

Artificial Intelligence (Künstliche Intelligenz) 1

Summer Semester 2018/19

– Lecture Notes –

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

July 25, 2019

Preface

Course Concept

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

Context: The course “Künstliche Intelligenz” (KI 1 & 2) at FAU Erlangen is a two-semester course in the “Wahlpflichtbereich” (specialization phase) in semesters 5/6 of the Bachelor program “Computer Science” at FAU Erlangen. It is also available as a (somewhat remedial) course in the “Vertiefungsmodul Künstliche Intelligenz” in the Computer Science Master’s program.

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

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

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

Course Contents

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

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

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

This Document

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

Caveat: This document is made available for the students of this course only. It is still very much a draft and will develop over the course of the current course and in coming academic years.

Licensing: This document is licensed under a [Creative Commons license](#) that requires attribution, allows commercial use, and allows derivative works as long as these are licensed under the same license.

Knowledge Representation Experiment:

This document is also an experiment in knowledge representation. Under the hood, it uses the [STEX](#) package [Koh08; Koh18], a TEX/LATEX extension for semantic markup, which allows to export the contents into [active documents](#) that adapt to the reader and can be instrumented with services based on the explicitly represented meaning of the documents.

Acknowledgments

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

All course materials have bee restructured and semantically annotated in the S^TE_X format, so that we can base additional semantic services on them.

AI Students: The following students have submitted corrections and suggestions to this and earlier versions of the notes: Rares Ambrus, Ioan Sucan, Yashodan Nevatia, Dennis Müller, Simon Rainer, Demian Vöhringer, Lorenz Gorse, Philipp Reger, Benedikt Lorch, Maximilian Lösch, Luca Reeb, Marius Frinken, Peter Eichinger, Oskar Herrmann, Daniel Höfer, Stephan Mattejat, Matthias Sonntag, Jan Urfei, Tanja Würsching, Adrian Kretschmer, Tobias Schmidt, Maxim Onciu, Armin Roth.

Recorded Syllabus

In this document, we record the progress of the course in the academic year 2017/18 in the form of a “recorded syllabus”, i.e. a syllabus that is created after the fact rather than before.

Recorded Syllabus Winter Semester 2017/18:

#	date	until	slide	page
1	Oct 17.	admin, what is AI?	544	335
2	Oct 18.	AI components, overview	549	342
3	Oct 24.	strong/narrow AI, PROLOG	33	28
4	Oct 25.	PROLOG	42	35
5	Nov. 2.	Complexity, rational Agents	69	53
	Nov. 1.	Allerheiligen (public holiday)		
6	Nov. 7.	Types of Environments/Agents	82	59
7	Nov. 8.	Problem Solving & Search	101	74
8	Nov. 14.	Uninformed Search	121	88
9	Nov. 15.	Informed Search, A*, heuristics	144	100
10	Nov. 21.	local search, Game Play	162	109
11	Nov. 22.	minimax, evaluation functions	180	117
12	Nov. 28.	alphabeta search, MCTS	194	124
13	Nov. 29.	CSP, Waltz Algorithm, Constraint Types	222	137
14	Dec. 5.	CSP as search/heuristics	237	145
15	Dec. 6.	CS Propagation, Forward Checking		
16	Dec. 12.	CSP decomposition	270	166
17	Dec. 13.	Wumpus & Propositional Logic	286	180
18	Dec. 19.	Logics and Inference in General	294	186
19	Dec 20.	Review Formal Systems, Propositional Tableaux	407	248
20	Jan 9.	New Year recap; Killing the Wumpus with Resolution	324	202
21	Jan 10.	DPLL (abbreviated), phase transitions		
22	Jan. 16.	First Order Semantics	384	234
23	Jan 17.	Free Variable Tableaux	409	249
24	Jan 23.	Unification, Prolog as Resolution, Abduction, Induction	432	262
25	Jan 24.	Intro to Planning, STRIPS	472	286
26	Jan 30.	Review Planning, PDDL, PlanEx/Len	472	286
27	Jan 31.	planning by heuristic search	501	305
28	Feb 7.	Belief States, AI-1 Conclusion	981	573

Recorded Syllabus Summer Semester 2019:

#	date	until	slide	page
1.	April 24.	admin, overview	562	353
2.	April 25.	Recap agents, belief states, probabilities	590	366
	May 1.	Tag der Arbeit		
3.	May 2.	Bayesian Reasoning	608	374
4.	May 8.	conditional independence, Naive Bayes Networks	622	381
5.	May 9.	Bayesian Networks	639	391
6.	May 16.	constructing and reasoning with BN	662	402
7.	May 17.	decision theory, preferences & utilities	673	410
8.	May 22.	Utilities and Decision Networks	685	417
9.	May 23	Information Value Theory, Markov Chains		
	May 29.	Public Holiday: Ascension		
10.	May 30.	Markov Inference Tasks, HMM	716	434
11.	June 5.	HMMs, Markov Decisions Problems	723	440
12.	June 6.	MDPs, Value/Policy Iteration/-Evaluation	744	450
13.	June 12.	inductive learning, decision tree learning	756	461
14.	June 13.	DTL + Information Gain, Over/Underfitting	768	467
15.	June 19.	learning evaluation, loss, Regularization	780	473
	June 20.	Public Holiday: Corpus Christi		
16.	26. June	PAC Learning, Linear regression	799	481
17.	27. June	Linear classification, Neural Networks	809	486
18.	3. July	Neural Networks	830	495
19.	4. July	Statistical Bayesian Learning	857	510
20.	10. July	Knowledge in Learning	907	529
21.	11. July	Tutorial Machine Learning Challenge		
22.	17. July	Support Vector Machines, Reinforcement Learning	921	537
23.	18. July	Natural Language Processing & Semantics Introduction	940	551
24.	24. July	<i>N</i> -grams, Information Retrieval	952	557
25.	25. July	Word Embeddings, Real Language, Wrap-Up; What did we learn?	986	578

Here the syllabus of the last academic year for reference, the current year should be similar; see the course notes of last year available for reference at <http://kwarc.info/teaching/AI/notes-2017-18.pdf>.

Recorded Syllabus Winter Semester 2017/18:

#	date	until	
1	Oct 19.	admin, overview	
2	Oct 25.	overview, topics	
3	Oct 26.	rationality	
	Nov. 1.	Allerheiligen (public holiday)	
4.	Nov. 2.	Environment & agent types.	
5.	Nov. 8.	Prolog	
6.	Nov. 9.	Problem Solving & Search	
7.	Nov. 15.	Relaxed Problems	
8.	Nov. 16.	Problem Solving & Search	
9.	Nov. 22.	minimax & alpha-pruning	
10.	Nov. 23.	MCTS & AlphaGo, Constraint Problems	
11.	Nov. 29.	CSP, Waltz Algorithm, Constraint Types	
12.	Nov. 30.	CSP search/heuristics	
13.	Dec. 6.	solving acyclic CSP graphs	
14.	Dec. 7.	CSP decomposition, Wumpus World	
15	Dec. 13	Kalah Competition Happening	
16.	Dec. 14	Calculi & Proofs, Propositional Natural Deduction	
17.	Dec 20.	Review Formal Systems, Propositional Tableaux	
18.	Dec 21.	Killing the Wumpus; DPPL	
19.	Jan 10.	New recap, Implication Graphs	
20.	Jan. 11	Clause Learning, First-order Logic	
21.	Jan. 17.	First Order Semantics, Substitutions, FO Natural Deduction	
22	Jan 18.	Free Variable Tableaux	
23	Jan 24.	Prolog as Resolution, Abduction, Induction	
24	Jan 25.	Intro to Planning, STRIPS, PDDL	
25	Jan 31.	Review Planning, Evaluation	
26	Feb 1.	planning by heuristic search	
27	Feb 7.	PlanEx ⁺	
28	Feb 8.	AI-1 Conclusion	

Recorded Syllabus Summer Semester 2017:

#	date	until	
1	May 4.	overview, some admin	
2.	May 8.	probabilities, Kolmogorov axioms	
3.	11. May	conditional probabilities, independence, Bayes rule	
4.	15. May	conditional independence, Bayesian Networks	
5.	18. May	constructing and reasoning with BN	
6.	22. May	decision theory, preferences & utilities	
7.	29. May	Multi-Attribute Utilities	
8.	1. Jun	Belief, transition, and sensor models, VPI	
9.	8. Jun	Markov Processes, temporal inference	
10.	12. Jun	temporal inference	
11.	19. Jun	Markov Decision Procedures, Value Iteration	
12.	22. Jun	POMDPs, ML Intro	
13.	26. Jun	decision tree learning, performance measurement	
14.	29. Jun	learning evaluation, loss	
15.	3. Juli	Linear regression and classification	
16.	6. Juli	Neural Networks	
17.	10. Juli	Support Vector Machines	
18.	13. Juli	Knowledge in Learning	
	17. - 21-0. Juli	Neural Nets Challenge	
19.	24. Juli	Statistical Learning	
20.	28. Juli	Wrap-Up; What did we learn?	

Contents

Preface	i
Course Concept	i
Course Contents	i
This Document	i
Acknowledgments	i
Recorded Syllabus	iii
1 Administrativa	1
2 About My Lecturing ...	5
3 Overview over the Course	9
3.1 What is Artificial Intelligence?	9
3.2 Artificial Intelligence is here today!	12
3.3 Two Ways to Attack the AI Problem	16
3.4 AI Topics Covered	17
3.5 Strong vs. Narrow AI	18
3.6 AI in the KWARC Group	19
I Getting Started with AI: A Conceptual Framework	23
4 Logic Programming	27
4.1 Introduction to Logic Programming and PROLOG	27
4.2 Programming as Search	30
5 Recap: Complexity Analysis in AI?	37
6 Intelligent Agents: a Unifying Framework for AI	45
6.1 Introduction: Rationality in Artificial Intelligence	45
6.2 Agents and Environments as a Framework for AI	48
6.3 Good Behavior \rightsquigarrow Rationality	51
6.4 Classifying Environments	53
6.5 Types of Agents	54
6.6 Representing the Environment in Agents	60
II General Problem Solving	63
7 Problem Solving and Search	67
7.1 Problem Solving	67
7.2 Search	74
7.3 Uninformed Search Strategies	76
7.4 Informed Search Strategies	88

7.4.1 Greedy Search	88
7.4.2 Heuristics and their Properties	92
7.4.3 A-Star Search	95
7.4.4 Finding Good Heuristics	99
7.5 Local Search	101
8 Adversarial Search for Game Playing	107
8.1 Introduction	107
8.2 Minimax Search	111
8.3 Evaluation Functions	114
8.4 Alpha-Beta Search	117
8.5 MCTS	122
8.6 State of the Art	126
9 Constraint Satisfaction Problems	129
9.1 Constraint Satisfaction Problems: Motivation	129
9.2 The Waltz Algorithm	133
9.3 CSP: Towards a Formal Definition	136
9.4 CSP as Search	141
10 Constraint Propagation	149
10.1 Introduction	149
10.2 Inference	150
10.3 Forward Checking	154
10.4 Arc Consistency	156
10.5 Decomposition: Constraint Graphs, and Two Simple Cases	161
10.6 Cutset Conditioning	163
10.7 Constraint Propagation with Local Search	164
10.8 Conclusion & Summary	166
III Knowledge and Inference	169
11 Propositional Reasoning, Part I: Principles	173
11.1 Introduction	173
11.2 Propositional Logic (Syntax/Semantics)	177
11.3 Formal Systems (Syntax and Semantics in General)	181
11.4 Propositional Natural Deduction Calculus	186
11.5 Machine-Oriented Calculi for Propositional Logic	190
11.5.1 Calculi for Automated Theorem Proving: Analytical Tableaux	191
11.5.2 Resolution for Propositional Logic	199
11.6 Killing a Wumpus with Propositional Inference	201
11.7 Conclusion	203
12 Propositional Reasoning: SAT Solvers	205
12.1 Introduction	205
12.2 Davis-Putnam	208
12.3 DPLL = (A Restricted Form of) Resolution	210
12.4 UP Conflict Analysis	213
12.5 Clause Learning	218
12.6 Phase Trans.	221
12.7 Conclusion	224

13 First Order Predicate Logic	227
13.1 Introduction	227
13.2 First-Order Logic	231
13.2.1 First-Order Logic: Syntax and Semantics	231
13.2.2 First-Order Substitutions	235
13.3 First-Order Calculi	237
13.3.1 Propositional Natural Deduction Calculus	238
13.4 First-Order Predicate Logic (Conclusion)	245
14 First-Order Inference	247
14.1 First-Order Inference with Tableaux	247
14.1.1 First-Order Tableaux	247
14.1.2 Free Variable Tableaux	249
14.1.3 First-Order Unification	251
14.2 First-Order Resolution	257
14.2.1 Resolution Examples	257
15 Logic Programming as Resolution Theorem Proving	261
IV Planning & Acting	265
16 Planning I: Framework	269
16.1 Planning: Introduction	269
16.2 Planning History	274
16.3 STRIPS Planning	280
16.4 PDDL Language	285
16.5 Planning Complexity	288
16.6 Conclusion	291
17 Planning II: Algorithms	293
17.1 Introduction	293
17.2 How to Relax	295
17.3 Delete Relaxation	302
17.4 The h^+ Heuristic	308
17.5 Planning Algorithms: Conclusion	315
18 Planning and Acting in the Real World	317
18.1 Introduction	317
18.2 Agent Architectures based on Belief States	319
18.3 Conformant Planning	320
18.4 Conditional Planning	323
19 Semester Change-Over	327
19.1 What did we learn in AI 1?	327
19.2 Administrativa	333
19.3 Overview over AI and Topics of AI-II	335
19.3.1 What is Artificial Intelligence?	335
19.3.2 Artificial Intelligence is here today!	337
19.3.3 Two Ways to Attack the AI Problem	342
19.3.4 AI in the KWARC Group	343
19.3.5 AI-II: Advanced Rational Agents	344

V Reasoning with Uncertain Knowledge	347
20 Quantifying Uncertainty	351
20.1 Dealing with Uncertainty: Probabilities	351
20.1.1 Sources of Uncertainty	351
20.1.2 Recap: Rational Agents as a Conceptual Framework	352
20.1.3 Agent Architectures based on Belief States	356
20.1.4 Modeling Uncertainty	358
20.1.5 Acting under Uncertainty	361
20.1.6 Agenda for this Chapter: Basics of Probability Theory	362
20.2 Unconditional Probabilities	363
20.3 Conditional Probabilities	367
20.4 Independence	368
20.5 Basic Methods	371
20.6 Bayes' Rule	374
20.7 Conditional Independence	376
20.8 The Wumpus World Revisited	380
20.9 Conclusion	383
21 Probabilistic Reasoning, Part II: Bayesian Networks	385
21.1 Introduction	385
21.2 What is a Bayesian Network?	387
21.3 What is the Meaning of a Bayesian Network?	389
21.4 Constructing Bayesian Networks	392
21.5 Inference in Bayesian Networks	397
21.6 Conclusion	402
22 Making Simple Decisions Rationally	405
22.1 Introduction	405
22.2 Rational Preferences	406
22.3 Utilities and Money	408
22.4 Multi-Attribute Utility	410
22.5 Decision Networks	416
22.6 The Value of Information	418
23 Temporal Probability Models	423
23.1 Modeling Time and Uncertainty	423
23.2 Inference: Filtering, Prediction, and Smoothing	426
23.3 Hidden Markov Models	431
23.4 Dynamic Bayesian Networks	436
24 Making Complex Decisions	439
24.1 Sequential Decision Problems	439
24.2 Utilities over Time	441
24.3 Value/Policy Iteration	443
24.4 Partially Observable MDPs	447
24.5 Online Agents with POMDPs	448
VI Machine Learning	451
25 Learning from Observations	455
25.1 Forms of Learning	455
25.2 Inductive Learning	457
25.3 Learning Decision Trees	460

CONTENTS	xi
25.4 Using Information Theory	463
25.5 Evaluating and Choosing the Best Hypothesis	467
25.6 Computational Learning Theory	473
25.7 Regression and Classification with Linear Models	477
25.8 Artificial Neural networks	485
25.9 Support Vector Machines	495
26 Statistical Learning	501
26.1 Full Bayesian Learning	501
26.2 Approximations of Bayesian Learning	504
26.3 Parameter Learning for Bayesian Networks	505
26.4 Naive Bayes Models	508
27 Knowledge in Learning	511
27.1 Logical Formulations of Learning	511
27.2 Explanation-Based Learning	515
27.3 Relevance-Based Learning	518
27.4 Inductive Logic Programming	522
27.4.1 An Example	523
27.4.2 Top-Down Inductive Learning: FOIL	525
27.4.3 Inverse Resolution	527
28 Reinforcement Learning	531
28.1 Reinforcement Learning: Introduction & Motivation	531
28.2 Passive Learning	532
28.3 Active Reinforcement Learning	536
VII Communication	539
29 Natural Language Processing	543
29.1 Introduction to NLP	543
29.2 Natural Language and its Meaning	545
29.3 Looking at Natural Language	549
29.4 Language Models	552
29.5 Information Retrieval	555
29.6 Word Embeddings	558
30 Natural Language for Communication	561
30.1 Grammar	563
30.2 Real Language	569
31 What did we learn in AI 1/2?	573
VIII Excursions	579
A Completeness of Calculi for Propositional Logic	581
A.1 Abstract Consistency and Model Existence	581
A.2 A Completeness Proof for Propositional Tableaux	587

B Completeness of Calculi for First-Order Logic	591
B.1 Abstract Consistency and Model Existence	591
B.2 A Completeness Proof for First-Order ND	598
B.3 Soundness and Completeness of First-Order Tableaux	599
B.4 Soundness and Completeness of First-Order Resolution	601

Chapter 1

Administrativa

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

Prerequisites for AI-1

- ▷ the mandatory courses in Computer Science from Semesters 1-4, in particular:
 - ▷ course “Algorithmen und Datenstrukturen”.
 - ▷ course “Grundlagen der Logik in der Informatik” (GLOIN).
 - ▷ course “Berechenbarkeit und Formale Sprachen”.
- If you have not taken these (or do not remember), read up on them as needed.
- ▷ Motivation, Interest, Curiosity, hard work
- ▷ You can do this course if you want!



©: Michael Kohlhase

1



Now we come to a topic that is always interesting to the students: the grading scheme.

Assessment, Grades

- ▷ **Academic Assessment:** 90 minutes exam at the end of the semester (~ Feb. 10.)
- ▷ **Retake Exam:** 90 min exam at the end of the following semester (~ July 15.)
- ▷ **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.



I basically do not have a choice in the grading sheme, as it is essentially the only one consistent with university policies. For instance, I would like to give you more incentives for the homework assignments – which would also mitigate the risk of having a bad day in the exam. Also, graded Wednesday quizzes would help you prepare for the lectures and thus let you get more out of them, but that is also impossible.

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](#)
 - ▷ please sign up for the AI-1 under <https://studon.fau.de/crs2361111.html>
 - ▷ Homeworks are handed in electronically (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.



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

Tutorials for Artificial Intelligence 1

- ▷ Weekly tutorials and homework assignments (first one in week two)
- ▷ **Instructor/Lead TA:** Dennis Müller (dennis.mueller@fau.de) Room: 11.138, Tel: 85-64053
- ▷ **Tutorials:** one each for Alpcan Dalga, Lisa-Marie Dreier, Marius Frinken, Max Rapp
- ▷ **Goal 1:** Reinforce what was taught in class (need)

- ▷ 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 it by having looked at the slides and the homework assignments
- ▷ Inverted Classroom: the latest craze in didactics (works well if done right)
in CS: Lecture + Homework assignments + Tutorials $\hat{=}$ Inverted Classroom



©: Michael Kohlhase

4



Do use the opportunity to discuss the AI-1 topics with others. After all, one of the non-trivial skills you want to learn in the course is how to talk about Artificial Intelligence topics. And that takes practice, practice, and practice.

Textbook, Handouts and Information, Forums

- ▷ 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 (unfortunately, the German version is based on edition 2)
- ▷ Course notes will be posted at <http://kwarc.info/teaching/AI>
 - ▷ 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)
- ▷ Announcements will be posted on the course forum (choose the right one)
 - ▷ <https://fsi.cs.fau.de/forum/144-Kuenstliche-Intelligenz>
 - ▷ <https://fsi.cs.fau.de/forum/149-Kuenstliche-Intelligenz-II>
- ▷ Check the forum frequently for
 - ▷ announcements, homeworks, questions
 - ▷ discussion among your fellow students



©: Michael Kohlhase

5



⚠ Special Admin Conditions ⚠

- ▷ 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

About My Lecturing . . .

First let me state the obvious – this is really still part of the admin – but there is an important point I want to make.

Do I need to attend the lectures

- ▷ Attendance is not mandatory for the AI-1 lecture
- ▷ There are two ways of learning AI-1: (both are OK, your mileage may vary)
 - ▷ Approach **B**: Read a [Book](#)
 - ▷ Approach **I**: come to the lectures, be [involved](#), interrupt me whenever you have a question.
- The only advantage of **I** over **B** is that books do not answer questions ([yet!](#) ↵ we are working on this in AI research)
- ▷ Approach **S**: come to the lectures and [sleep does not work!](#)
- ▷ [I really mean it](#): If you come to class, be involved, ask questions, challenge me with comments, tell me about errors, . . .
 - ▷ I would much rather have a lively discussion than get through all the slides
 - ▷ You learn more, I have more fun (Approach **B** serves as a [backup](#))
 - ▷ You may have to change your habits, overcome shyness, . . . (please do!)
- ▷ This is what I get paid for, and I am more expensive than most books([get your money's worth](#))



©: Michael Kohlhase

7



That being said – I know that it sounds quite idealistic – can I do something to help you along in this? Let me digress on lecturing styles ↵ take the following with “cum kilo salis”¹, I want to make a point here, not bad-mouth my colleagues.!

Traditional Lectures (cum kilo salis)

¹with much more than the proverbial grain of salt.

- ▷ One person talks to 50+ students who just listen and take notes
- ▷ The *I have a book hat you do not have* style makes it hard to stay awake



- ▷ It is well-known that frontal teaching does not optimize learning
- ▷ But it scales very well
(especially when televised)



©: Michael Kohlhase

8



So there is a tension between

- scalability of teaching – which is a legitimate concern for an institution like FAU, and
- effectiveness/efficiency of learning – which is a legitimate concern for students

My Lectures? What can I do to keep you awake?

- ▷ We know how to keep large audiences engaged and motivated (even televised)
- ▷ But the topic is different (AI-1 is arguably more complex than Sports/Media)



- ▷ We're not gonna be able to go all the way to TV entertainment ("AI-1 total")
- ▷ But I am going to (try to) incorporate some elements ...



©: Michael Kohlhase

9



I will use interactive elements I call “questionnaires in my course. Here is one example to give you an idea of what is coming.

Questionnaire

- ▷ **Question:** How many scientific articles (6-page double-column “papers”) were submitted to the 2016 International Joint Conference on Artificial Intelligence (IJCAI’16) in New York City?
 - a) 7?
 - b) 811?
 - c) 1996?
 - d) 2296?
- ▷ **Answer:** (d) is correct. (Previous year, IJCAI’15, answer (c) was correct . . .)
- ▷ **Questionnaires:** are my attempt to get you to interact
 - ▷ At end of each logical unit (most, if I can get around to preparing them)
 - ▷ You get 2 -5 minutes, feel free to make noise (e.g. discuss with your neighbors)



©: Michael Kohlhase

10



One of the reasons why I like the questionnaire format is that it is a small instance of a question-answer game that is much more effective in inducing learning – recall that learning happens in the head of the student, no matter what the instructor tries to do – than frontal lectures. In fact Sokrates – the grand old man of didactics – is said to have taught his students exclusively by asking leading questions. His style coined the name of the teaching style “Socratic Dialogue”, which unfortunately does not scale to a class of 100+ students.

