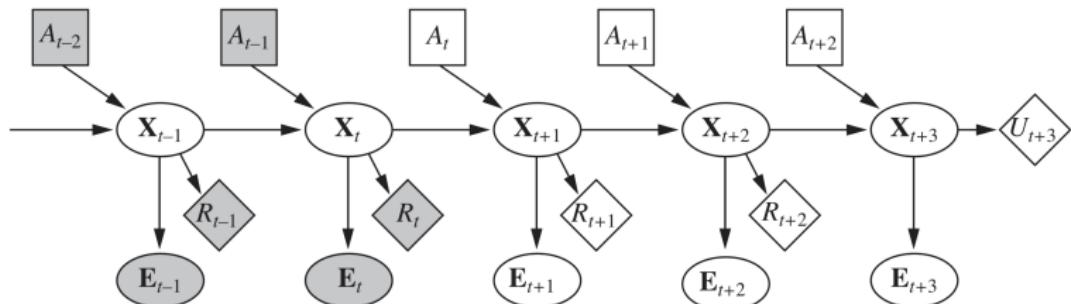
- ▶ Recap: The POMDP decision cycle is search in belief state space.
- ▶ **Observation 4.5.** Actions change the belief state, not just the physical state.
- ▶ Thus POMDP solutions automatically include information-gathering behavior.
- ▶ Problem: The belief state is continuous: If there are n states, b is an n -dimensional real-valued vector.
- ▶ **Example 4.6.** The belief state of the 4x3 world is a 11-dimensional continuous space (11 states)
- ▶ **Theorem 4.7.** Solving POMDPs is very hard! (actually, PSPACE-hard)
- ▶ In particular, none of the algorithms we have learned applies. (discreteness assumption)
- ▶ The real world is a POMDP (with initially unknown transition model T and sensor model O)

5 Online Agents with POMDPs

Designing Online Agents for POMDPs

► Definition 5.1 (Dynamic Decision Networks).

- Transition and sensor model are represented as a DBN (a dynamic Bayesian network).
- action nodes and utility nodes are added to create a **dynamic decision network (DDN)**.
- a filtering algorithm is used to incorporate each new percept and action and to update the **belief state** representation.
- decisions are made by projecting forward possible action sequences and choosing the best one.
- Generic structure of a **dynamic decision network** at time t

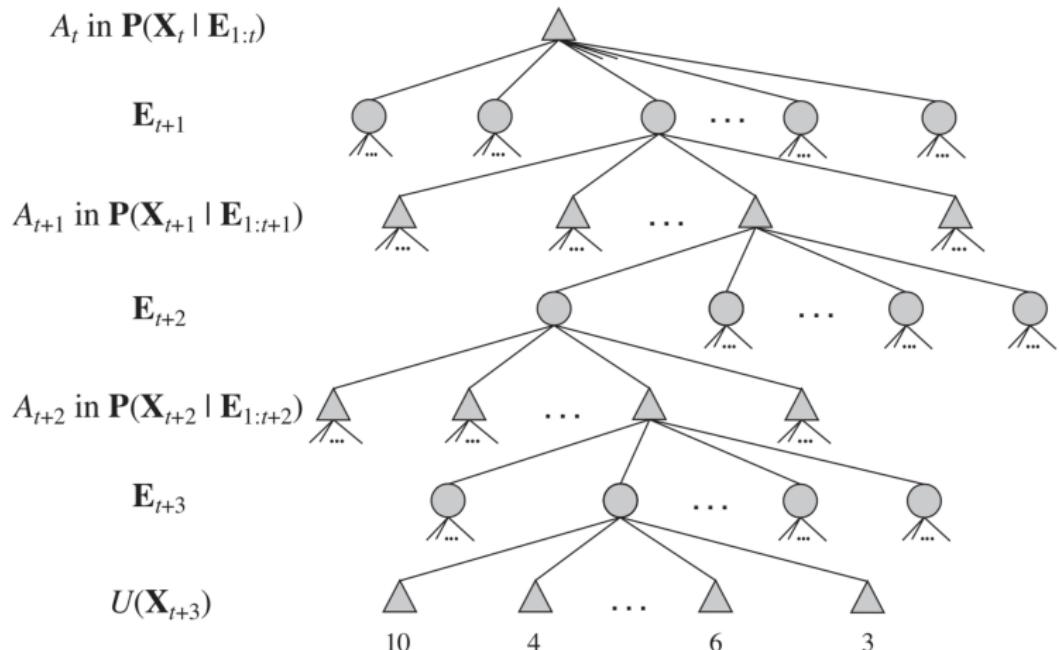


Variables with known values are gray, agent must choose a value for A_t .

Rewards for $t = 0, \dots, t + 2$, but utility for $t + 3$ (\cong discounted sum of rest)

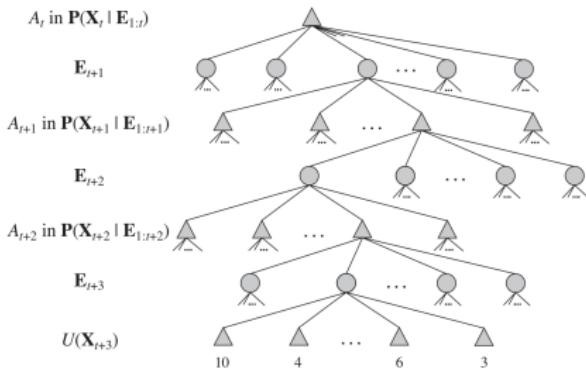
Designing Online Agents for POMDPs (continued)

- Part of the lookahead solution of the DDN above (search over action tree)



circle $\hat{=}$ chance nodes (the environment decides) triangle $\hat{=}$ belief state (each action decision is taken there)