More Generally: My Questions to You

- ▷ **When will I ask them?**
 - ▷ In questionnaires.
 - ▷ At various points during the lectures.
 - ▷ We’ll do examples together.
- ▷ **Why do I ask them?**
 - ▷ They give you the option to follow the lectures *actively*.
 - ▷ They allow me to check whether or not you are able to follow.
- ▷ **How will I look for answers?**
 - ▷ “Streber syndrom”: 3 students answer all the questions, $N - 3$ sleep.
 - ▷ If this happens, I may resort to picking students randomly.

There is nothing to be ashamed of when giving a wrong answer! You wouldn't believe the number of times I got something wrong myself (I do hope all bugs are removed now, but . . .)



©: Michael Kohlhase

11



Unfortunately, this idea of adding questionnaires is mitigated by a simple fact of life. Good questionnaires require good ideas, which are hard to come by; in particular for AI-1-2, I do not have many. But maybe you – the students – can help.

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)



©: Michael Kohlhase

12



Chapter 3

Overview over the Course

We restart the new semester by reminding ourselves of (the problems, methods, and issues of) Artificial Intelligence, and what has been achieved so far.

Plot of this Course

- ▷ Today: Motivation, Admin, and find out what you already know
 - ▷ What is Artificial Intelligence?
 - ▷ What has AI already achieved?
 - ▷ a (very) quick walk through the topics
 - ▷ how can you get involved with AI at KWARC



©: Michael Kohlhase

13



3.1 What is Artificial Intelligence?

The first question we have to ask ourselves is “what is Artificial Intelligence”, i.e. how can we define it. And already that poses a problem since the natural definition *like human intelligence, but artificially realized* presupposes a definition of Intelligence, which is equally problematic; even Psychologists and Philosophers – the subjects nominally “in charge” of human intelligence – have problems defining it, as witnessed by the plethora of theories e.g. found at [WHI].

What is Artificial Intelligence? Definition

▷ **Definition 3.1.1 (According to Wikipedia)**

Artificial Intelligence (AI) is intelligence exhibited by machines

▷ **Definition 3.1.2 (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

▷ **Elaine Rich**: AI studies how we can make the computer do things that humans can still do better at the moment.

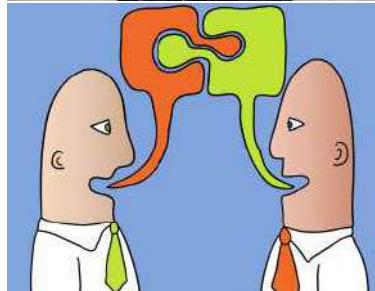


©: Michael Kohlhase

14

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

What is Artificial Intelligence? Components



3.2 Artificial Intelligence is here today!

The components of Artificial Intelligence are quite daunting, and none of them are fully understood, much less achieved artificially. But for some tasks we can get by with much less. And indeed that is what the field of Artificial Intelligence does in practice – but keeps the lofty ideal around. This practice of “trying to achieve AI in selected and restricted domains” (cf. the discussion starting with slide 24) has borne rich fruits: systems that meet or exceed human capabilities in such areas. Such systems are in common use in many domains of application.

Artificial Intelligence is here today

- ▷ in outer space
 - ▷ in outer space systems need autonomous control:
 - ▷ remote control impossible due to time lag
- ▷ in artificial limbs
 - ▷ the user controls the prosthesis via existing nerves, can e.g. grip a sheet of paper.
- ▷ in household appliances
 - ▷ The iRobot Roomba vacuums, mops, and sweeps in corners,, parks, charges, and discharges.
 - ▷ general robotic household help is on the horizon.
- ▷ in hospitals
 - ▷ in the USA 90% of the prostate operations are carried out by RoboDoc
 - ▷ Paro is a cuddly robot that eases solitude in nursing homes.





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



- ▷ AlphaGo is a program developed 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.
- ▷ 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.



We will conclude this Section with a note of caution.

The AI Conundrum

- ▷ **Observation:** Reserving the term “Artificial Intelligence” has been quite a land-grab!
- ▷ **But:** researchers at the Dartmouth Conference (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 3.2.1** Functional/Logic Programming, Automated Theorem Proving, Planning, Machine Learning, Knowledge Representation, ...
- ▷ **Still Consequence:** AI research was alternately flooded with money and cut off brutally.



3.3 Two Ways to Attack the AI Problem

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

There are currently two main avenues of attack to the problem of building artificially intelligent systems. The (historically) first is based on the symbolic representation of knowledge about the world and uses inference-based methods to derive new knowledge on which to base action decisions. The second uses statistical methods to deal with uncertainty about the world state and learning methods to derive new (uncertain) world assumptions to act on.

Two ways of reaching Artificial Intelligence?

- ▷ Two avenues of attack for the problem: knowledge-based and statistical techniques
(they are complementary)

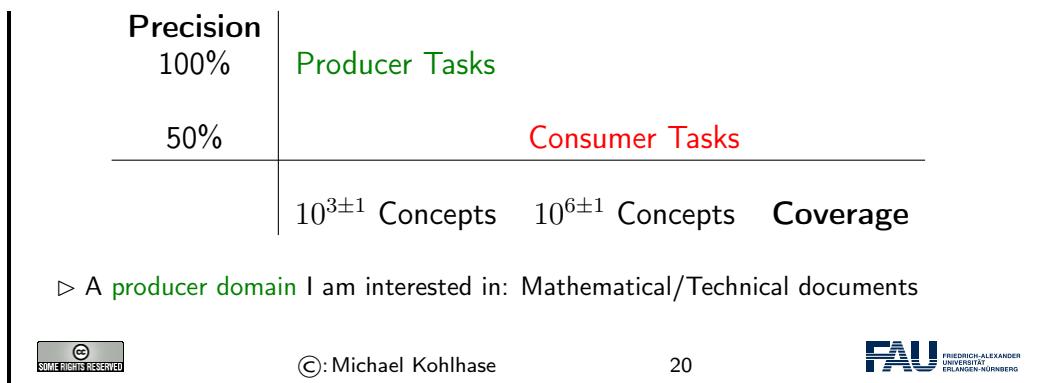
Deep	Knowledge-based AI-1	Not there yet <i>cooperation?</i>
Shallow	no-one wants this	Statistical Methods AI-2
Analysis ↑ vs. Coverage →	Narrow	Wide

- ▷ We will cover foundational methods of deep/narrow processing in this semester and shallow/wide-coverage ones in the next.



Environmental Niches for both Approaches to AI

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



©: Michael Kohlhase

20



To get this out of the way ...



- ▷ AlphaGo = search + neural networks
 - ▷ we do search this semester and cover neural networks in KI-2.
 - ▷ I will explain AlphaGo a bit in Chapter 8.



©: Michael Kohlhase

21



3.4 AI Topics Covered

Topics of AI-1 (Winter Semester)

- ▷ Getting Started
 - ▷ What is Artificial Intelligence (situating ourselves)
 - ▷ Intelligent Agents (a unifying framework)
 - ▷ Logic Programming in Prolog (An influential paradigm)
- ▷ Problem Solving
 - ▷ Problem Solving and Search
 - ▷ Game playing (Adversarial Search)
 - ▷ Constraint Satisfaction Problems

- ▷ Knowledge and Reasoning
 - ▷ Formal Logic as the Mathematics of Meaning
 - ▷ Logic Programming
- ▷ Planning
 - ▷ Planning
 - ▷ Planning and Acting in the real world



Topics of AI-2 (Summer Semester)

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



3.5 Strong vs. Narrow AI

Strong AI vs. Narrow AI

- ▷ **Definition 3.5.1** 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, controlling elevators, composing music, ...)
- ▷ **Definition 3.5.2** 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 3.5.3** Problems requiring strong AI to solve are called **AI complete**.
- ▷ **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.



A few words on AGI...

- ▷ The conceptual and mathematical framework (Agents, environments etc.) is the same for strong AGI 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 fluctuates massively, recently increased (again). (correlates somewhat with AI winters and golden years)
- ▷ Public attention increasing due to talk of "existential risks" (e.g. Hawking, Musk, Bostrom, Yudkowsky, Obama, ...)



AGI Research

- ▷ "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!



3.6 AI in the KWARC Group

▷ The KWARC Research Group

- ▷ **Observation:** The ability to represent knowledge about the world and to draw logical inferences is one of the central components of intelligent behavior.
- ▷ **Thus:** reasoning components of some form are at the heart of many AI systems.

- ▷ **KWARC Angle:** Scaling up (web-coverage) without dumbing down (too much)
 - ▷ **Content markup** instead of full formalization (too tedious)
 - ▷ **User support** and **quality control** instead of “The Truth” (elusive anyway)
 - ▷ use **Mathematics** as a test tube (\triangleleft Mathematics $\hat{=}$ Anything Formal \triangleleft)
 - ▷ 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



©: Michael Kohlhase

27



Overview: KWARC Research and Projects

Applications: eMath 3.0, Active Documents, Semantic Spreadsheets, Semantic 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:

- ▷ LATEX XML: LATEX \rightarrow XML
- ▷ STEX: Semantic LATEX
- ▷ invasive editors
- ▷ Context-Aware IDEs
- ▷ Mathematical Corpora
- ▷ Linguistics of Math
- ▷ ML for Math Semantics Extraction

Foundations: Computational Logic, Web Technologies, OMDoc/MMT



©: Michael Kohlhase

28



Research Topics in the KWARC Group

- ▷ We are always looking for bright, motivated KWARCies
- ▷ We have topics in for all levels (Enthusiast, Bachelor, Master, Ph.D.)
- ▷ List of current topics: <https://gl.kwarc.info/kwarc/thesis-projects/>
 - ▷ Automated Reasoning: Maths Representation in the Large
 - ▷ Logics development, (Meta) n -Frameworks
 - ▷ Math Corpus Linguistics: Semantics Extraction
 - ▷ Serious Games, Cognitive Engineering, Math Information Retrieval
- ▷ 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)



©: Michael Kohlhase

29



Part I

Getting Started with AI: A Conceptual Framework

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

After having seen what AI can do and where AI is being employed today (see Section 19.3), will now

1. introduce a programming language to use in the course, and
2. prepare a conceptual framework in which we can think about “intelligence” (natural and artificial).

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

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

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

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

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

4.1 Introduction to Logic Programming and PROLOG

Logic Programming is a programming style that differs from functional and imperative programming in the basic procedural intuition. Instead of transforming the state of the memory by issuing instructions (as in imperative programming), or computing the value of a function on some arguments, logic programming interprets the program as a body of knowledge about the respective situation, which can be queried for consequences. This is actually a very natural intuition; after all we only run (imperative or functional) programs if we want some question answered.

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 4.1.1**

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



Writing ProLog Programs via Facts and Rules

- ▷ Express program knowledge in ProLog as **Prolog terms** made of
 - ▷ **Prolog constants** denoted by lower-case strings
 - ▷ **Prolog variables** denoted by upper-case strings or starting with underscore
 - ▷ **Prolog functions** and **Prolog predicates** (lowercase strings) applied to Prolog terms.
- ▷ **Example 4.1.2** The following are Prolog terms: john, X, _, father(john), loves(john,mary), loves(john,_),...
- ▷ **Definition 4.1.3** A **Prolog program** consists of a sequence of **Prolog clauses**, i.e.
 - ▷ **Prolog facts** of the form *t*. (a term and a dot)
 - ▷ **Prolog rules** of the form *h:-b₁,...,b_n*, where *h* is called the **head literal** (or simply **head**) and the *b_i* are called the **body literals** of the rule.
- ▷ **Example 4.1.4** The following is a Prolog program:


```
human(leibniz).
human(sokrates).
greek(sokrates).
fallible(X):-human(X).
```
- ▷ **Definition 4.1.5** The **knowledge base** given by a Prolog program is the set of facts that can be derived from it.



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

Querying the Knowledge base

- ▷ **Idea:** We want to see whether a term is in the knowledge base.
- ▷ **Definition 4.1.6** A **query** is a list of ProLog terms called **goals**. Write as *?- A₁, ..., A_n.*, if *A_i* are terms.
- ▷ **Problem:** Knowledge bases can be big and even infinite.
- ▷ **Example 4.1.7** The knowledge base induced by the program

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

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

▷ **Idea:** interpret this as a search problem.

- ▷ state = tuple of goals; goal state = empty list (of goals).
- ▷ $\text{next}(\langle G, R_1, \dots, R_l \rangle) := \langle \sigma(B_1), \dots, \sigma(B_m, R_1, \dots, R_l) \rangle$ (**backchaining**) if there is a rule $H:-B_1, \dots, B_m$. and a substitution σ with $\sigma(H) = \sigma(G)$.

▷ **Example 4.1.8 (Continuing)**

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



This search process has as action the backchaining rule, which is really just a clever combination of the inference rules from `?knowledge-base.def?` applied backwards.

The backchaining rule takes the current goal G (the first one) and tries to find a substitution σ and a head H in a program clause, such that $\sigma(G) = \sigma(H)$. If such a clause can be found, it replaces G with the body of the rule instantiated by σ .

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

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

Querying the Knowledge base: Failure

- ▷ If no instance of the statement in a query can be derived from the knowledge base, then the ProLog interpreter reports failure.

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



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

Querying the Knowledge base: Answer Substitutions

▷ If a query contains variables, then ProLog will return an **answer substitution**.

▷

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



In the `?query-vars-ex?` the first backchaining step binds the variable `X` to the query variable `Y`, which gives us the two subgoals `has_wheels(Y,4),has_motor(Y)`. which again have the query variable `Y`. The next backchaining step binds this to `mybmw`, and the third backchaining step exhausts the subgoals. So the query succeeds with the (overall) substitution `[mybmw/Y]`, which is reported by the ProLog interpreter as `Y = mybmw`.

With this setup, we can already do the “fallible Greeks” example from the introduction.

PROLOG: Are there Fallible Greeks?

▷ **Program:**

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

▷ **Example 4.1.9 (Query)** `?- fallible(X),greek(X).`

▷ **Answer substitution:** `[sokrates/X]`



4.2 Programming as Search

In this Section, we want to really use ProLog as a programming language, so let's first get our tools set up.

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

We will introduce working with the interpreter using unary natural numbers as examples: we first add the fact¹ to the knowledge base

```
unat(zero).
```

¹for “unary natural numbers”; we cannot use the predicate `nat` and the constructor function `s` here, since their meaning is predefined in ProLog

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

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

or by directly typing

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

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

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

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

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

Running ProLog in an `emacs` window is incredibly nicer than at the command line, because you can see the whole history of what you have done. Its better for debugging too. If you’ve never used `emacs` before, it still might be nicer, since its pretty easy to get used to the little bit of `emacs` that you need. (Just type “`emacs &`” at the UNIX command line to run it; if you are on a remote terminal like `putty`, you can use “`emacs -nw`”).

If you don’t already have a file in your home directory called “`.emacs`” (note the dot at the front), create one and put the following lines in it. Otherwise add the following to your existing `.emacs` file:

```
(autoload 'run-prolog "prolog" "Start a Prolog sub-process." t)
  (autoload 'prolog-mode "prolog" "Major mode for editing Prolog programs." t)
  (setq prolog-program-name "swipl"); or whatever the prolog executable name is
  (add-to-list 'auto-mode-alist '("^\pl\$" . prolog-mode))
```

The file `prolog.el`, which provides `prolog-mode` should already be installed on your machine, otherwise download it at <http://turing.ubishops.ca/home/bruda/emacs-prolog/>

Now, once you’re in `emacs`, you will need to figure out what your “meta” key is. Usually its the alt key. (Type “control” key together with “h” to get help on using `emacs`). So you’ll need a “meta-X” command, then type “`run-prolog`”. In other words, type the meta key, type “x”, then there will be a little window at the bottom of your `emacs` window with “`M-x`”, where you type `run-prolog`³. This will start up the SWI ProLog interpreter, . . . et voilà!

The best thing is you can have two windows “within” your `emacs` window, one where you’re editing your program and one where you’re running ProLog. This makes debugging easier.

Depth-First Search with Backtracking

▷ So far, all the examples led to direct success or to failure. (simpl. KB)

▷ **Definition 4.2.1 (ProLog Search Procedure)** top-down, left-right depth-first (`backtracking`) 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`.

²for “unary natural numbers”.

³Type “control” key together with “h” then press “m” to get an exhaustive mode help.

▷ 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 ;.



Note: We have seen Definition 7.3.2 that depth-first search has the problem that it can go into loops. And in fact this is a necessary feature and not a bug for a programming language: we need to be able to write non-terminating programs, since the language would not be Turing-complete otherwise. The argument can be sketched as follows: we have seen that for Turing machines the [halting problem](#) is undecidable. So if all ProLog programs were terminating, then ProLog would be weaker than Turing machines and thus not [Turing complete](#).

We will now fortify our intuition about the ProLog search procedure by an example that extends the setup from [?query-vars-ex?](#) by a new choice of a vehicle that could be a car (if it had a motor).

Backtracking by Example

```

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

```



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

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

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

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

Can We Use This For Programming?

- ▷ **Question:** What about functions? E.g. the addition function?
- ▷ **Question:** We do not have (binary) functions, in ProLog
- ▷ **Idea (back to math):** use a three-place predicate.

Example 4.2.2 `add(X,Y,Z)` stands for $X+Y=Z$

- ▷ Now we can directly write the recursive equations $X + 0 = X$ (base case) and $X + s(Y) = s(X + Y)$ into the knowledge base.

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

- ▷ similarly with multiplication and exponentiation.

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



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

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

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

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

More Examples from elementary Arithmetics

▷ **Example 4.2.3** 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 since “generate-and-test approach”)

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

▷ **Example 4.2.4** 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 4.2.5** 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.
```



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

We will now add the primitive data structure of lists to ProLog. They are very similar to lists in SML, only the concrete syntax is different. Just as in SML lists are constructed by prepending an element (the head) to an existing list (which becomes the rest list of the constructed one).

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).
```



Just as in SML, we can define list operations by recursion, only that we program with relations instead of with functions.

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

Relational Programming Techniques

- ▷ Parameters have no unique direction “in” or “out”

```
:— rev(L,[1,2,3]).  
:— rev([1,2,3],L1).  
:— rev([1,X],[2,Y]).
```

- ▷ Symbolic programming by structural induction

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

- ▷ **Example 4.2.6** Generate and test

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



From a programming practice point of view it is probably best understood as “relational programming” in analogy to functional programming, with which it shares a focus on recursion.

The major difference to functional programming is that relational programming does not have a fixed input/output distinction, which makes the control flow in functional programs very direct and predictable. Thanks to the underlying search procedure, we can sometime make use of the flexibility afforded by logic programming.

If the problem solution involves search (and depth-first search is sufficient), we can just get by with specifying the problem and letting the ProLog interpreter do the rest. In Example 4.2.6 we just specify that list `Xs` can be sorted into `Ys`, iff `Ys` is a permutation of `Xs` and `Ys` is ordered. Given a concrete (input) list `Xs`, the ProLog interpreter will generate all permutations of `Ys` of `Xs` via the predicate `perm/2` and then test them whether they are ordered.

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

It is easy to see that the runtime of the ProLog program from Example 4.2.6 is not $\mathcal{O}(n \log_2(n))$ which is optimal for sorting algorithms. This is the flip-side of the flexibility in logic programming. But ProLog has ways of dealing with that: the **cut operator**, which is a ProLog atom, which always succeeds, but which cannot be backtracked over. This can be used to prune the search tree in ProLog. We will not go into that here but refer the readers to the literature.

Chapter 5

Recap: Complexity Analysis in AI?

We now come to an important topic which is not really part of Artificial Intelligence but which adds an important layer of understanding to this enterprise: We (still) live in the era of Moore's law (the computing power available on a single CPU doubles roughly every two years) leading to an exponential increase. A similar rule holds for main memory and disk storage capacities. And the production of computers (using CPUs and memory) is (still) very rapidly growing as well; giving mankind as a whole, institutions, and individual exponentially grow of computational resources.

This development is often cited as the reason why (strong) AI is inevitable in public discussion. But the argument is fallacious if all the algorithms we have are of very high complexity (i.e. at least exponential in either time or space). So, to judge the state of play in Artificial Intelligence, we have to know the complexity of our algorithms.

In this Chapter, we will give a very brief recap of some aspects of elementary complexity theory and make a case of why this is a generally important for computer scientists.

In order to get a feeling what we mean by "fast algorithm", we do some preliminary computations.

Performance and Scaling

- ▷ Suppose we have three algorithms to choose from. (which one to select)
- ▷ Systematic analysis reveals performance characteristics.
- ▷ **Example 5.0.1** For a problem of size n (i.e., detecting cycles out of n nodes) we have

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

What?! One year?

$$\triangleright 2^{10} = 1\,024 \quad (1024\mu s \simeq 1\text{ms})$$

$$\triangleright 2^{45} = 35\,184\,372\,088\,832 \quad (3.5 \times 10^{13}\mu s = 3.5 \times 10^7\text{s} \simeq 1.1Y)$$

\triangleright **Example 5.0.2** we denote all times that are longer than the age of the universe with –

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



So it does make a difference for larger problems what algorithm we choose. Considerations like the one we have shown above are very important when judging an algorithm. These evaluations go by the name of complexity theory.

Let us now recapitulate some notions of elementary complexity theory: we are interested in the worst case growth of the resources (time and space) required by an algorithm in terms of the sizes of its arguments. Mathematically we look at the functions from input size to resource size and classify them into “big-O” classes, abstracting from constant factors (which depend on the machine the algorithm runs on and which we cannot control) and initial (algorithm startup) factors.

Recap: Time/Space Complexity of Algorithms

\triangleright We are mostly interested in worst-case complexity in AI-1

\triangleright **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 $s \in S$.

\triangleright time/space complexity depends on size measures. (no canonical one)

\triangleright **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.
[\(next\)](#)



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

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 $T(\psi) \in F$, 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 $T(\psi) \in F$, 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 $T(\alpha) \in \mathcal{O}(f(n) \cdot g(n))$



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

Why Complexity Analysis? (General)

- ▷ **Example 5.0.3** Once upon a time I was trying to invent an efficient algorithm for some problem.
 - ▷ 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.



▷ Ah ba, this is overdoing it.



Why Complexity Analysis? (General)

▷ **Example 5.0.4** 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.



©: Michael Kohlhase

48



It's like, you're trying to find a route to India (from Spain), and you presume it's somewhere to the east, and then you hit a coast, but no; try again, but no; try again, but no; ... if you don't have a map, that's the best you can do. But NP-hardness gives you the map: you can check that there actually is no way through here.

But what is this notion of NP-completeness alluded to above? We observe that we can analyze the complexity of problem classes by the complexity of the algorithms that solve them. This gives us a notion of what to expect from solutions to a given problem class, and thus whether efficient (i.e. polynomial) algorithms can exist at all.

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

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



©: Michael Kohlhase

49



Questionnaire

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



©: Michael Kohlhase

50



Chapter 6

Intelligent Agents: a Unifying Framework for AI

6.1 Introduction: Rationality in Artificial Intelligence

What is AI?

▷ What is AI?: Four possible answers:

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



©: Michael Kohlhase

51



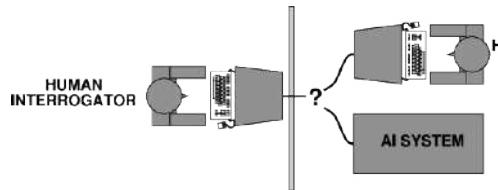
So, what does modern AI do?

- ▷ 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



Acting humanly: The Turing test

- ▷ Alan Turing (1950) “Computing machinery and intelligence” [Tur50]:
 - ▷ “Can machines think?” → “Can machines behave intelligently?”
 - ▷ **Definition 6.1.1** 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 6.1.2 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 6.1.3 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!



©: Michael Kohlhase

54



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?



©: Michael Kohlhase

55



Acting rationally

- ▷ **Definition 6.1.4 Rational** behavior: doing the right thing
- ▷ The right thing: that which is expected to maximize goal achievement (*given the available information*)

- ▷ Doesn't necessarily involve thinking — e.g., blinking reflex — but thinking should be in the service of rational action
- ▷ Aristotle (Nicomachean Ethics): *Every art and every inquiry, and similarly every action and pursuit, is thought to aim at some good*



Rational agents

- ▷ **Definition 6.1.5** An **agent** is an entity that perceives and acts.
- ▷ This course is about designing **rational agents**
- ▷ **Definition 6.1.6** Abstractly, an agent is a function from percept histories to actions:

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

For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance

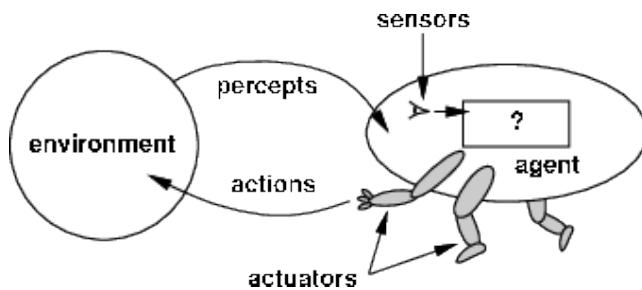
- ▷ Caveat: *computational limitations make perfect rationality unachievable*
→ design best **program** for given machine resources



6.2 Agents and Environments as a Framework for AI

Agents and Environments

- ▷ **Definition 6.2.1** 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 6.2.2** Agents include humans, robots, softbots, thermostats, etc.



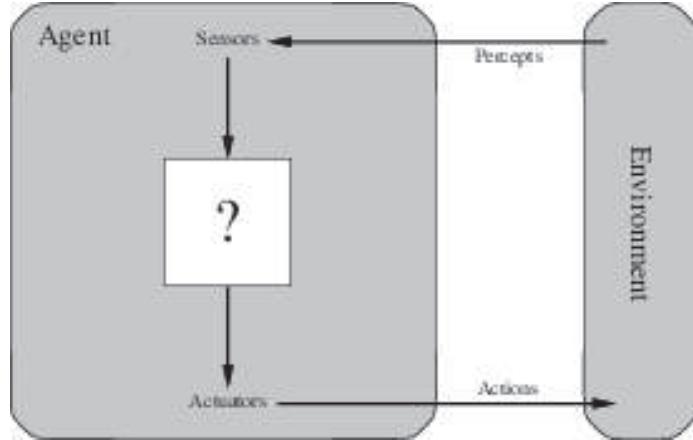
Modeling Agents Mathematically and Computationally

- ▷ **Definition 6.2.3** A **percept** is the perceptual input of an agent at a specific instant.
 - ▷ **Definition 6.2.4** Any recognizable, coherent employment of the actuators of an agent is called an **action**.
 - ▷ **Definition 6.2.5** 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 6.2.6** An agent function can be implemented by an **agent program** that runs on a physical **agent architecture**.



Agent Schema: Visualizing the Internal Agent Structure

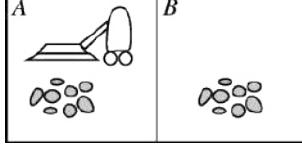
- ▷ **Agent Schema:** We will use the following kind of schema to visualize the internal structure of an **agents**:



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

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

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



Example: Vacuum-Cleaner World and Agent

- ▷ Example 6.2.7 (Agent Program)

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



Table-Driven Agents

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

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

Problem: Why is this not a good idea?

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



6.3 Good Behavior \sim Rationality

Rationality

- ▷ **Idea:** Try to design agents that are successful (aka. "do the right thing")
- ▷ **Definition 6.3.1** A **performance measure** is a function that evaluates a sequence of environments.
- ▷ **Example 6.3.2** 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 6.3.3** An agent is called **rational**, if it chooses whichever action maximizes the expected value of the performance measure given the perceptual 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 \sim exploration, learning, autonomy
 - ▷ **Definition 6.3.4** 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 6.3.5** 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 6.3.6 (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



Questionnaire: Think about the following

- ▷ **Question 1:** Which are agents?

- (A) James Bond.
- (B) Your dog.
- (C) Vacuum cleaner.
- (D) Thermometer.

▷ **Question 2:** Who is rational?

- (A) James Bond, crossing the street without looking.
- (B) Your dog, crossing the street without looking.
- (C) Vacuum cleaner, deciding to clean under your bed.
- (D) Thermostat, deciding to cool down your fridge.



Questionnaire: Answers

▷ **Answer 1:** Which are agents?

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

▷ **Answer 2:** Who is rational?

- (A) : Depends on whether safety is part of his performance measure.
- (B) : Depends on whether or not we consider dogs to be able to check the traffic. If they can't, then just running over could be optimal (e.g. to meet fellow dogs or grab a sausage).
- (C) : Yes. (Hypothetical best-case if it's dirty under your bed, and you're not currently sleeping in it.)
- (D) : Not clear whether a thermostat is an agent. On the one hand, in difference to the Thermometer, the Thermostat takes an action. On the other hand, in a classical Thermostat the "action decision" is just a physical reaction (like a solar panel that produces electricity if the sun shines).



6.4 Classifying Environments

Environment types

▷ **Observation 6.4.1** Agent design is largely determined by environment type.

▷ **Problem:** There is a vast number of possible environments in AI.

- ▷ **Solution:** Classify along a handful of “dimensions” (independent characteristics)
- ▷ **Definition 6.4.2** For an agent a we call an environment e
 - ▷ **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**.
 - ▷ **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**.
 - ▷ **discrete**, iff the sets of e 's **states** and a 's actions are countable, else **continuous**.
 - ▷ **single-agent**, iff only a acts on e (when must we count parts of e as agents?)



©: Michael Kohlhase

70



Some examples will help us understand this better.

Environment Types (Examples)

- ▷ **Example 6.4.3** Some environments classified:

	Solitaire	Backgammon	Internet shopping	Taxi
observable	Yes	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 6.4.4** *The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, and multi-agent (worst case for AI)*



©: Michael Kohlhase

71



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

6.5 Types of Agents

We will now discuss the main types of agents we will encounter in this course, get an impression

of the variety, and what they can and cannot do. We will start from [simple reflex agents](#), add state, and utility, and finally add learning.

Agent types

- ▷ **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)
- ▷ 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. [reflex agents with state](#)
 3. [goal-based agents](#)
 4. [utility-based agents](#)
- ▷ All these can be turned into [learning agents](#)



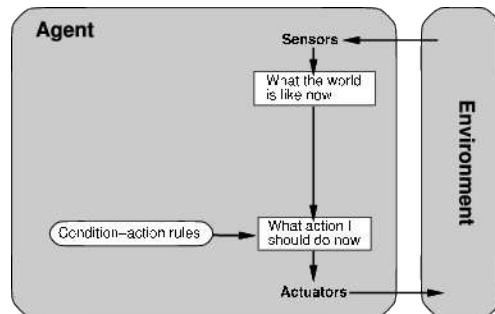
©: Michael Kohlhase

72



Simple reflex agents

- ▷ **Definition 6.5.1** 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 6.5.2**

```

procedure Reflex-Vacuum-Agent [location,status] returns an action if status =  

  Dirty then ...
  
```



©: Michael Kohlhase

73



Simple reflex agents (continued)

▷ General Agent Program:

```
function Simple-Reflex-Agent (percept) returns an action
  persistent: rules (* a set of condition-action rules*)

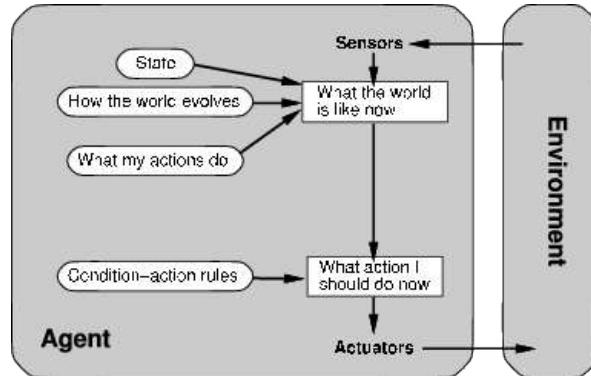
  state := Interpret-Input(percept)
  rule := Rule-Match(state,rules)
  action := Rule-action[rule]
  return action
```

- ▷ Problem: simple reflex agents can only react to the perceived state of the environment, not to changes.
- ▷ Example 6.5.3 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 6.5.4 Vacuum cleaner robot with defective location sensor \rightsquigarrow infinite loops.



Reflex agents with state

- ▷ Idea: Keep track of the state of the world we cannot see now in an internal model
- ▷ Definition 6.5.5 A stateful reflex agent (also called reflex agent with state or model-based agent) whose agent function depends on a model of the world (called the world model).
- ▷ Agent Schema:



Reflex agents with state (continued)

- ▷ **Observation 6.5.6** The agent program for a stateful reflex agent is of the following form:

```
function Stateful–Reflex–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
```

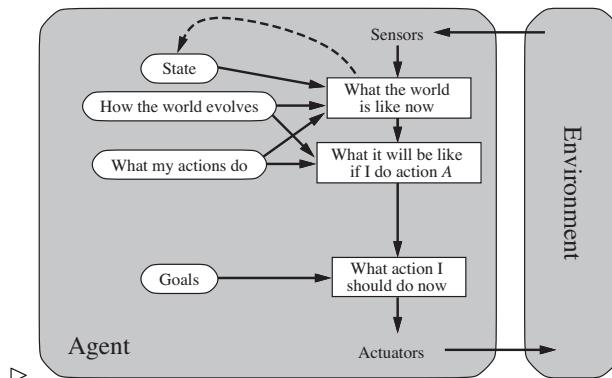
Problem: Having a model of the world does not always determine what to do (rationally)

- ▷ **Example 6.5.7** 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 6.5.8** A **goal-based agent** is a stateful reflex 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.

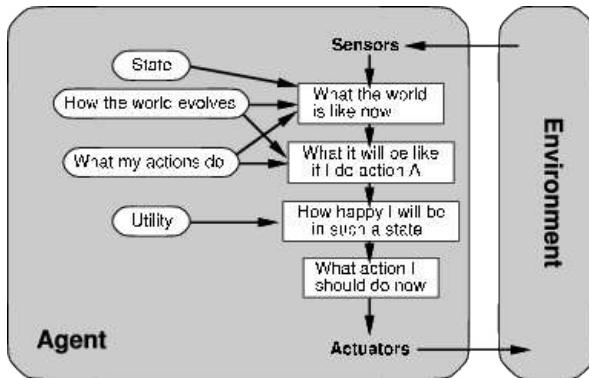
- ▷ **Example 6.5.9** A goal-based agent can easily be changed to go to a new destination, a reflex agent's rules make it go to exactly one destination.



Utility-based agents

- ▷ **Definition 6.5.10** 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 agents

- ▷ 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)



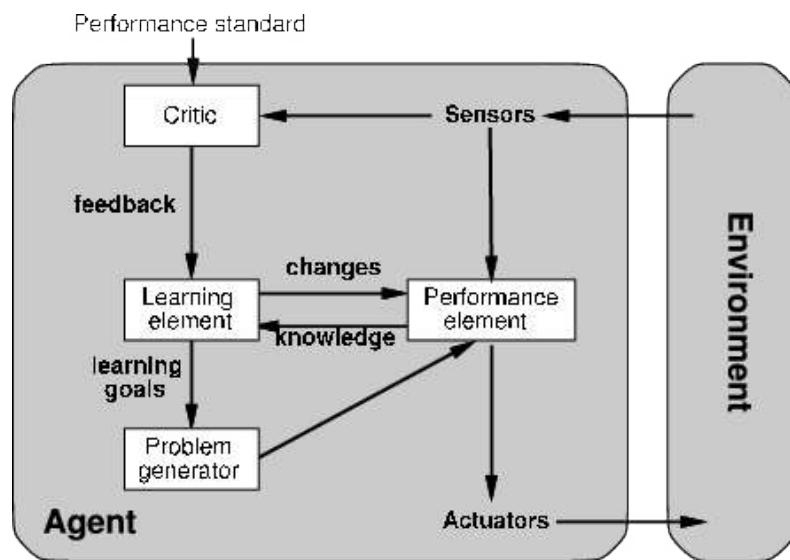
Learning Agents

- ▷ **Definition 6.5.11** 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 knowledge
 - ▷ a **critic** which gives feedback to the **learning element** based on an external **performance standard**

- ▷ a **problem generator** which suggests actions that lead to new and informative experiences.
- ▷ The performance element is what we took for the whole agent above.

Learning Agents

- ▷ **Agent Schema:**



Learning Agents: Example

- ▷ **Example 6.5.12 (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	▷	
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?



6.6 Representing the Environment in Agents

We now come to a very important topic, which has a great influence on agent design: how does the agent represent the environment. After all, in all agent designs above (except the simple reflex agent) maintain a notion of world state and how the world state evolves given percepts and actions. The form of this model determines the algorithms

Representing the Environment in Agents

- ▷ We have seen various components of agents that answer questions like
 - ▷ *What is the world like now?*
 - ▷ *What action should I do now?*
 - ▷ *What do my actions do?*

Next natural question: How do these work? (see the rest of the course)

- ▷ **Important Distinction:** How the agent represents the environment it is in (*states*)

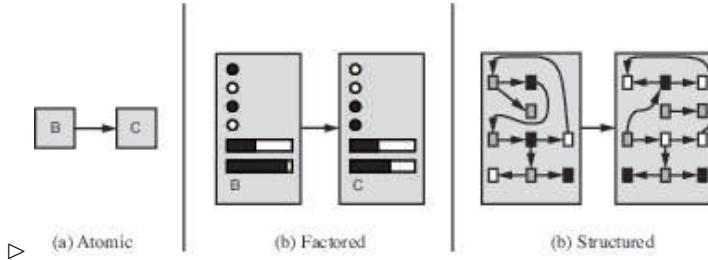
- ▷ **Definition 6.6.1** We call a *state* representation

▷ *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 6.6.2 (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 “TruckAheadBackingIntoDairyFarmDrivewayBlocked-ByLooseCow” is unlikely)
 - ▷ In a **structured** representation, we can have objects for trucks, cows, etc. and their relationships



Summary

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



Part II

General Problem Solving

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

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

Chapter 7

Problem Solving and Search

In this Chapter, we will look at a class of algorithms called search algorithms. These are algorithms that help in quite general situations, where there is a precisely described problem, that needs to be solved.

7.1 Problem Solving

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

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

Problem solving

- ▷ **Problem:** 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.
We will use the following two concepts to describe problems
 - States** A set of possible situations in our problem domain
 - Actions** A set of possible actions that get us from one state to another.Using these, we can view a sequence of actions as a solution, if it brings us into a situation, where the problem is solved.
- ▷ **Definition 7.1.1 Offline problem solving:** Acting only with complete knowledge of problem and solution
- ▷ **Definition 7.1.2 Online problem solving:** Acting without complete knowledge

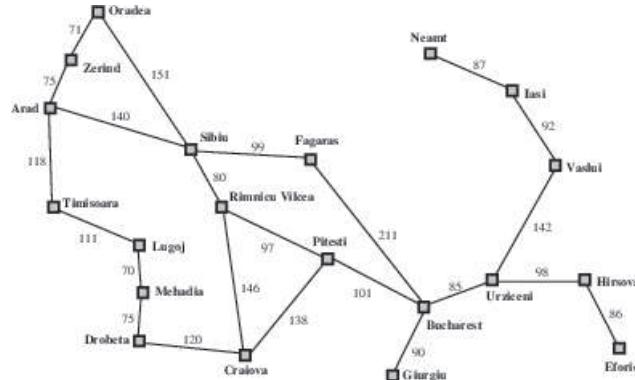
- **Here:** we are concerned with **offline** problem solving only.



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

Example: Traveling in Romania

- ▷ **Scenario:** On holiday in Romania; currently in Arad, Flight leaves tomorrow from Bucharest.
 - ▷ **Formulate problem:** *States:* various cities *Actions:* drive between cities
 - ▷ **Solution:** Appropriate sequence of cities, e.g.: Arad, Sibiu, Fagaras, Bucharest



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

Problem Formulation

- ▷ The problem formulation models the situation 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. (excludes, e.g. to stay another couple of weeks.)
 - ▷ Finding the right level of abstraction and the required (not more!) information is often the key to success.
 - ▷ **Definition 7.1.3** A search problem $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ consists of a set \mathcal{S} of states, a set \mathcal{A} of actions, and a transition model $\mathcal{T}: \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$ 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 there is a unique initial state $\mathcal{I} \in \mathcal{S}$.

- ▷ **Definition 7.1.4** 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 .
- ▷ We will restrict ourselves to transition models, where $|\mathcal{T}(a, s)| \leq 1$ for the moment. Then \mathcal{T}_a is a **partial function** whose domain is the set of states where a is applicable.
- ▷ **Definition 7.1.5** The predicate that tests for goal states is called a **goal test**.
- ▷ **Definition 7.1.6** A **solution** for a problem \mathcal{P} consists of a sequence of actions that bring us from \mathcal{I} to a goal state.
- ▷ **Definition 7.1.7** Often we add a **cost function** $c: \mathcal{A} \rightarrow \mathbb{R}_0^+$ that associates a **step cost** $c(a)$ to an action $a \in \mathcal{A}$.



Observation: The **problem** of problems from Definition 7.1.3 is essentially a “black-box” description that provides the functionality needed to construct the state space.

Blackbox/Declarative Problem Descriptions

- ▷ **Observation:** $\langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ from Definition 7.1.3 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 7.1.8** A **declarative description** (also called **whitebox description**) describes the problem itself \rightsquigarrow **problem description language**
- ▷ **Example 7.1.9** 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 7.1.10** *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 \rightsquigarrow planning.



¹EDNOTE: mark up strips and create appropriate references

Note that this definition is very general, it applies to many many problems. So we will try to characterize these by difficulty.

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)



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

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

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

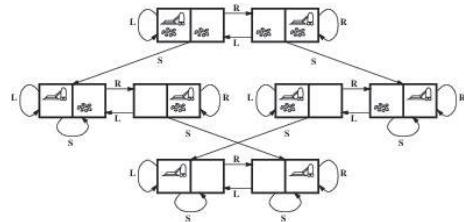
$$[right(\{2, 4, 6, 8\}), suck(\{4, 8\}), left(\{3, 7\}), suck(\{7\})]$$

Example: vacuum-cleaner world

▷ Single-state Problem:

- ▷ Start in 5

▷ Solution: $[right, suck]$



▷ Multiple-state Problem:

- ▷ Start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$

- ▷ **Solution:** $[right, suck, left, suck]$
- | | |
|--------------|------------------------------|
| <i>right</i> | $\rightarrow \{2, 4, 6, 8\}$ |
| <i>suck</i> | $\rightarrow \{4, 8\}$ |
| <i>left</i> | $\rightarrow \{3, 7\}$ |
| <i>suck</i> | $\rightarrow \{7\}$ |



Example: vacuum-cleaner world (continued)

- ▷ **Contingency Problem:**

▷ Murphy's Law: *suck* can dirty a clean carpet

▷ Local sensing: *dirty / notdirty* at location only

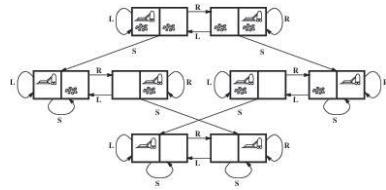
▷ Start in: $\{1, 3\}$

▷ **Solution:** $[suck, right, suck]$

suck $\rightarrow \{5, 7\}$

right $\rightarrow \{6, 8\}$

suck $\rightarrow \{6, 8\}$



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



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

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

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

Single-state problem formulation

- ▷ Defined by the following four items

1. **Initial state:** (e.g. *Arad*)
2. **Successor function S :** (e.g. $S(Arad) = \{\langle goZer, Zerind \rangle, \langle goSib, Sibiu \rangle, \dots\}$)
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 operators leading from the initial state to a goal state



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

Selecting a state space

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



©: Michael Kohlhase

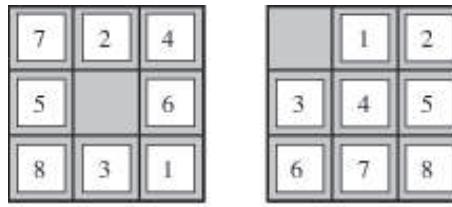
96



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

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

Example: The 8-puzzle



Start State

Goal State

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



©: Michael Kohlhase

97

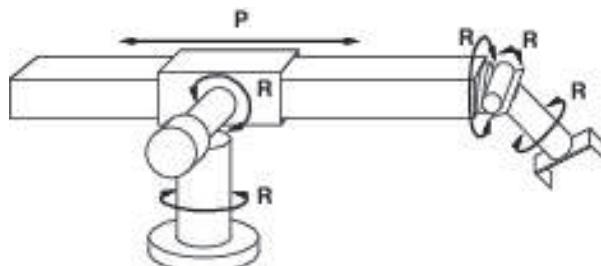


How many states are there? N factorial, so it is not obvious that the problem is in **NP**. One needs to show, for example, that polynomial length solutions do always exist. Can be done by combinatorial arguments on state space graph (really?).

Example: Vacuum-cleaner

1		2	
3		4	
5		6	
7		8	
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

Questionnaire

▷ **Question:** Which are “Problems”?

- (A) You didn't understand any of this.
- (B) Your bus today will probably be late.
- (C) Your vacuum cleaner wants to clean your apartment.
- (D) You want to win a Chess game.

(A/B) These are problems in the natural-language use of the word, but not “problems” in the sense defined here.