Designing Online Agents for POMDPs (continued)



- ▶ Note: belief state update is deterministic irrespective of the action outcome
~ no chance nodes for action outcomes
- ▶ belief state at triangle computed by filtering with actions/percepts leading to it
 - ▶ for decision A_{t+i} will have percepts $\mathbf{E}_{t+1:t+i}$ (even if it does not know their values at time t)
 - ▶ A POMDP-agent automatically takes into account the value of information and executes information-gathering actions where appropriate.
- ▶ Time complexity for exhaustive search up to depth d is $\mathcal{O}(|A|^d \cdot |E|^d)$ ($|A| \hat{=} \text{number of actions, } |E| \hat{=} \text{number of percepts})$

- ▶ Decision theoretic **agents** for sequential environments
- ▶ Building on temporal, probabilistic models/inference **(dynamic Bayesian networks)**
- ▶ **MDPs** for fully observable case.
- ▶ Value/Policy Iteration for **MDPs** \leadsto optimal policies.
- ▶ **POMDPs** for **partially observable** case
- ▶ \cong MDP on **belief state** space.
- ▶ The world is a **POMDP** with (initially) unknown **transition** and **sensor models**.

- [BF95] Avrim L. Blum and Merrick L. Furst. "Fast planning through planning graph analysis". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Chris S. Mellish. Montreal, Canada: Morgan Kaufmann, San Mateo, CA, 1995, pp. 1636–1642.
- [BF97] Avrim L. Blum and Merrick L. Furst. "Fast planning through planning graph analysis". In: *Artificial Intelligence* 90.1-2 (1997), pp. 279–298.
- [BG01] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search". In: *Artificial Intelligence* 129.1–2 (2001), pp. 5–33.
- [BG99] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search: New Results". In: *Proceedings of the 5th European Conference on Planning (ECP'99)*. Ed. by S. Biundo and M. Fox. Springer-Verlag, 1999, pp. 60–72.
- [BKS04] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. "Towards Understanding and Harnessing the Potential of Clause Learning". In: *Journal of Artificial Intelligence Research* 22 (2004), pp. 319–351.

References II

- [Bon+12] Blai Bonet et al., eds. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 2012.
- [Byl94] Tom Bylander. "The Computational Complexity of Propositional STRIPS Planning". In: *Artificial Intelligence* 69.1–2 (1994), pp. 165–204.
- [CKT91] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. "Where the Really Hard Problems Are". In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by John Mylopoulos and Ray Reiter. Sydney, Australia: Morgan Kaufmann, San Mateo, CA, 1991, pp. 331–337.
- [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.
- [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.

References III

- [Ede01] Stefan Edelkamp. "Planning with Pattern Databases". In: *Proceedings of the 6th European Conference on Planning (ECP'01)*. Ed. by A. Cesta and D. Borrajo. Springer-Verlag, 2001, pp. 13–24.
- [FD14] Zohar Feldman and Carmel Domshlak. "Simple Regret Optimization in Online Planning for Markov Decision Processes". In: *Journal of Artificial Intelligence Research* 51 (2014), pp. 165–205.
- [Fis] John R. Fisher. *prolog :- tutorial*. URL:
https://www.cpp.edu/~jrfisher/www/prolog_tutorial/
(visited on 10/10/2019).
- [FL03] Maria Fox and Derek Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124.
- [Fla94] Peter Flach. Wiley, 1994. ISBN: 0471 94152 2. URL:
<https://github.com/simply-logical/simply-logical/releases/download/v1.0/SL.pdf>.

References IV

- [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.
- [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.
- [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+13] Ivan Herman et al. *RDFa 1.1 Primer – Second Edition. Rich Structured Data Markup for Web Documents*. W3C Working Group 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.

References VII

- [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.
- [Hof05] Jörg Hoffmann. "Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks". In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 685–758.
- [How60] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
- [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.

References VIII

- [Kee74] R. L. Keeney. "Multiplicative utility functions". In: *Operations Research* 22 (1974), pp. 22–34.
- [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.

References IX

- [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>.
- [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.
- [LPN] *Learn Prolog Now!* URL: <http://lpn.swi-prolog.org/> (visited on 10/10/2019).
- [Luc96] Peter Lucas. "Knowledge Acquisition for Decision-theoretic Expert Systems". In: *AISB Quarterly 94* (1996), pp. 23–33. URL: https://www.researchgate.net/publication/2460438_Knowledge_Acquisition_for_Decision-theoretic_Expert_Systems.
- [McD+98] Drew McDermott et al. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee. 1998.
- [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.
- [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.

References XIII

- [Pra+94] Malcolm Pradhan et al. "Knowledge Engineering for Large Belief Networks". In: *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence*. UAI'94. Seattle, WA: Morgan Kaufmann Publishers Inc., 1994, pp. 484–490. ISBN: 1-55860-332-8. URL: <http://dl.acm.org/citation.cfm?id=2074394.2074456>.
- [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>.

- [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: *Proceedings 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. Pearson Education, 2003. ISBN: 0137903952.
- [RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall Press, 2009. ISBN: 0136042597, 9780136042594.
- [RW10] Silvia Richter and Matthias Westphal. "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 127–177.

- [Sil+16] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529 (2016), pp. 484–503.
URL: <http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>.
- [SWI] *SWI Prolog Reference Manual*. URL:
<https://www.swi-prolog.org/pldoc/refman/> (visited on 10/10/2019).
- [Tur50] Alan Turing. "Computing Machinery and Intelligence". In: *Mind* 59 (1950), pp. 433–460.
- [Wal75] David Waltz. "Understanding Line Drawings of Scenes with Shadows". In: *The Psychology of Computer Vision*. Ed. by P. H. Winston. McGraw-Hill, 1975, pp. 1–19.