- (C) Yes, presuming that this is a robot, an autonomous vacuum cleaner, and that the robot has perfect knowledge about your apartment (else, it's not a classical search problem).
- (D) That's a search problem, but not a classical search problem (because it's multi-agent). We'll tackle this kind of problem in Chapter 8



7.2 Search

Tree Search Algorithms

- ▷ **Observation:** The **state space** of a search problem $\mathcal{P} := \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ forms a **graph** $\langle \mathcal{S}, \mathcal{O} \rangle$.
- ▷ As graphs are difficult to compute with, we often compute a corresponding tree and work on that. (**standard trick in graph algorithms**)
- ▷ **Definition 7.2.1** Given a search problem $\mathcal{P} := \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, the **tree search algorithm** consists of the simulated exploration of **state space** $\langle \mathcal{S}, \mathcal{O} \rangle$ in a **search tree** formed by successively generating **successors** of already-explored states. (**Offline Algorithm**)

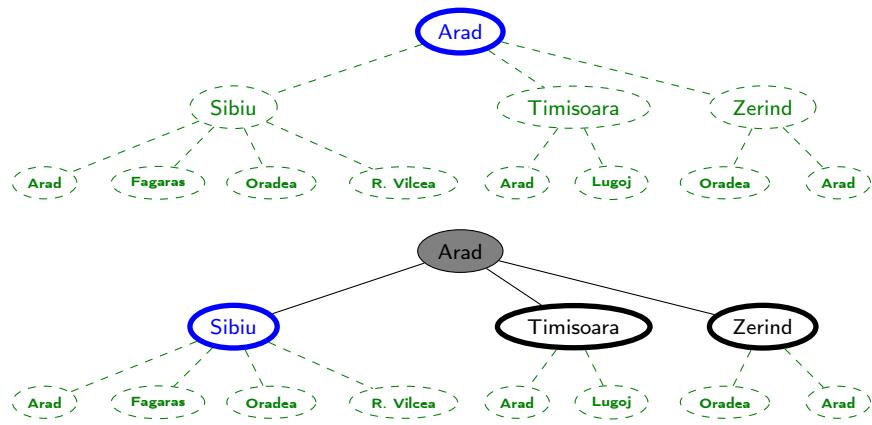
```

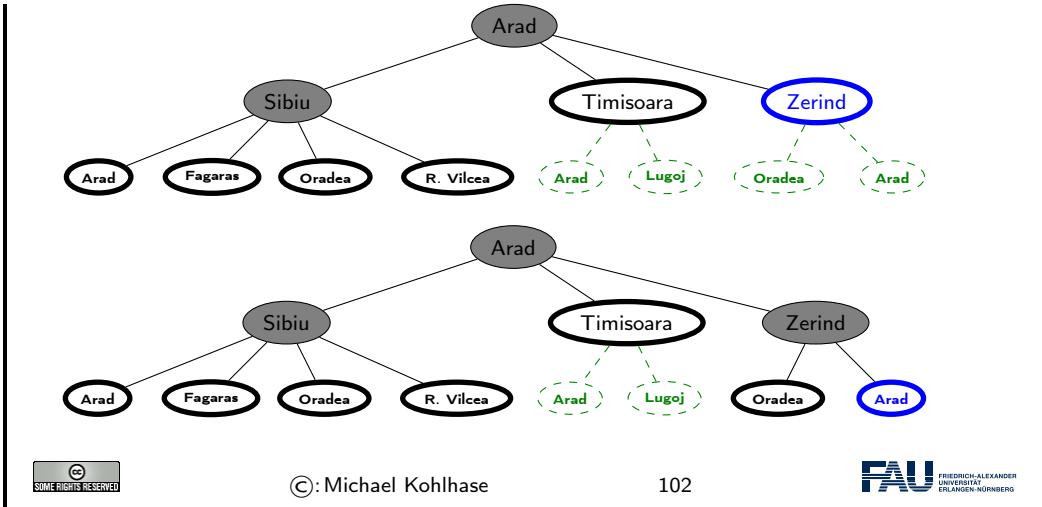
procedure Tree–Search (problem, strategy) : <a solution or failure>
    <initialize the search tree using the initial state of problem>
    loop
        if <there are no candidates for expansion> <return failure> end if
        <choose a leaf node for expansion according to strategy>
        if <the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

```



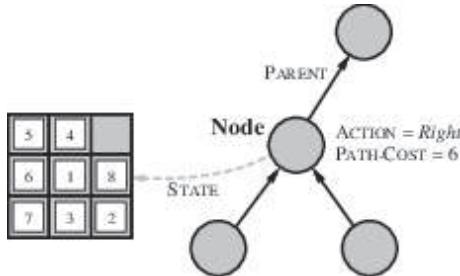
Tree Search: Example





Implementation: States vs. nodes

- ▷ **Recap:** A state is a (representation of) a physical configuration.
- ▷ 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 7.2.2** 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.

Some Rights Reserved
© Michael Kohlhase
103
FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

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 7.2.3** The *fringe* is a list nodes not yet considered. It is ordered by the *search strategy*. (see below)



STATE gives the state that is represented by *node*

EXPAND = creates new nodes by applying possible actions to *node*

A node is a data structure representing states, will be explained in a moment.

MAKE-QUEUE creates a queue with the given elements.

fringe holds the queue of nodes not yet considered.

REMOVE-FIRST returns first element of queue and as a side effect removes it from *fringe*.

STATE gives the state that is represented by *node*.

EXPAND applies all operators of the problem to the current node and yields a set of new nodes.

INSERT inserts an element into the current *fringe* queue. This can change the behavior of the search.

INSERT-ALL Perform INSERT on set of elements.

Search strategies

- ▷ **Definition 7.2.4** A *search strategy* is a function that picks a node from the fringe of a search tree. (equivalently, orders the fringe and picks the first.)

▷ **Definition 7.2.5 (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 depth of a solution in the search tree
m	maximum depth of the search tree (may be ∞)



Complexity means here always *worst-case* complexity.

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

7.3 Uninformed Search Strategies

Uninformed search strategies

- ▷ **Definition 7.3.1 (Uninformed search)** Use only 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

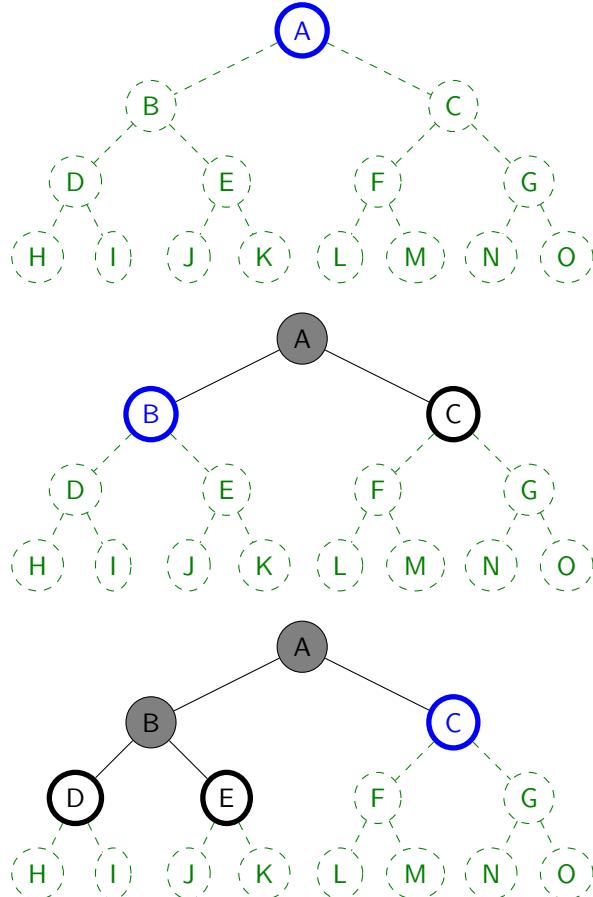


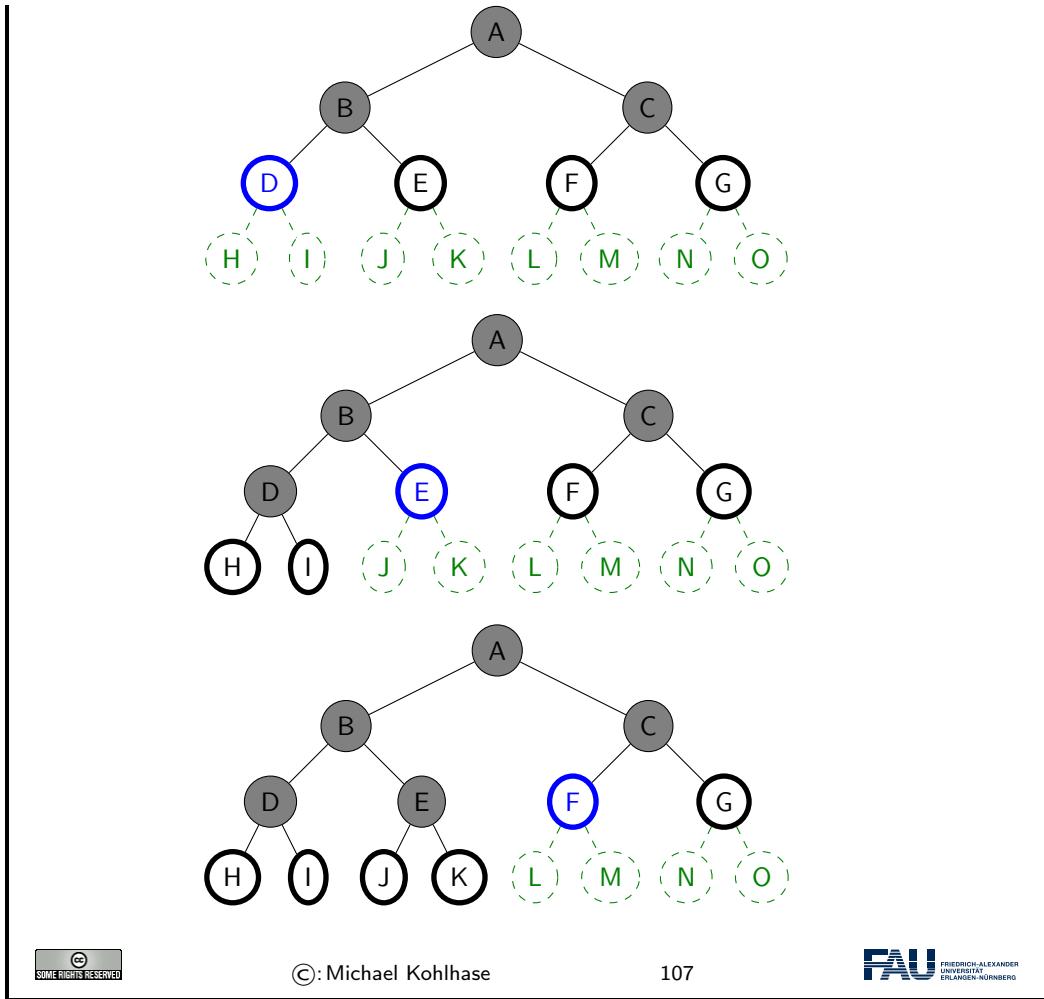
The opposite of uninformed search is informed or *heuristic* search. In the example, one could add, for instance, to prefer cities that lie in the general direction of the goal (here SE).

Uninformed search is important, because many problems do not allow to extract good heuristics.

Breadth-first search

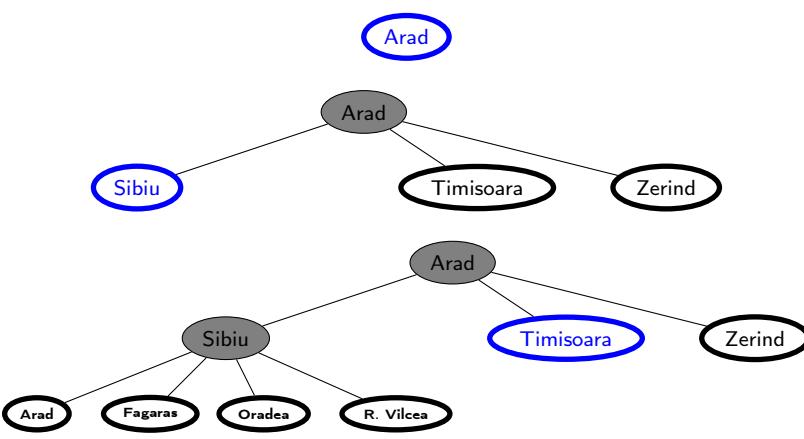
- ▷ **Idea:** Expand shallowest unexpanded node
- ▷ **Implementation:** The fringe is a FIFO queue, i.e. successors go in at the end of the queue

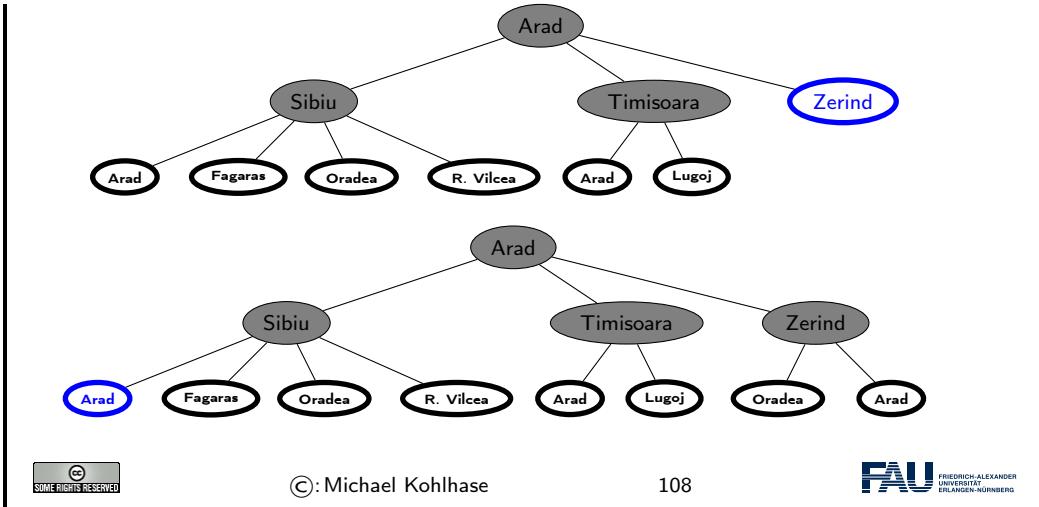




We will now apply the breadth-first search strategy to our running example: Traveling in Romania. Note that we leave out the green dashed nodes that allow us a preview over what the search tree will look like (if expanded). This gives a much cleaner picture – we assume that the readers already have grasped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

Breadth-First Search: Romania





Breadth-first search: Properties

Complete	Yes (if b is finite)
Time	$1 + b + b^2 + b^3 + \dots + b^d = b^{d+1}$, so $\mathcal{O}(b^{d+1})$ 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

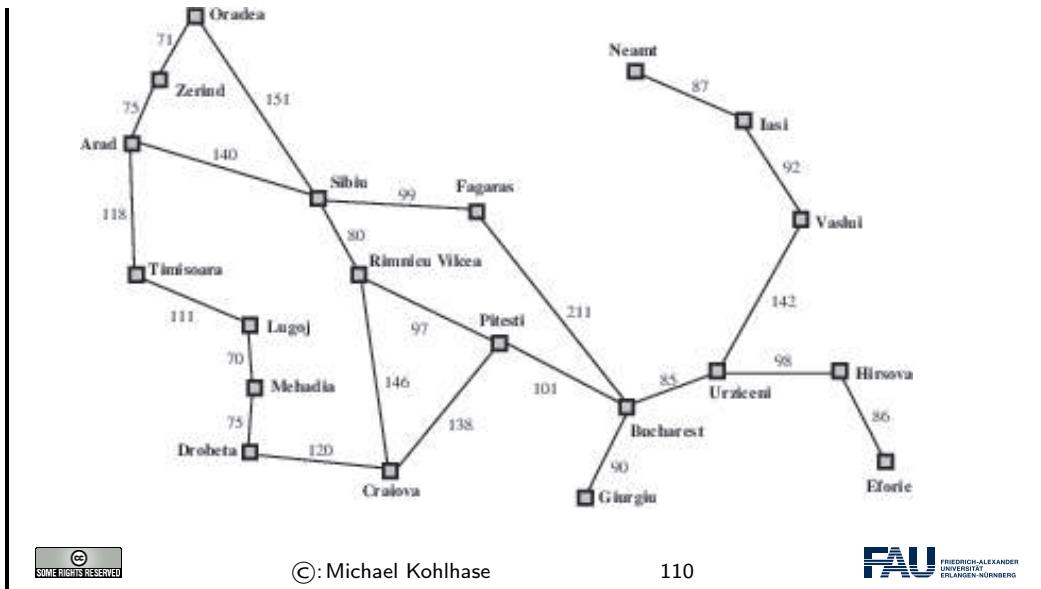
- ▷ **Disadvantage:** Space is the big problem (can easily generate nodes at 500MB/sec $\cong 1.8\text{TB/h}$)
- ▷ **Optimal?**: if cost varies for different steps, there might be better solutions below the level of the first solution.
- ▷ An alternative is to generate *all* solutions and then pick an optimal one. This works only, if m is finite.



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

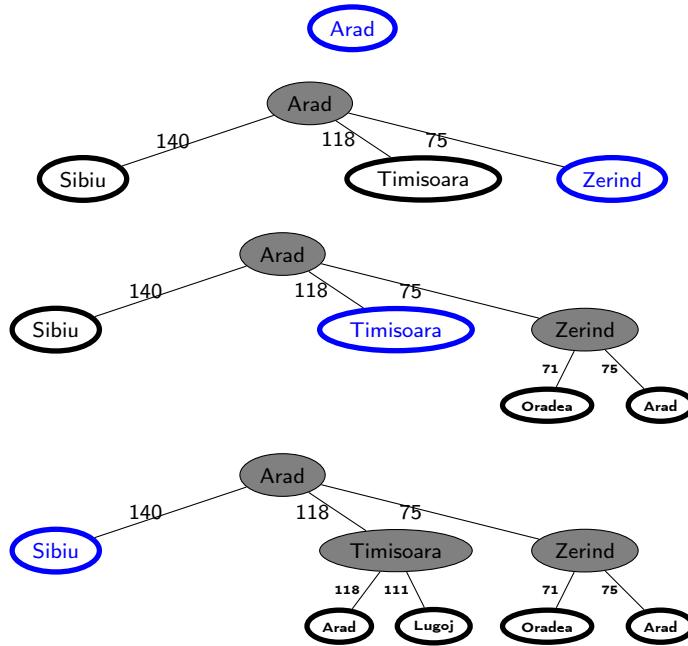
Of course we need to update our problem formulation with the necessary information.

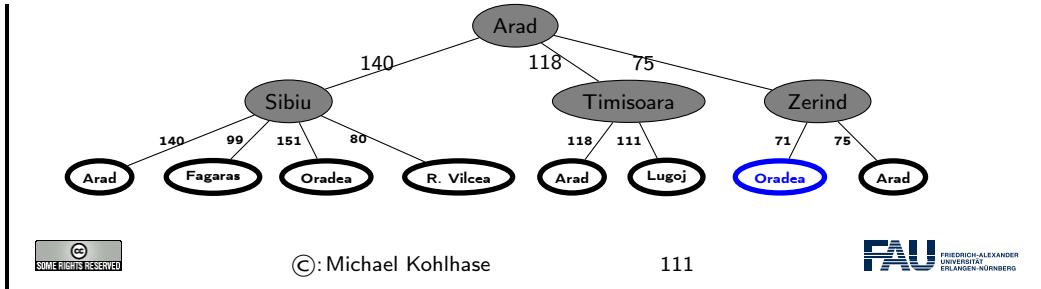
Romania with Step Costs as Distances



Uniform-cost search

- ▷ **Idea:** Expand least-cost unexpanded node
 - ▷ **Implementation:** fringe is queue ordered by increasing path cost.
 - ▷ **Note:** Equivalent to breadth-first search if all step costs are equal.





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

Uniform-cost search: Properties

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



©: Michael Kohlhase

112



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

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

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

Depth-first search

- ▷ **Idea:** Expand deepest unexpanded node
- ▷ **Definition 7.3.2 (Implementation)** **depth first search** is tree search where the fringe is organized as a LIFO queue (a stack), i.e. successors go in at front of queue
- ▷ **Note:** Depth-first search can perform infinite cyclic excursions
Need a finite, non-cyclic search space (or repeated-state checking)

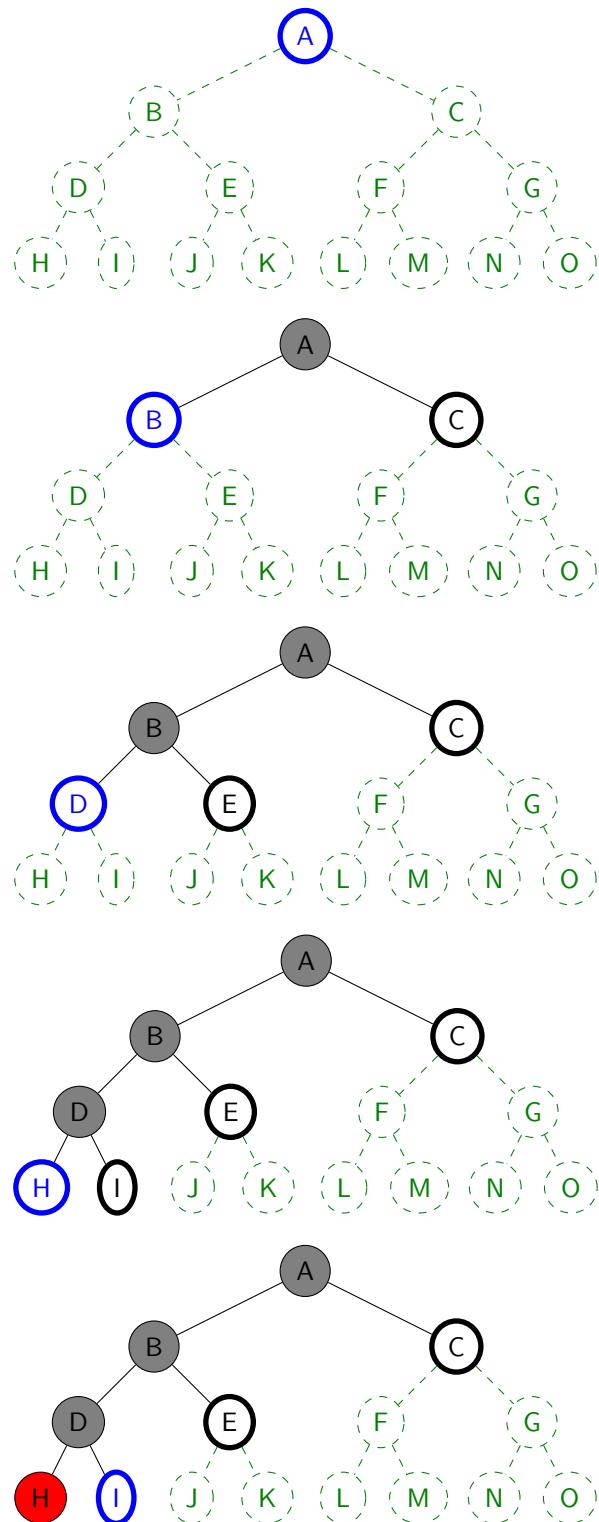


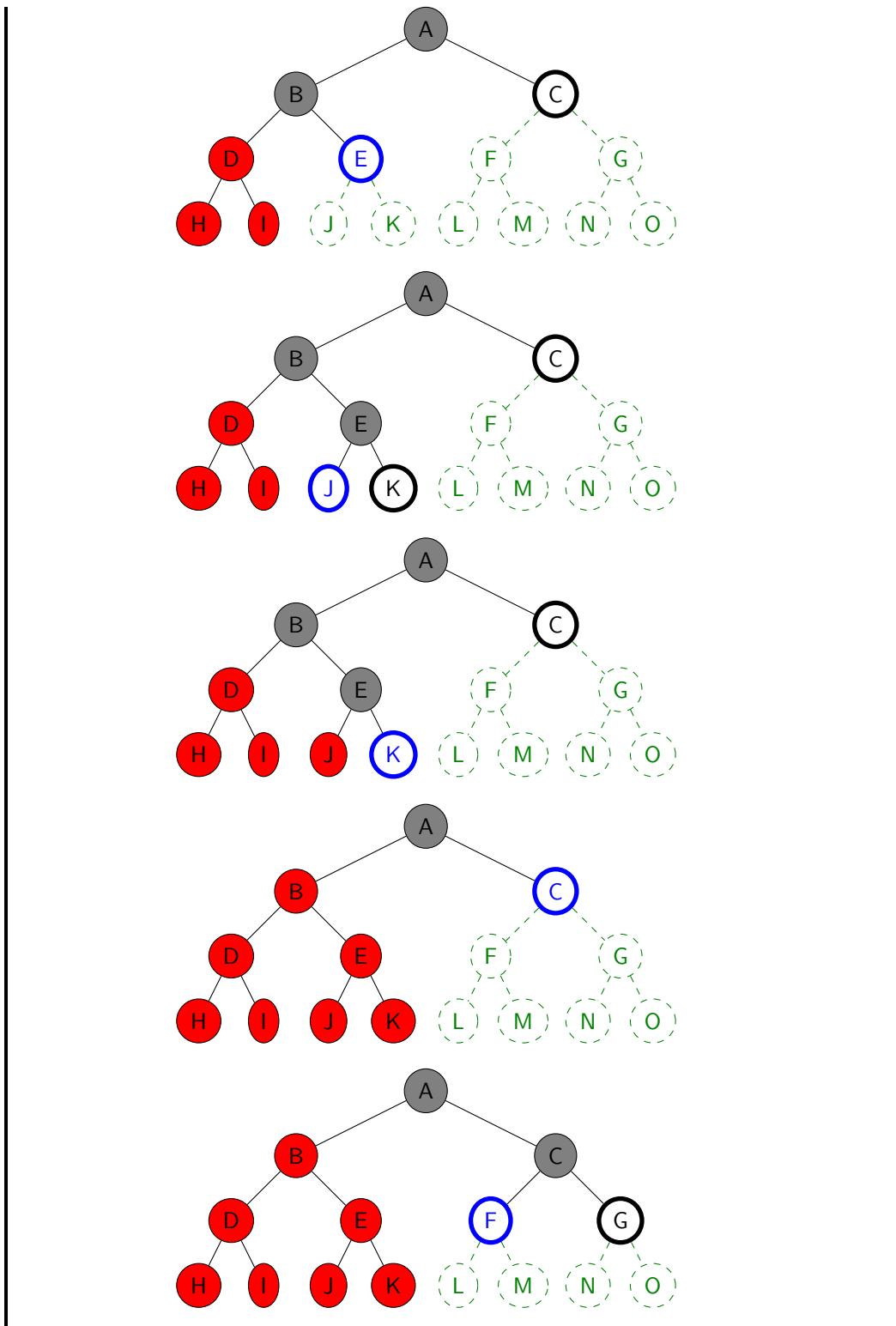
©: Michael Kohlhase

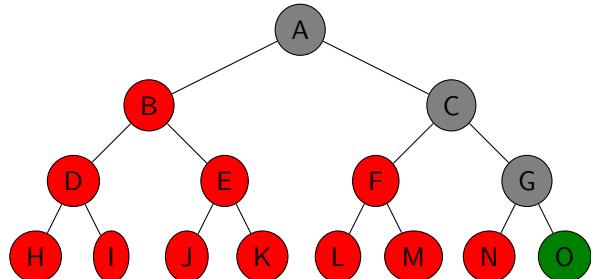
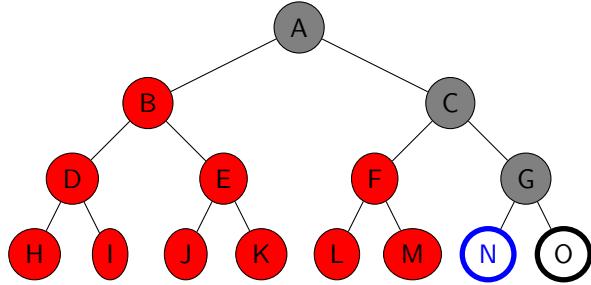
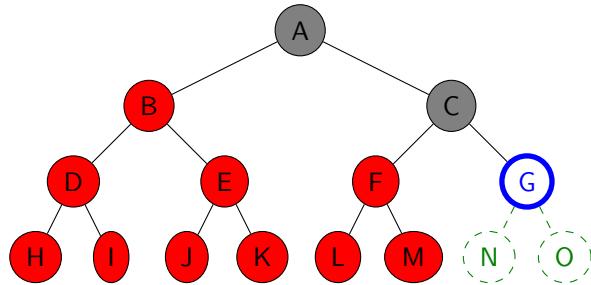
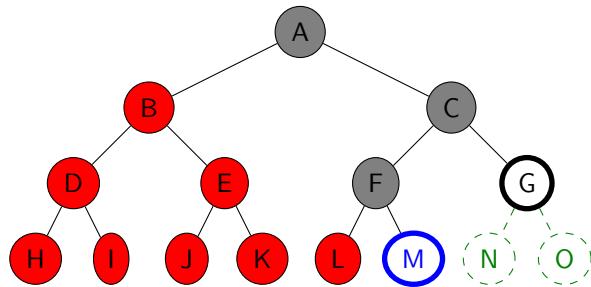
113



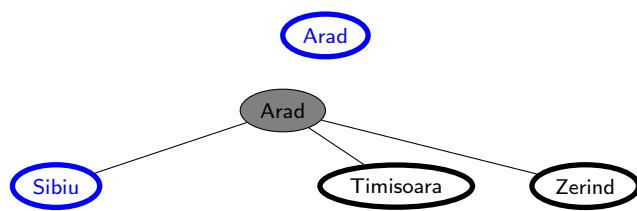
Depth-First Search

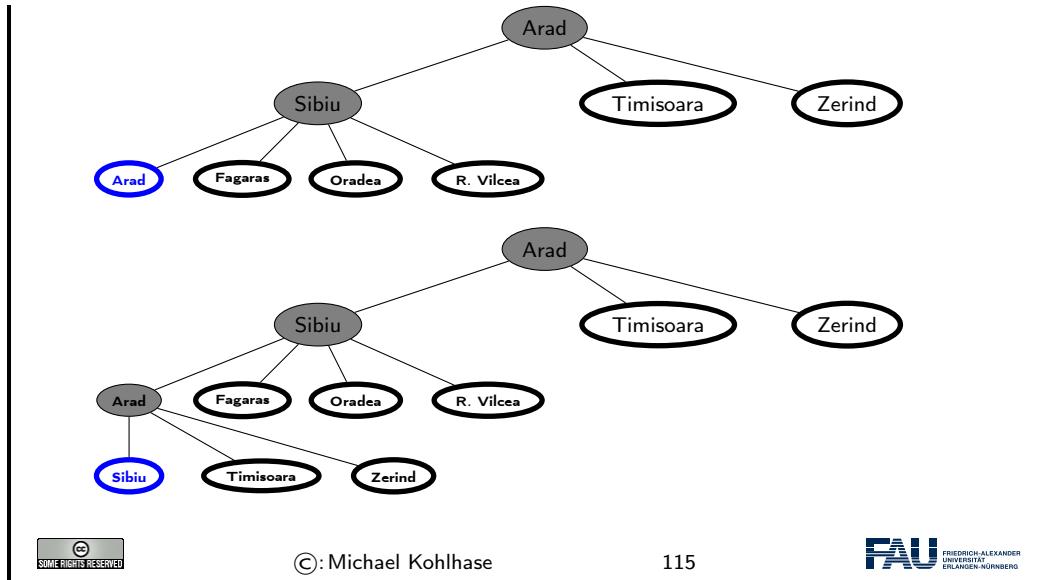






Depth-First Search: Romania





©: Michael Kohlhase

115

Depth-first search: Properties

Complete	Yes: if state space finite No: if state contains infinite paths or loops
Time	$\mathcal{O}(b^m)$ (we need to explore until max depth m in any case!)
Space	$\mathcal{O}(b m)$ (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.



©: Michael Kohlhase

116

Iterative deepening search

- ▷ **Definition 7.3.3** depth-limited search is depth-first search with a depth limit.
- ▷ **Definition 7.3.4** Iterative deepening search is depth-limited search with ever increasing limits.
- ▷ **procedure** Tree_Search (problem)

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

```

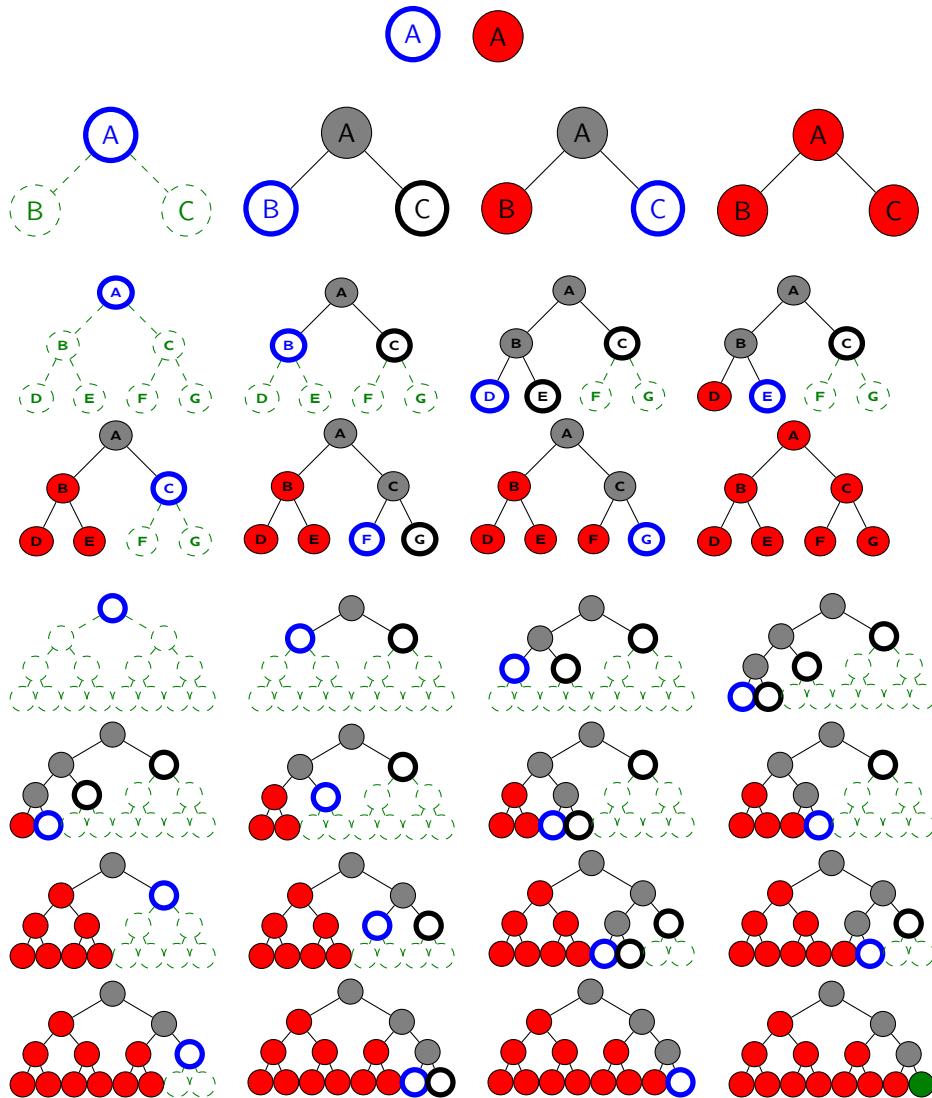
for depth = 0 to  $\infty$ 
    result := Depth_Limited_search(problem,depth)
    if depth  $\neq$  cutoff return result end if
end for

```

```
end procedure
```



Iterative Deepening Search at various Limit Depths



Iterative deepening search: Properties

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

▷ (Depth-First) Iterative-Deepening Search often used in practice for search spaces of large, infinite, or unknown depth.

Criterion	Breadth-first	Uniform-cost	Depth-first	Iterative deepening
Complete?	Yes*	Yes*	No	Yes
Time	b^{d+1}	$\approx b^d$	b^m	b^d
Space	b^d	$\approx b^d$	bm	bd
Optimal?	Yes*	Yes	No	Yes*



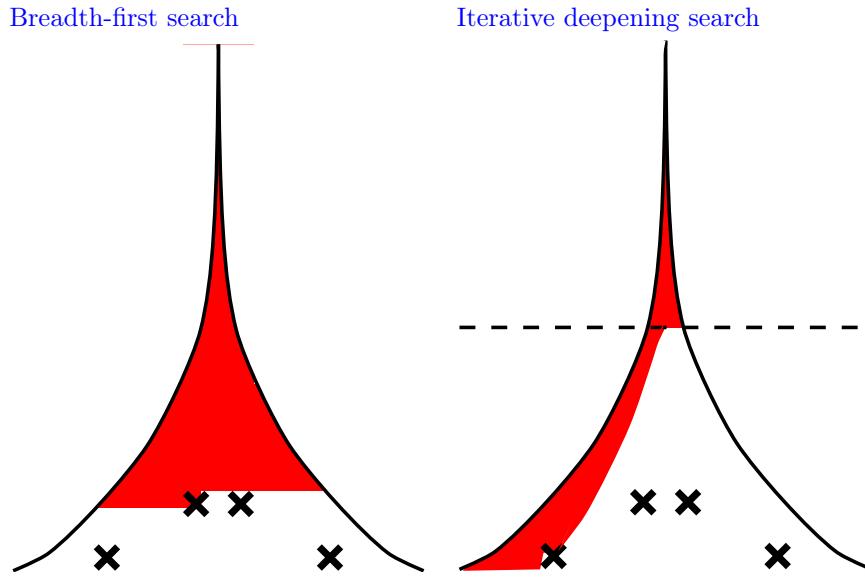
Note: To find a solution (at depth d) we have to search the whole tree up to d . Of course since we do not save the search state, we have to re-compute the upper part of the tree for the next level. This seems like a great waste of resources at first, however, iterative deepening search tries to be complete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

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

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

Comparison BFS (optimal) and IDS (not)

▷ **Example 7.3.5** IDS may fail to be optimal at step sizes > 1 .





Tree Search vs. Graph Search

- ▷ We have only covered tree search algorithms
- ▷ nobody uses these in practice ↳ states duplicated in nodes are a huge problem for efficiency.
- ▷ **Definition 7.3.6** 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 7.3.7** *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, but are more difficult to understand (and to prove properties about).
- ▷ The (complexity) properties are largely stable under duplicate pruning.



7.4 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 (informed search)
 - ▷ Best-first-, A^* -strategies (guide the search by heuristics)
 - ▷ Iterative improvement algorithms



7.4.1 Greedy Search

Best-first search

- ▷ **Idea:** Use an **evaluation function** for each node (estimate of “desirability”)
 - ▷ Expand most desirable unexpanded node

- ▷ **Implementation:** The fringe is a queue sorted in decreasing order of desirability
- ▷ **Special cases:** Greedy search, A^* -search



©: Michael Kohlhase

123



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

If the heuristics is arbitrary, we expect incompleteness!

Depends on how we measure “desirability”.

Concrete examples follow.

Greedy search

- ▷ **Definition 7.4.1** A **heuristic** is an evaluation function h on states that estimates of cost from n to the nearest goal state.
- ▷ **Idea:** Greedy search expands the node that **appears** to be closest to goal
- ▷ **Example 7.4.2** 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 \sim external search control
partial solution cost \sim goal cost estimation



©: Michael Kohlhase

124



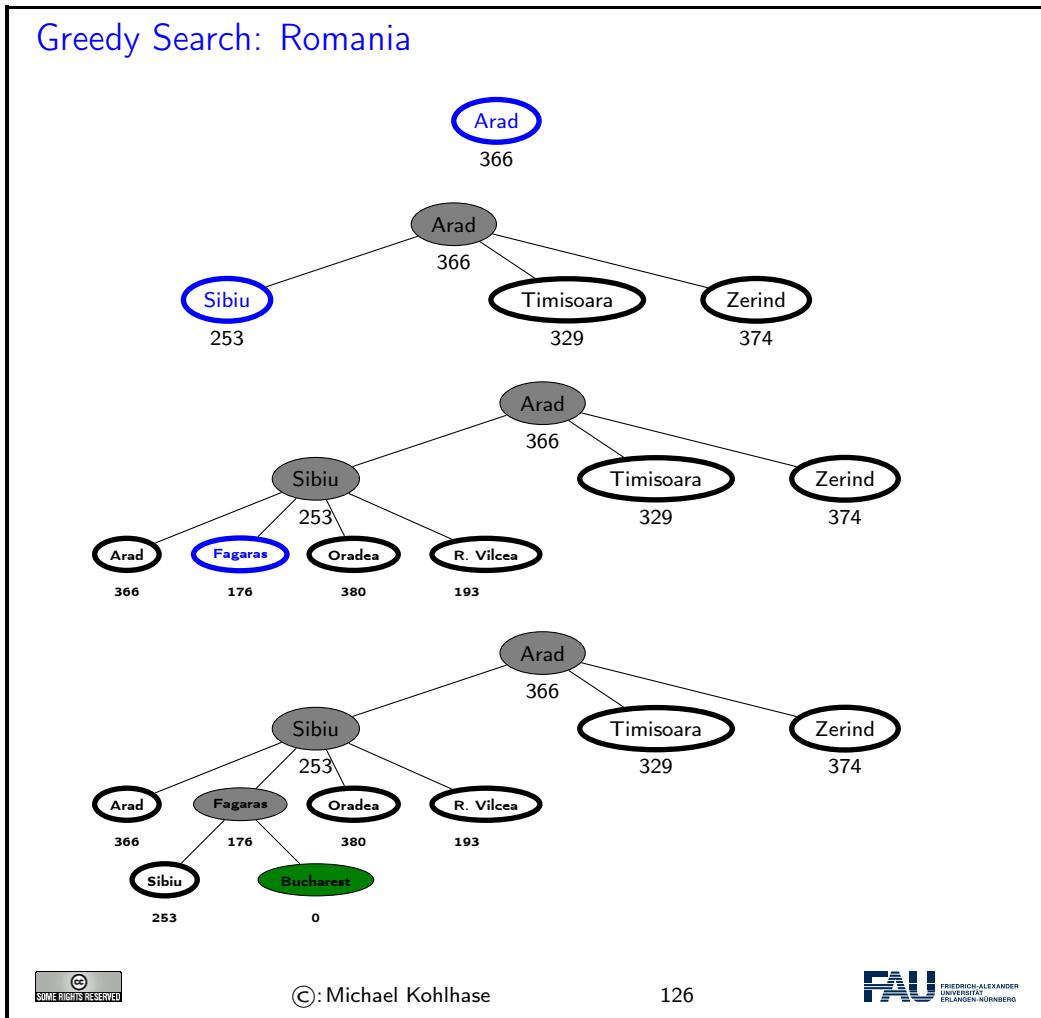
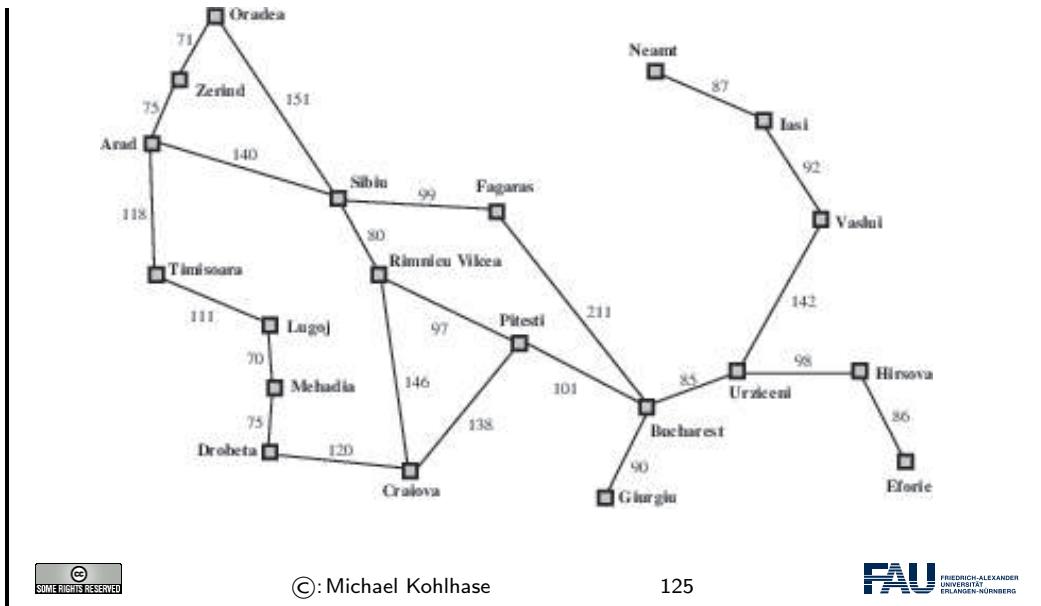
In greedy search we replace the *objective* cost to *construct* the current solution with a heuristic or *subjective* measure from which we think it gives a good idea how far we are from a *solution*. Two things have shifted:

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

Romania with Straight-Line Distances

- ▷ **Example 7.4.3 (Informed Travel In Romania)**
 $h_{SLD}(n)$ = straight-line distance from n to Bucharest

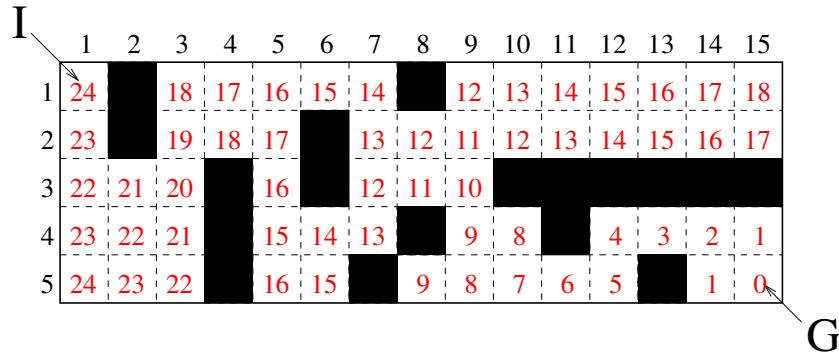
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Heuristic Functions in Path Planning

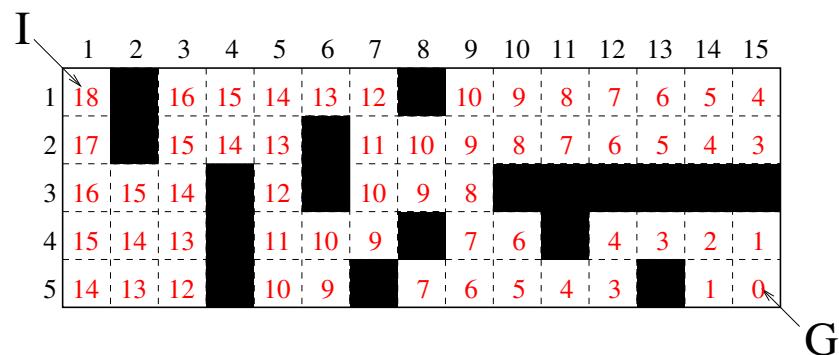
▷ **Example 7.4.4 (The maze solved)**

We indicate h^* by giving the goal distance



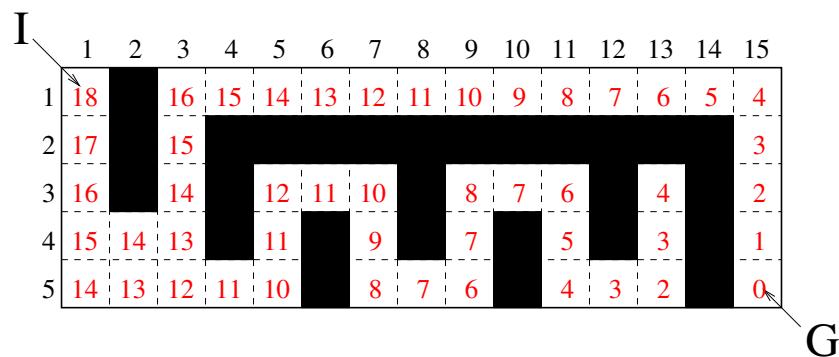
▷ **Example 7.4.5 (Maze Heuristic: the good case)**

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



▷ **Example 7.4.6 (Maze Heuristic: the bad case)**

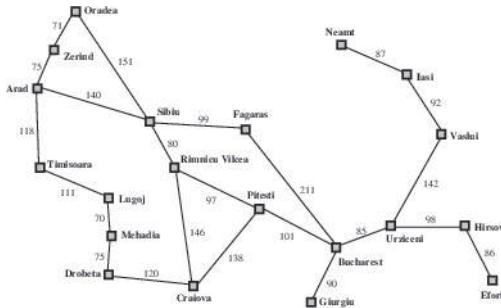
We use the [Manhattan distance](#) to the goal as a heuristic again



Greedy search: Properties

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

- ▷ **Example 7.4.7** Greedy search can get stuck going from Iasi to Oradea:
 $\text{Iasi} \rightarrow \text{Neamt} \rightarrow \text{Iasi} \rightarrow \text{Neamt} \rightarrow \dots$



- ▷ Worst-case time same as depth-first search,
- ▷ Worst-case space same as breadth-first
- ▷ But a good heuristic can give dramatic improvement



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

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

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

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

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

7.4.2 Heuristics and their Properties

Heuristic Functions

- ▷ **Definition 7.4.8** 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 of the **goal distance**
- ▷ **Definition 7.4.9** 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. $\epsilon\nu\rhoεκα!$)
- ▷ Popularized in modern science by George Polya: “How to solve it” [Pól73]
- ▷ same word often used for “rule of thumb” or “imprecise solution method”.



Heuristic Functions: The Eternal Trade-Off

- ▷ “Distance Estimate”? $(h$ is an arbitrary function in principle)
- ▷ In practice, we want it to be **accurate** (aka: **informative**), i.e., close to the actual goal distance.
- ▷ We also want it to be fast, i.e., a small **overhead** for computing h .
- ▷ These two wishes are in contradiction!
- ▷ **Example 7.4.10 (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 7.4.11** *We need to trade off the accuracy of h against the overhead for computing it.*



Properties of Heuristic Functions

- ▷ **Definition 7.4.12** 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.

- ▷ **Proposition 7.4.13 (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.
 - P.1.1 base case:** $h(s) = 0$ by definition of heuristic functions, so $h(s) \leq h^*(s)$ as desired. \square
 - P.1.2 step case:**
 - P.1.2.1** We assume that $h(s') \leq h^*(s')$ for all states s' with a cheapest goal path of length n .
 - P.1.2.2** Let s be a state whose cheapest goal path has length $n + 1$ and the first transition is $o = (s, s')$.
 - P.1.2.3** By consistency, we have $h(s) - h(s') \leq c(o)$ and thus $h(s) \leq h(s') + h^*(a)$.
 - P.1.2.4** By construction, $h^*(s)$ has a cheapest goal path of length n and thus, by induction hypothesis $h(s') \leq h^*(s')$.
 - P.1.2.5** By construction, $h^*(s) = h^*(s') + c(o)$.
 - P.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 7.4.14** 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 7.4.15** *In practice, admissible heuristics are typically consistent.*

- ▷ **Example 7.4.16 (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)



7.4.3 A-Star Search

A* Search: Evaluation Function

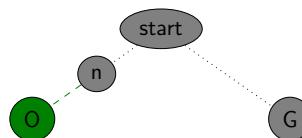
- ▷ **Idea:** Avoid expanding paths that are already expensive (make use of actual cost)
The simplest way to combine heuristic and path cost is to simply add them.
- ▷ **Definition 7.4.17** The evaluation function for A^* -search is given by $f(n) = g(n) + h(n)$, where $g(n)$ is the path cost for n and $h(n)$ is the estimated cost to goal from n .
- ▷ Thus $f(n)$ is the estimated total cost of path through n to goal
- ▷ **Definition 7.4.18** Best-First-Search with evaluation function $g+h$ is called A^* search.



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

A* Search: Optimality

- ▷ **Theorem 7.4.19** A^* search with admissible heuristic is optimal
- ▷ **Proof:** We show that sub-optimal nodes are never selected by A^*
- P.1 Suppose a suboptimal goal G has been generated then we are in the following situation:



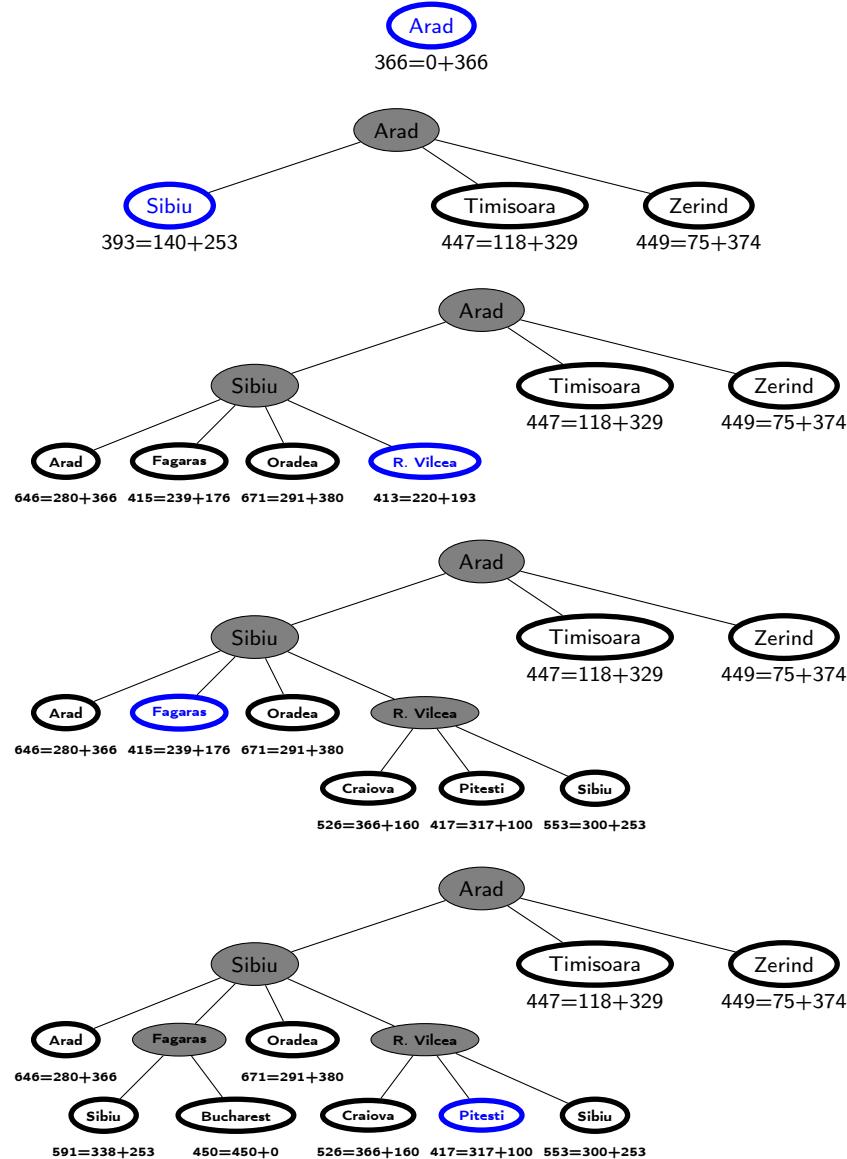
- P.2 Let n be an unexpanded node on a path to an optimal goal O , then

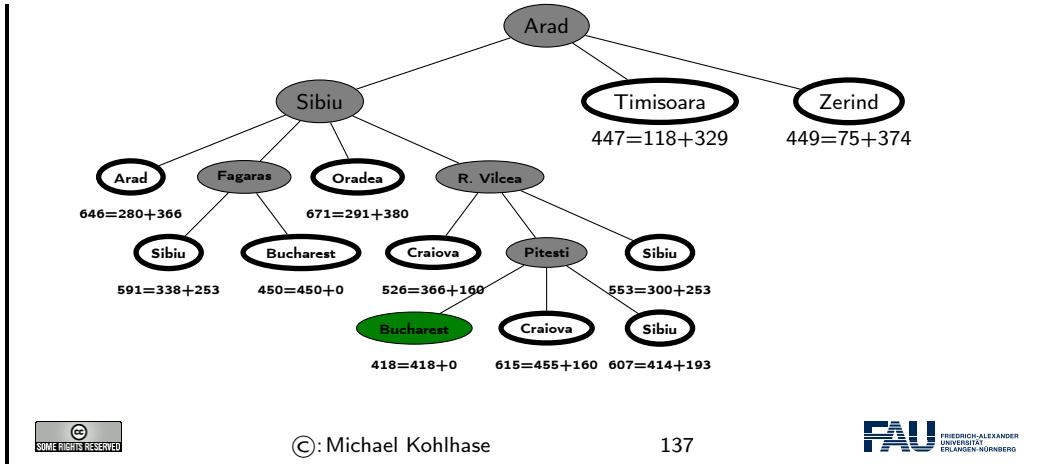
$$\begin{array}{ll}
 f(G) = g(G) & \text{since } h(G) = 0 \\
 g(G) > g(O) & \text{since } G \text{ suboptimal} \\
 g(O) = g(n) + h^*(n) & n \text{ on optimal path} \\
 g(n) + h^*(n) \geq g(n) + h(n) & \text{since } h \text{ is admissible} \\
 g(n) + h(n) = f(n) &
 \end{array}$$

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



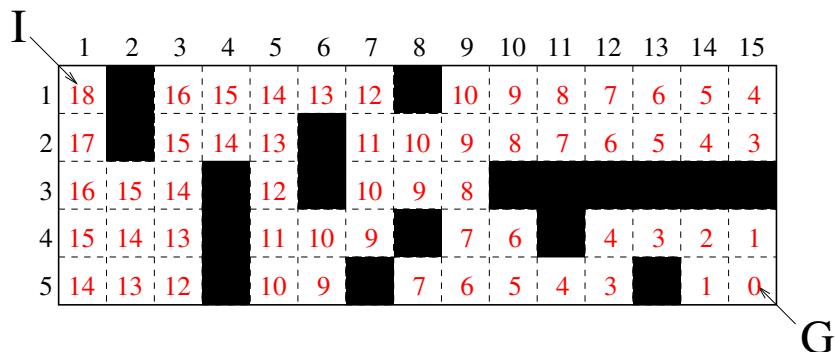
A^* Search Example





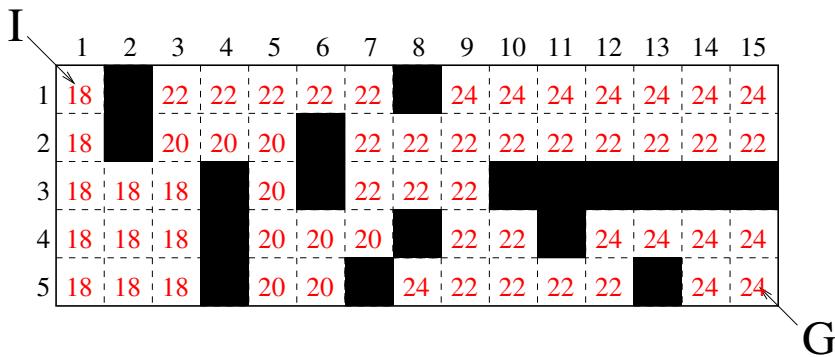
Additional Observations (Not Limited to Path Planning)

▷ Example 7.4.20 (Greedy best-first search, “good case”)



We will find a solution with little search.

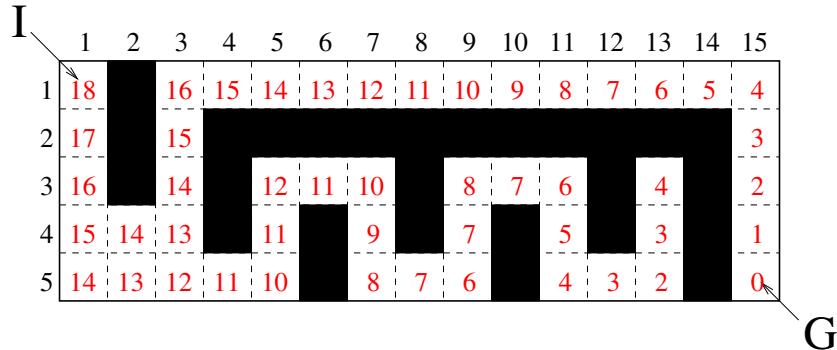
▷ Example 7.4.21 (A^* ($g + h$), “good case”)



▷ A^* with a consistent heuristic $g + h$ always increases monotonically (h cannot decrease more than g increases)

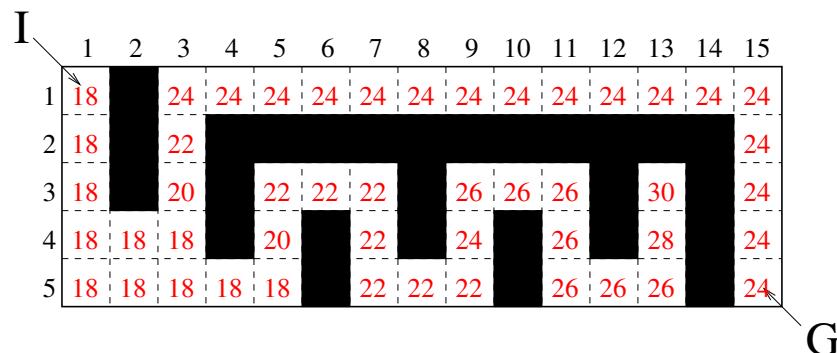
▷ We need more search, in the “right upper half”. This is typical: Greedy best-first search tends to be faster than A^* .

▷ Example 7.4.22 (Greedy best-first search, “bad case”)



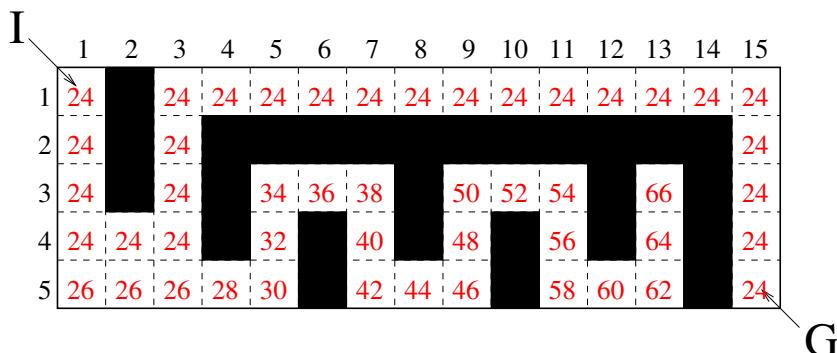
Search will be mis-guided into the “dead-end street”.

▷ Example 7.4.23 (A^* ($g + h$), “bad case”)



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

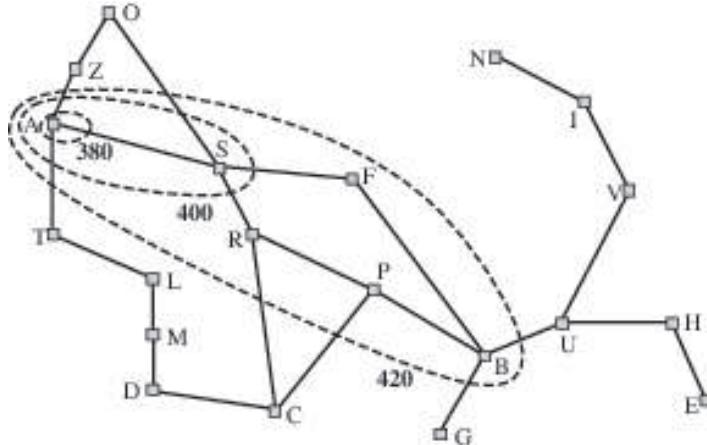
▷ Example 7.4.24 (A^* ($g + h$) using h^*)



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



©: Michael Kohlhase

139

*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 .



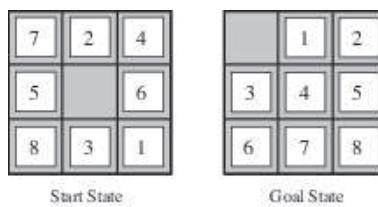
©: Michael Kohlhase

140

7.4.4 Finding Good Heuristics

Since the availability of admissible heuristics is so important for informed search (particularly for A^*), let us see how such heuristics can be obtained in practice. We will look at an example, and then derive a general procedure from that.

Admissible heuristics: Example 8-puzzle



- ▷ **Example 7.4.25** Let $h_1(n)$ be the number of misplaced tiles in node n
 $(h_1(S) = 6)$
- ▷ **Example 7.4.26** 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 7.4.27 (Typical search costs)** ($\text{IDS} \cong \text{iterative deepening search}$)

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



Dominance

- ▷ **Definition 7.4.28** 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 7.4.29** If h_2 dominates h_1 , then h_2 is better for search than h_1 .



Relaxed problems

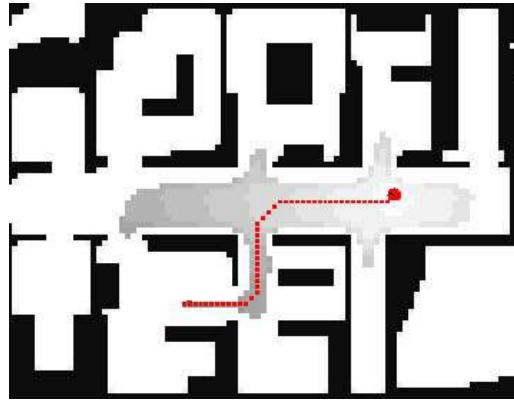
- ▷ **Finding good admissible heuristics is an art!**
- ▷ **Idea:** Admissible heuristics can be derived from the [exact](#) solution cost of a *relaxed* version of the problem.
- ▷ **Example 7.4.30** If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then we get heuristic h_1 .
- ▷ **Example 7.4.31** If the rules are relaxed so that a tile can move to *any adjacent square*, then we get heuristic h_2 . ([Manhattan distance](#))
- ▷ **Definition 7.4.32** Let $\mathcal{P} := \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ be a search problem, then we call $\mathcal{P}^r := \langle \mathcal{S}, \mathcal{O}^r, \mathcal{I}^r, \mathcal{G}^r \rangle$ a **relaxed problem** (wrt. \mathcal{P} ; or simply **relaxation** of \mathcal{P}), iff $\mathcal{O} \subseteq \mathcal{O}^r$, $\mathcal{I} \subseteq \mathcal{I}^r$, and $\mathcal{G} \subseteq \mathcal{G}^r$.
- ▷ **Lemma 7.4.33** If \mathcal{P}^r relaxes \mathcal{P} , then every solution for \mathcal{P} is one for \mathcal{P}^r .
- ▷ **Key point:** The optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem



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

Empirical Performance: A^* in Path Planning

- ▷ Example 7.4.34 (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!



7.5 Local Search

Systematic Search vs. Local Search

- ▷ Definition 7.5.1 We call a search algorithm **systematic**, if it considers all states at some point.
- ▷ Example 7.5.2 All tree search algorithms are systematic (given reasonable assumptions e.g. about costs.)
- ▷ Observation 7.5.3 *Systematic search procedures are complete.*
- ▷ Observation 7.5.4 *There is no limit of the number of search nodes that are kept in memory at any time systematic search procedures.*
- ▷ Alternative: Keep only one (or a few) search nodes at a time
 - ▷ no systematic exploration of all options, thus incomplete



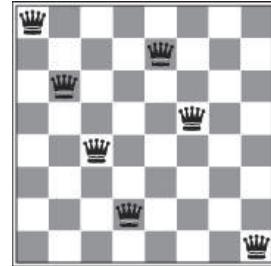
Local Search Problems

- ▷ Idea: Sometimes the path to the solution is irrelevant

▷ **Example 7.5.5 (8 Queens Problem)** Place 8 queens on a chess board, so that no two queens threaten each other.

▷ This problem has various solutions (the one of the right isn't one of them)

▷ **Definition 7.5.6** 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 successors of the current state, and do not retain search paths.

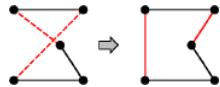
▷ Applications include: integrated circuit design, factory-floor layout, job-shop scheduling, portfolio management, fleet deployment,...



Local Search: Iterative improvement algorithms

▷ **Definition 7.5.7 (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 7.5.8 (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.

▷ Depth-first search with heuristic and w/o memory

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

```

- ▷ Like starting anywhere in search tree and making a heuristically guided DFS.
- ▷ Works, if solutions are dense and local maxima can be escaped.



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

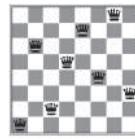
Of course, this will only get us into local maxima, and has no guarantee of getting us into global ones (remember, we are blind). The solution to this problem is to re-start the search at random (we do not have any information) places, and hope that one of the random jumps will get us to a slope that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

Example Hill-Climbing with 8 Queens

▷ **Idea:** Heuristic function h is number of queens that threaten each other.

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

▷ **Example 7.5.9** An 8-queens state with heuristic cost estimate $h = 17$ showing h -values for moving a queen within its column



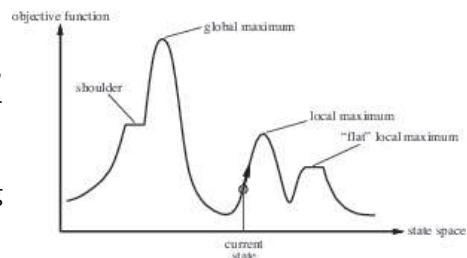
▷ **Problem:** The state space has local minima. e.g. the board on the right has $h = 1$ but every successor has $h > 1$.



Hill-climbing

▷ **Problem:** Depending on initial state, can get stuck on local maxima/minima and plateaus

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



- ▷ **Idea:** Escape local maxima by allowing some “bad” or random moves.
- ▷ **Example 7.5.10** local search, simulated annealing...
- ▷ **Properties:** All are incomplete, non-optimal.
- ▷ Sometimes performs well in practice (if (optimal) solutions are dense)



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

Simulated annealing (Idea)



- ▷ **Definition 7.5.11** **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 cristal structure.
- ▷ Simulated Annealing is like shaking a ping-pong ball occasionally on a bumpy surface to free it. (so it does not get stuck)
- ▷ Devised by Metropolis et al., 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 \Rightarrow 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?



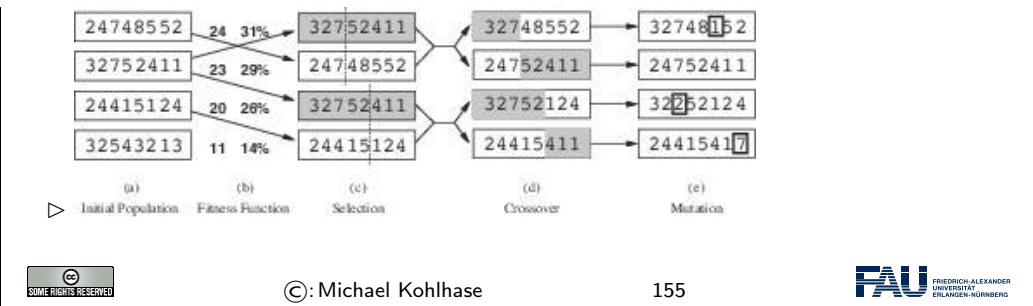
Local beam search

- ▷ **Idea:** Keep k states instead of 1; choose top k of all their successors
- ▷ 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 successors randomly, biased towards good ones. (Observe the close analogy to natural selection!)



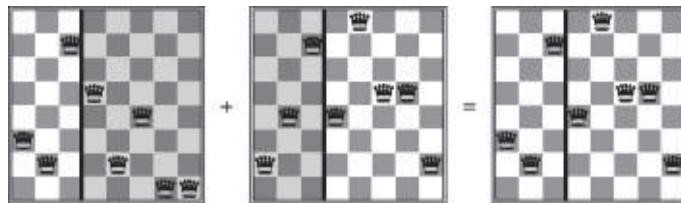
Genetic algorithms (very briefly)

- ▷ **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)



Genetic algorithms (continued)

- ▷ Problem: Genetic Algorithms require states encoded as strings (GPs use programs)
- ▷ Crossover helps iff substrings are meaningful components
- ▷ Example 7.5.12 (Evolving 8 Queens)



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

Chapter 8

Adversarial Search for Game Playing

8.1 Introduction

The Problem

(cf. Chapter 7)



“Adversarial search” = Game playing against an opponent.



©: Michael Kohlhase

157



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)



©: Michael Kohlhase

158



"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).



©: Michael Kohlhase

159



An Example Game



SOME RIGHTS RESERVED

© Michael Kohlhase

160

FAU FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

“Game” Playing? Which Games *Not*?

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

SOME RIGHTS RESERVED

© Michael Kohlhase

161

FAU FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

(A Brief Note On) Formalization

- ▷ **Definition 8.1.1 (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 $s \xrightarrow{s} s'$ then $s \in S^{\text{Max}}$ and $s' \in (S^{\text{Min}} \cup S^T)$.
 - ▷ For $a \in A^{\text{Min}}$, if $s \xrightarrow{s} s'$ 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 8.1.2 (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 8.1.3** 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 8.1.4** 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.
- ▷ Declarative! With “game description language” = 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. Chapter 7).
 - ▷ 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).



8.2 Minimax Search

“Minimax”?

▷ We want to compute an optimal strategy for player “Max”.

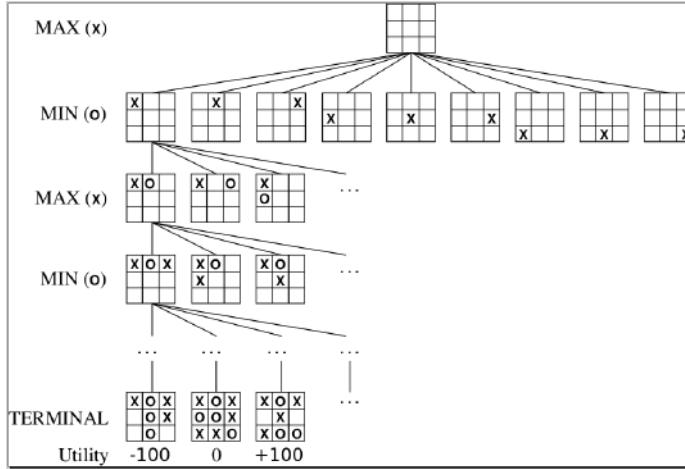
▷ In other words: “**We are Max, and our opponent is Min.**”

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 \rightsquigarrow hence “minimax”.



Example Tic-Tac-Toe



▷ Game tree, current player marked on the left.

▷ Last row: terminal positions with their utility.



Minimax: Outline

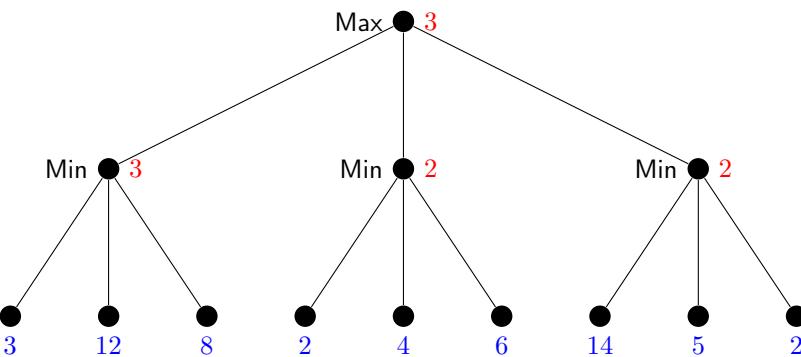
▷ **We max, we min, we max, we min ...**

1. Depth-first search in game tree, with Max in the root.
2. Apply 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 8.2.1** 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

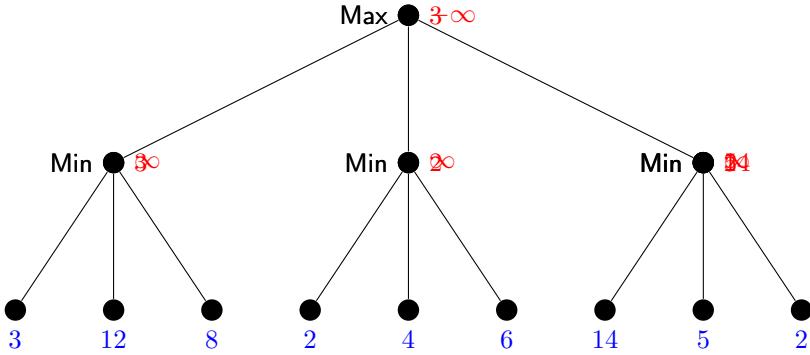
function Max–Value( $s$ ) returns a utility value
  if Terminal–Test( $s$ ) then return  $u(s)$ 
   $v := -\infty$ 
  for each  $a \in \text{Actions}(s)$  do
     $v := \max(v, \text{Min–Value}(\text{ChildState}(s,a)))$ 
  return  $v$ 

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



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



Minimax, Pro and Contra

- ▷ Minimax advantages:
 - ▷ Minimax is the simplest possible (reasonable) 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.



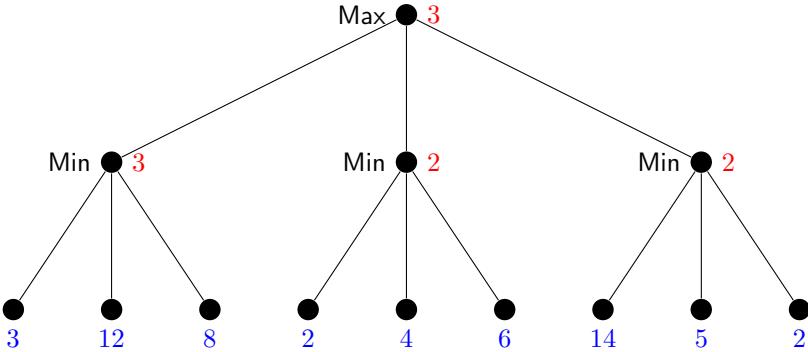
8.3 Evaluation Functions

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 8.3.1** 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. Section 7.4): We want f to be both **(a) accurate and (b) fast**.
- ▷ Another analogy: **(a) and (b) are in contradiction** \rightsquigarrow need to trade-off accuracy against overhead.
 - ▷ In typical game playing algorithms today, f is inaccurate but very fast.
(Usually no good methods known for computing accurate f)



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 .



Example Chess



▷ 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)



© Michael Kohlhase

177



Linear Evaluation Functions

▷ 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 $s \xrightarrow{s} s'$, adapt $f(s)$ to $f(s')$ by considering only those features whose values have changed.



© Michael Kohlhase

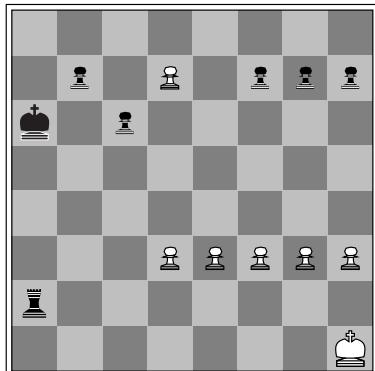
178



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

The Horizon Problem

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



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.



©: Michael Kohlhase

179



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 . . . → Keep searching until the end of the piece exchange is reached.



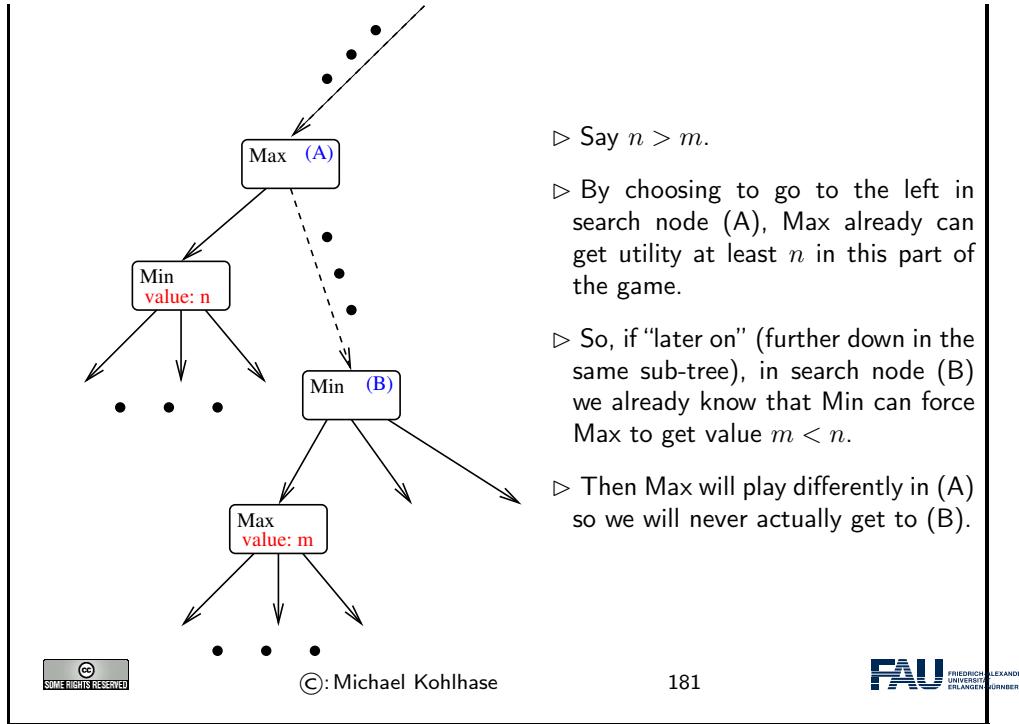
©: Michael Kohlhase

180



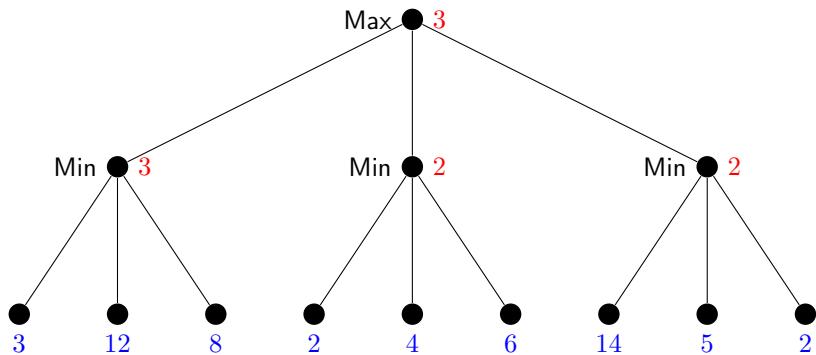
8.4 Alpha-Beta Search

[When We Already Know We Can Do Better Than This](#)



Alpha Pruning: Basic Idea

▷ Question: Can we save some work here?

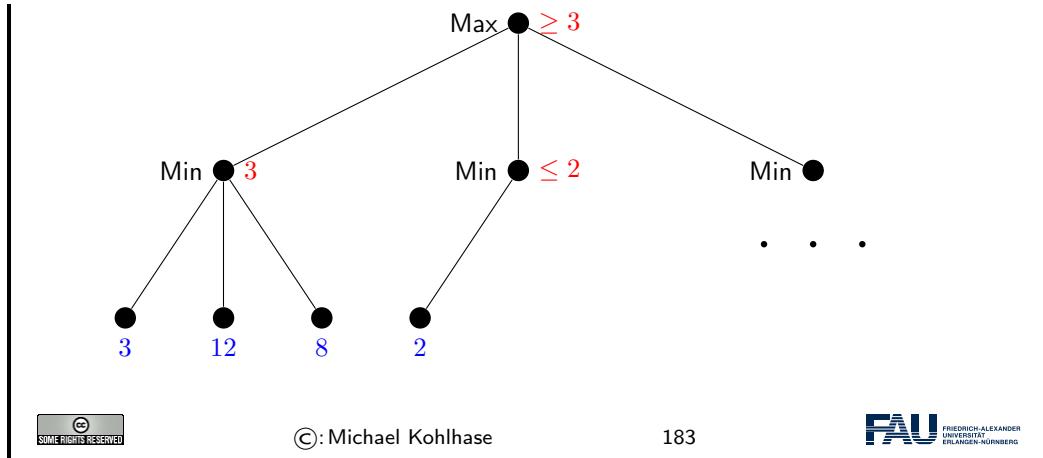


182

FAU FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

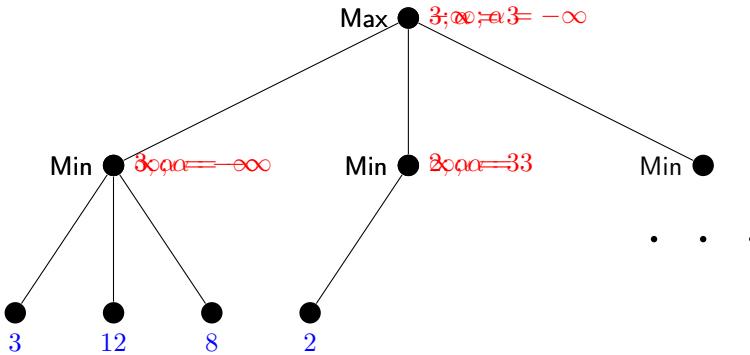
Alpha Pruning: Basic Idea (Continued)

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

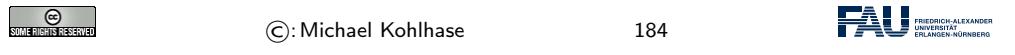


Alpha Pruning

- ▷ 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 .



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.)



▷ Alpha-Beta Pruning

- ▷ Recall:
 - ▷ **What is α :** For each search node n , the highest Max-node utility that search has encountered on its path from the root to n .
 - ▷ **How to use α :** In a Min node n , if one of the successors already has utility $\leq \alpha$, 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 8.4.1** 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  $v$ 

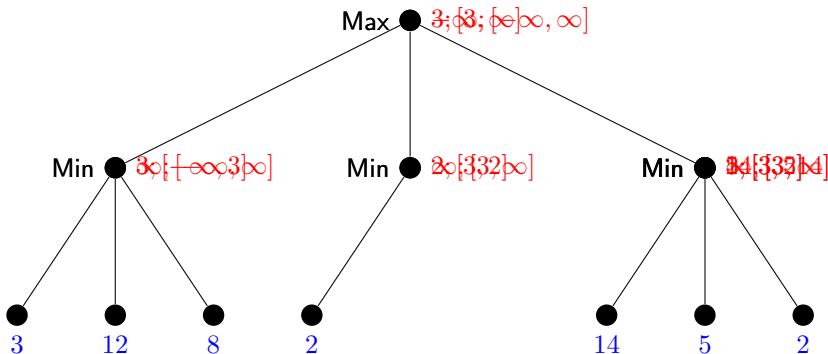
```

$\hat{=}$ Minimax (slide 172) + α/β book-keeping and pruning.



Alpha-Beta Search: Example

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

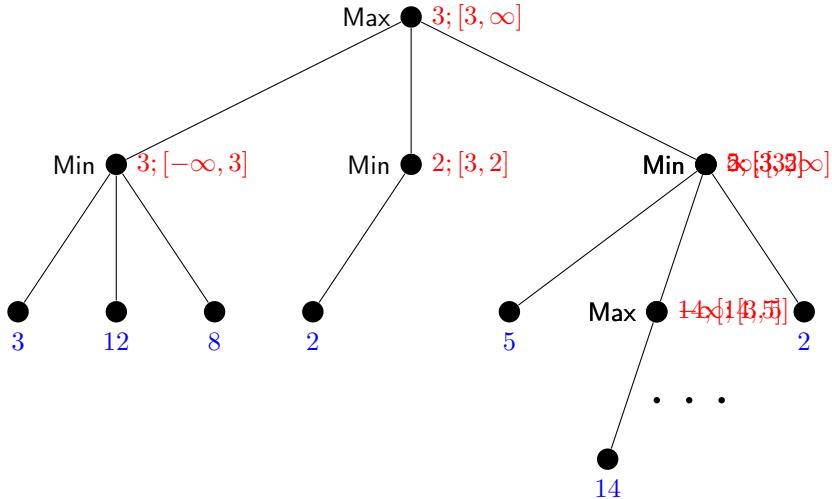


Note: We could have saved work by choosing the opposite order for the successors of the rightmost Min node. Choosing the best moves (for each of Max and Min) first yields more pruning!



▷ Alpha-Beta Search: 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.



8.5 Monte-Carlo Tree Search (MCTS)

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

And now ...



▷

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



Monte-Carlo Tree Search: Basic Ideas

▷ **Definition 8.5.1** 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 8.5.2** For **Monte-Carlo tree search** we maintain a search tree T .

```
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.

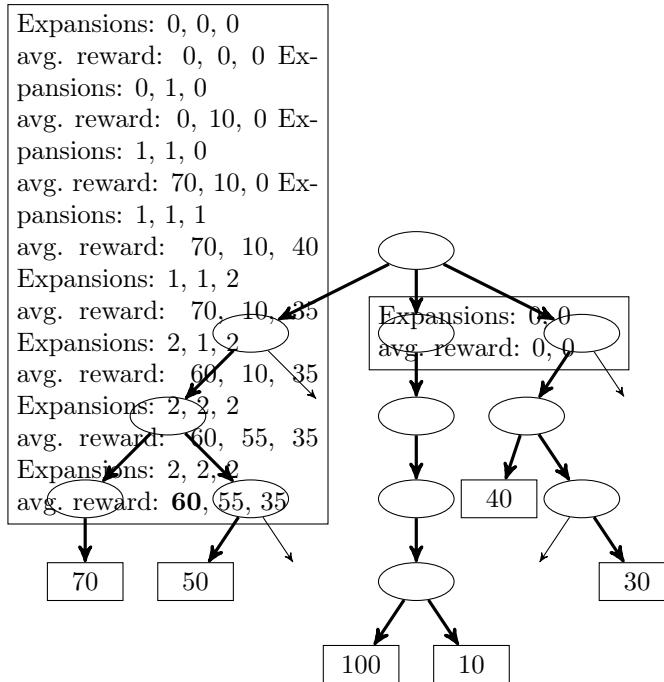
- ▷ ▷ **Pro:** runtime & memory.
- ▷ **Contra:** need good guidance how to “select” and “sample”.



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

Monte-Carlo Sampling: Illustration of Sampling

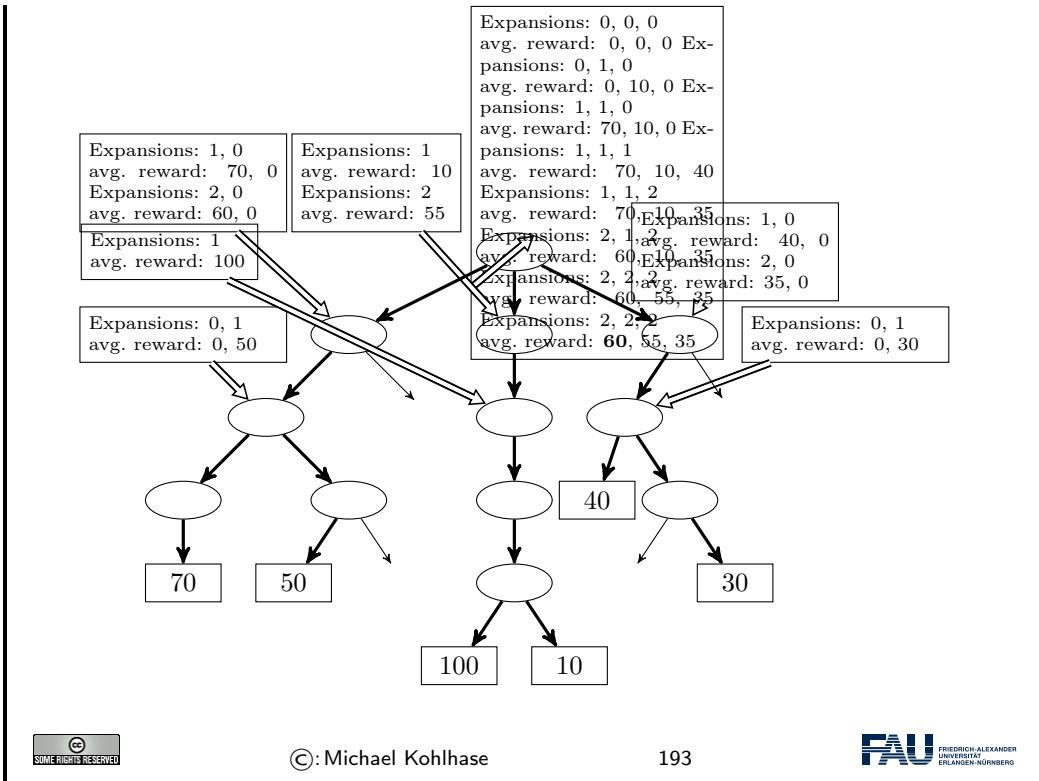
- ▷ **Idea:** Sample the search tree keeping track of the average utilities.
- ▷ **Example 8.5.3 (Single-player, for simplicity)** (with adversary, distinguish max/min nodes)



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

Monte-Carlo Tree Search: Building the Tree

- ▷ **Idea:** we can save work by building the tree as we go along
- ▷ **Example 8.5.4 (Redoing the previous example)**



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

How to Guide the Search?

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



AlphaGo: Overview

▷ **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:**

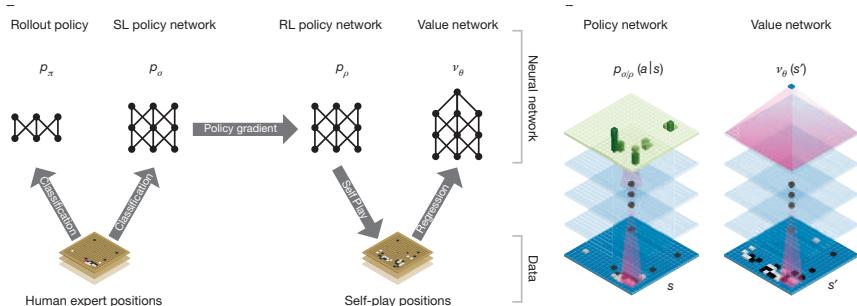
- ▷ Policy networks bias the action choices within the MCTS tree (and hence the leaf-state selection), and bias the random samples.
- ▷ Value networks are an additional source of state values in the MCTS tree, along with the random samples.

▷ And now in a little more detail



Neural Networks in AlphaGo

▷ **Illustration:** taken from [Sil+16]

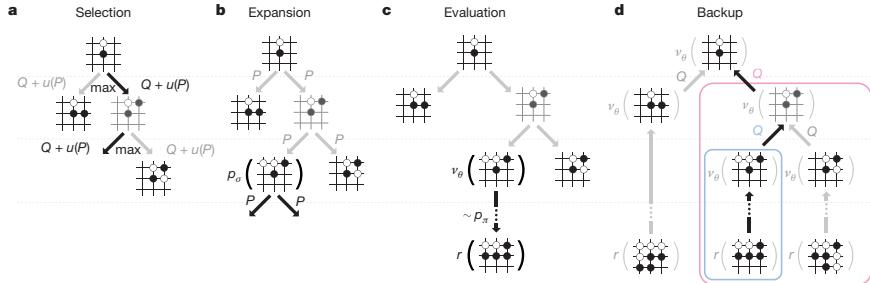


- ▷ **Rollout policy p_π :** Simple but fast, \approx prior work on Go.
- ▷ **SL policy network p_σ :** Supervised learning, human-expert data ("learn to choose an expert action").
- ▷ **RL policy network p_ρ :** Reinforcement learning, self-play ("learn to win").
- ▷ **Value network v_θ :** Use self-play 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

▷ 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 .



AlphaGo, Conclusion?:

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

8.6 State of the Art

State of the Art

▷ Some well-known board games:

- ▷ **Chess:** Up next.
- ▷ **Othello (Reversi):** In 1997, “Logistello” beat the human world champion. Best computer players now are clearly better than best human players.
- ▷ **Checkers (Dame):** Since 1994, “Chinook” is the 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:** The best computer players nowadays (“Zen”, “Mogo”, “Crazystone”) are at the level of good amateurs. Go remains challenging due to the immense branching factor (cf. slide 10). The best known methods use UCT variants.



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



Computer Chess: Famous Quotes

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

▷ Chess is the drosophila of Artificial Intelligence. ([Alexander Kronrod \(1965\)](#))



Computer Chess: Another Famous Quote

▷ In 1965, the Russian mathematician Alexander Kronrod said, “Chess is the Drosophila of artificial intelligence.”

However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophilae. We would have some science, but mainly we would have very fast fruit flies. ([John McCarthy \(1997\)](#))



Summary

- ▷ Games (2-player turn-taking zero-sum discrete and finite games) can be understood as a simple extension of classical search problems.

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



Reading

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



Chapter 9

Constraint Satisfaction Problems

9.1 Constraint Satisfaction Problems: Motivation

A (Constraint Satisfaction) Problem

- ▷ **Example 9.1.1 (Tournament Schedule)** Who's going to play against who, when and where?



SOME RIGHTS RESERVED

©: Michael Kohlhase

204

FAU FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Constraint satisfaction problems (CSPs)

- ▷ Standard search problem: `state` is a “black box”—any old data structure that supports goal test, eval, successor
- ▷ **Definition 9.1.2** 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** $\{D_v \mid v \in V\}$ and the goal test is a set of **constraints** specifying allowable combinations of values for subsets of variables.

- ▷ Simple example of a *formal representation language*
- ▷ Allows useful *general-purpose* algorithms with more power than standard search algorithms



©: Michael Kohlhase

205



Another Constraint Satisfaction Problem

- ▷ Example 9.1.3 (SuDoKu)

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:** Content of each cell.
- ▷ **Domains:** Numbers $1, \dots, 9$.
- ▷ **Constraints:** Each number only once in each row, column, block.



©: Michael Kohlhase

206



Example: Map-Coloring



- ▷ **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., $WA \neq NT$ (if the language allows this), or $\langle WA, NT \rangle \in \{\langle \text{red}, \text{green} \rangle, \langle \text{red}, \text{blue} \rangle, \langle \text{green}, \text{red} \rangle, \dots\}$



- ▷ **Solutions** are assignments satisfying all constraints,
- ▷ e.g., $\{WA = \text{red}, NT = \text{green}, \dots\}$



©: Michael Kohlhase

207



Bundesliga Constraints

▷ **Variables:** $v_{A \text{vs. } B}$ where A and B are teams, with domain $\{1, \dots, 34\}$: 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.

▷ ...



©: Michael Kohlhase

208



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 million 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]

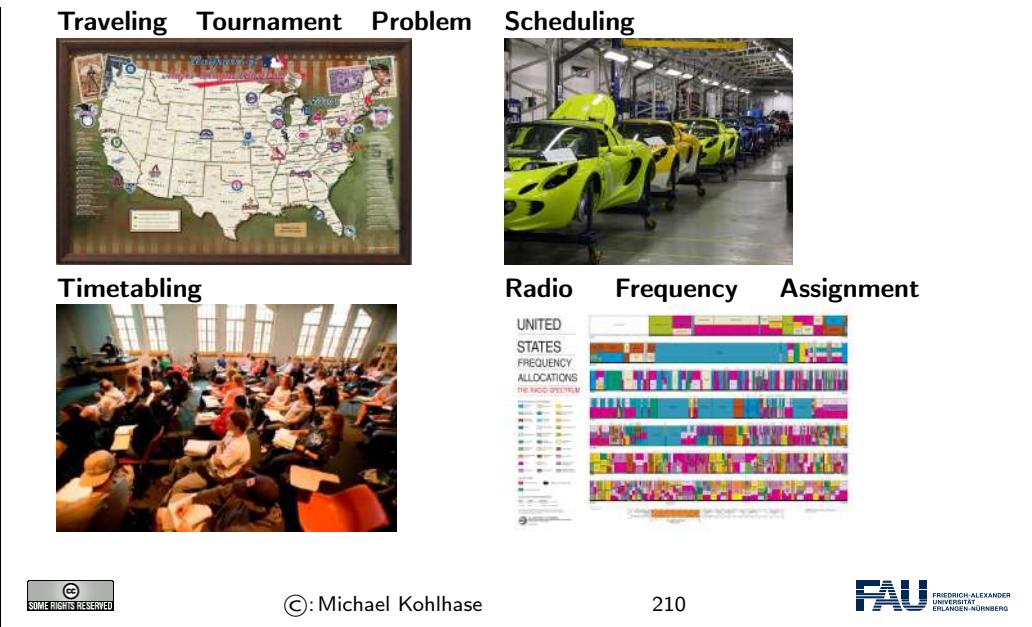


©: Michael Kohlhase

209



Some Applications



SOME RIGHTS RESERVED

©: Michael Kohlhase

210

FAU FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

1. U.S. Major League Baseball, 30 teams, each 162 games. There's one crucial additional difficulty, in comparison to Bundesliga. Which one? Travel is a major issue here!! Hence "Traveling Tournament Problem" in reference to the TSP.
2. This particular scheduling problem is called "car sequencing", how to most efficiently get cars through the available machines when making the final customer configuration (non-standard/flexible/custom extras).
3. Another common form of scheduling ...
4. The problem of assigning radio frequencies so that all can operate together without noticeable interference. Var domains are available frequencies, constraints take form of $|x - y| > \delta_{xy}$, where delta depends on the position of x and y as well as the physical environment.

Our Agenda for This Topic

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

SOME RIGHTS RESERVED

©: Michael Kohlhase

211

FAU FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

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 give direction to a backtracking search?
 - ▷ Simple methods for making backtracking aware of the structure of the problem, and thereby reduce search.



©: Michael Kohlhase

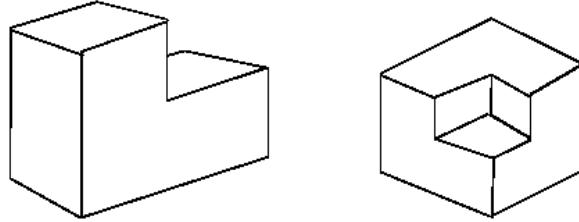
212



9.2 The Waltz Algorithm

The Waltz Algorithm

- ▷ One of the earliest examples of applied CSPs
- ▷ **Motivation**: interpret line drawings of solid 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.



©: Michael Kohlhase

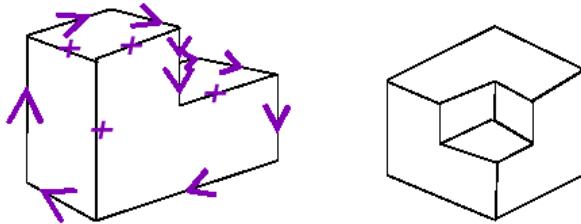
213



Waltz Algorithm on Simple Scenes

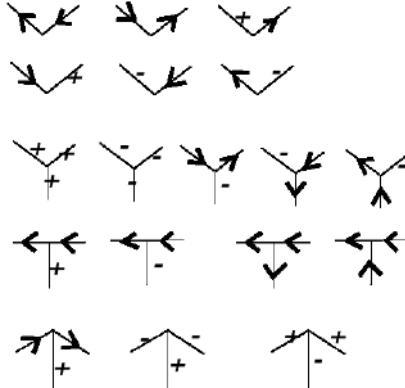
- ▷ **Assumptions**: All objects
 - ▷ have no shadows or cracks

- ▷ have only three-faced vertices
- ▷ are in “general position”, i.e. no junctions change with small movements of the eye
- ▷ **Observation 9.2.1** Then each line on the images is one of the following:
 - ▷ a boundary line (edge of an object) ($<$) with right hand of arrow denoting “solid” and left hand denoting “space”
 - ▷ an interior convex edge (label with “+”)
 - ▷ an interior concave edge (label with “-”)



18 Legal Kinds of Junctions

- ▷ **Observation 9.2.2** There are only 18 “legal” kinds of junctions:



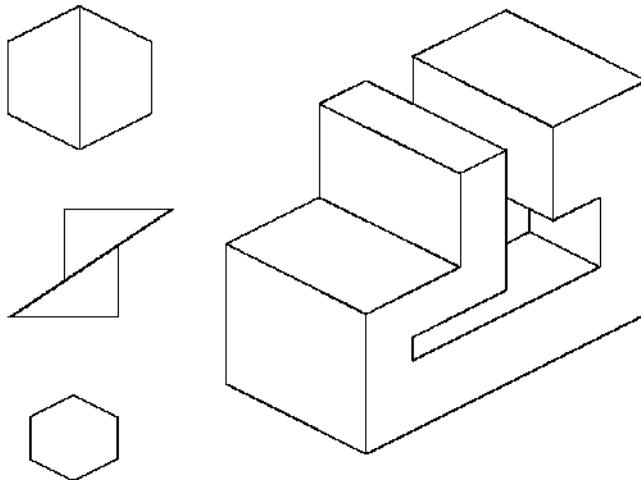
Idea: given a representation of a diagram

- ▷▷ 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

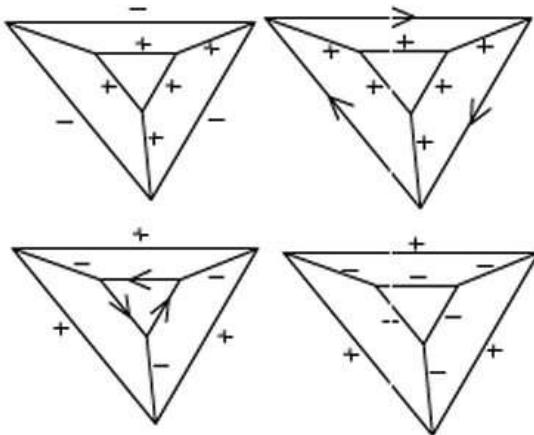


©: Michael Kohlhase

216



Waltz Algorithm (More Examples): Ambiguous Figures

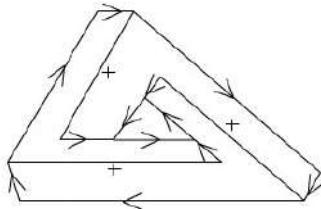


©: Michael Kohlhase

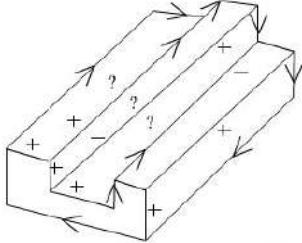
217



Waltz Algorithm (More Examples): Impossible Figures



Consistent labelling for impossible figure



No consistent labelling possible



9.3 CSP: Towards a Formal Definition

Varieties of CSPs

- ▷ n discrete variables
 - ▷ finite domains; size $d \sim \mathcal{O}(d^n)$ complete assignments
 - ▷ e.g., Boolean CSPs, incl. Boolean satisfiability (**NP**-complete)
 - ▷ infinite domains (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 solvable, nonlinear undecidable
- ▷ Continuous variables
 - ▷ e.g., 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



Types of Constraints

- ▷ **Unary constraints** involve a single variable, e.g., $SA \neq green$
- ▷ **Binary constraints** involve pairs of variables, e.g., $SA \neq WA$

- ▷ Higher-order constraints involve 3 or more variables, e.g., cryptarithmetic column constraints
- ▷ Preferences (soft constraints) (e.g., red is better than green) often representable by a cost for each variable assignment
→ constrained optimization problems.



Non-Binary Constraints

- ▷ Example 9.3.1 (Send More Money)

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

Puzzle: letters stand for digits, addition should work out

- ▷ Variables: S, E, N, D, M, O, R, Y with domain $0, \dots, 9$
- ▷ Constraints:
 - ▷ all variables should have different values: $S \neq E, S \neq N, \dots$
 - ▷ first digits are non-zero: $S \neq 0, M \neq 0$.
 - ▷ addition $1000S + 100E + 10N + D + 1000M + 100O + 10R + E = 10000M + 100000 + 100N + 10E + Y$.

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

- ▷ Problem: The last constraint is of order 8. (8 variables involved)
- ▷ solution: For $n \geq 3$, encode $C(v_1, \dots, v_{n-1}, v_n)$ as

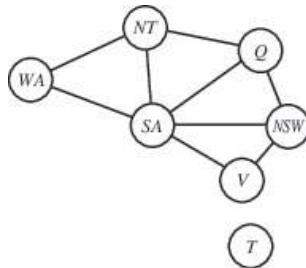
$$C(p_1(x), \dots, p_{n-1}(x), v_n) \wedge v_1 = p_1(x) \wedge \dots \wedge v_{n-1} = p_{n-1}(x)$$

- ▷ Problem: The problem structure gets hidden (search algorithms get confused)



Constraint Graph

- ▷ Definition 9.3.2 A binary CSP is a CSP where each constraint relates at most two variables.
- ▷ Observation 9.3.3 A constraint network forms a graph whose nodes are variables, and whose edges represent the constraints.



▷ Example 9.3.4

▷ General-purpose CSP algorithms use the graph structure to speed up search.
(E.g., Tasmania is an independent subproblem!)



Real-world CSPs

- ▷ Example 9.3.5 (Assignment problems) (e.g., who teaches what class)
- ▷ Example 9.3.6 (Timetabling problems) (e.g., which class is offered when and where?)
- ▷ Example 9.3.7 (Hardware configuration)
- ▷ Example 9.3.8 (Spreadsheets)
- ▷ Example 9.3.9 (Transportation scheduling)
- ▷ Example 9.3.10 (Factory scheduling)
- ▷ Example 9.3.11 (Floorplanning)
- ▷ Notice that many real-world problems involve real-valued variables



Constraint Satisfaction Problems (Definition)

- ▷ Definition 9.3.12 A **constraint network** is a triple $\langle V, D, C \rangle$, where
 - ▷ $V := \{X_1, \dots, X_n\}$ is a finite set of **variables**,
 - ▷ $D := \{D_v \mid v \in V\}$ the set of their **domains**, and
 - ▷ $C := \{C_{uv} \mid u, v \in V \text{ and } u \neq v\}$, where a (binary) **constraint** C_{uv} is a relation ($C_{uv} \subseteq D_u \times D_v$) and $C_{uv} = C_{vu}$.
- ▷ Observation 9.3.13 *Binary CSPs can be formulated as constraint networks.*

Remarks:

- ▷ $C_{uv} \subseteq D_u \times D_v \hat{=} \text{possible assignments to variables } u \text{ and } v$
- ▷ relations are most general formalization, generally we use e.g. " $u = v$ " ($C_{uv} = \{(a, b) \mid a = b\}$) or " $u \neq v$ ".

- ▷ A unary constraint C_u can be expressed by restricting the domain of v :
 $D_v := C_v$.
- ▷ **Lemma 9.3.14** Higher-order constraints can be transformed into equisatisfiable binary constraints using auxiliary variables.
- ▷ **Corollary 9.3.15** Any CSP can be represented by a constraint network.



Example: SuDoKu

- ▷ **Example 9.3.16 (Formalize SuDoKu as a CSP)**

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

- ▷ **Variables:** $V = \{v_{ij} \mid 1 \leq i, j \leq 9\}$: 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 Constraints:** $C_{v_{ij} v_{i'j'}} \stackrel{\Delta}{=} "v_{ij} \neq v_{i'j'}"$, i.e.
 $C_{v_{ij} v_{i'j'}} = \{(d, d') \in D \times D \mid d \neq d'\}$, for: $i = i'$ (same row), or $j = j'$ (same column), or $(\lceil \frac{i}{3} \rceil, \lceil \frac{j}{3} \rceil) = (\lceil \frac{i'}{3} \rceil, \lceil \frac{j'}{3} \rceil)$ (same block).



Constraint Satisfaction Problems (Solutions)

- ▷ **Definition 9.3.17** Let $\langle V, D, C \rangle$ be a **constraint satisfaction problem**, 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 9.3.18** 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.
- ▷ **Example 9.3.19** The **empty assignment** \emptyset is (trivially) consistent
- ▷ **Definition 9.3.20** 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 9.3.21** Let $\gamma := \langle V, D, C \rangle$ be a constraint satisfaction problem, then we call a consistent (total) assignment a solution γ and γ itself solvable.
- ▷ We can view a constraint network as a search problem, if we take the states as the partial assignments, the operators as assignment extension, and the goal states as consistent assignments.



Questionnaire

- ▷ **Question:** Which of the following statements imply that the empty assignment, a_0 , can always be extended to a solution?
 - A a_0 is consistent.
 - B The network is inconsistent.
 - C There are no binary constraints.
 - D The network is solvable.
- ▷ (A) No. Being consistent does not imply being extensible to a solution (cf. previous slide). For a_0 in particular: a_0 is always consistent; it can be extended to a solution if and only if the network is solvable.
- ▷ (B) No. If the network is inconsistent then there are no solutions, so no assignment can be extended to a solution, in particular not a_0 .
- ▷ (C): If one of the unary constraints (variable domains) is empty, then the network is inconsistent and we are in case (B). Otherwise, the network is solvable and a_0 is extensible to a solution, c.f. answer to (A).
- ▷ (D): Yes. The empty assignment can be extended to any solution for the network, if such a solution does exist.



Computational Complexity of CSP

- ▷ **Input size vs. solution space size:** Assume constraint network γ with n variables, all with domain size k .
 - ▷ Number of total assignments: k^n .
 - ▷ Size of description of γ : $n k$ for variables and domains; at most n^2 constraints, each of size at most $k^2 \sim \mathcal{O}(n^2 k^2)$.
 - ▷ The number of assignments is exponentially bigger than the size of γ .
 - ▷ It is therefore no surprise that:
 - ▷ **Theorem 9.3.22 (CSP is NP-complete)** *It is NP-complete to decide whether or not a given constraint network γ is solvable.*
 - ▷ **Proof:**

P.1 ([Membership in NP](#)) Just guess a total assignment a and verify (in polynomial time) whether a is a solution.

P.2 ([NP-Hardness](#)) The special case of planar graph coloring with 3 colors (our illustrative example) is known to be **NP-hard** \square



Questionnaire

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

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

Can this partial assignment be extended to a solution?

No: The open cells in the 2nd column can only be filled by 1 and 3. Neither of these fits into the 2nd row (v_{22}).

Can this partial assignment be extended to a solution?

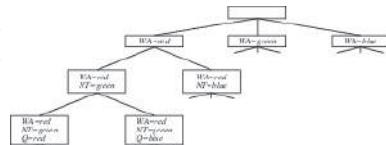
Yes: $v_{22} = 1$, $v_{32} = 3$, $v_{35} = 1$, $v_{25} = 2$, $v_{15} = 3$, $v_{11} = 2$, $v_{21} = 6$.



9.4 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 legal assignments (not fixable!)
 - ▷ **Goal test:** the current assignment is complete
- ▷ This is the same for all CSPs! \odot
- ▷ 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 $nd^n!$ leaves!!!! \odot



Backtracking search

- ▷ Variable assignments are **commutative**, i.e., $[WA = red \text{ then } NT = green]$ same as $[NT = green \text{ then } WA = red]$
- ▷ Only need to consider assignments to a single variable at each node
- ▷ $\implies b = d$ and there are d^n leaves
- ▷ **Definition 9.4.1** 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 $n \approx 25$



©: Michael Kohlhase

231



Backtracking search (Implementation)

```

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
  
```



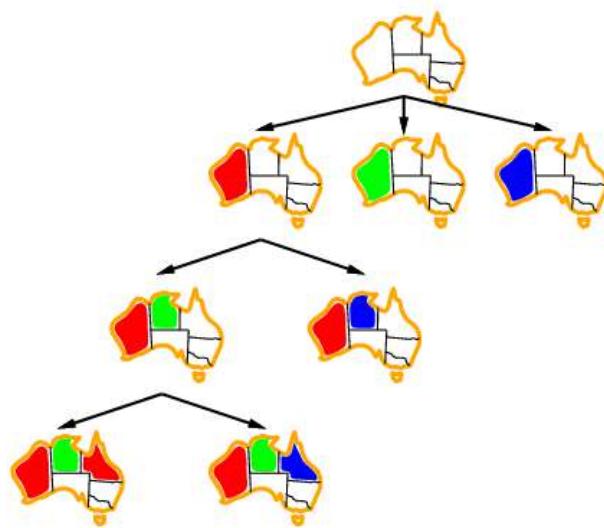
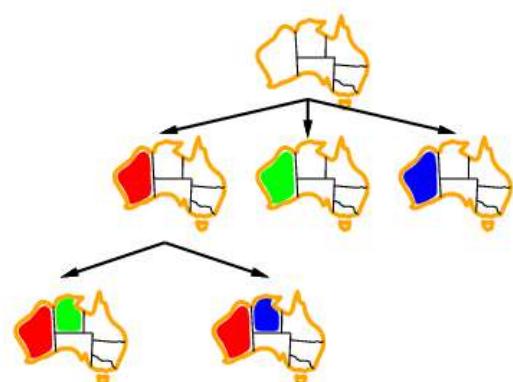
©: Michael Kohlhase

232



Backtracking in Australia



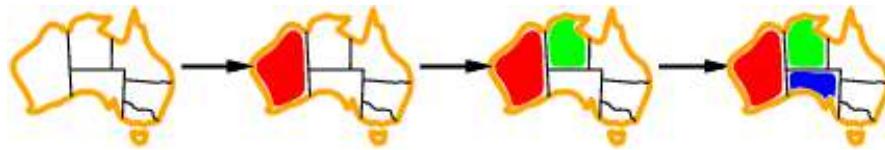


Improving backtracking efficiency

- ▷ General-purpose methods can give huge gains in speed:
 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?

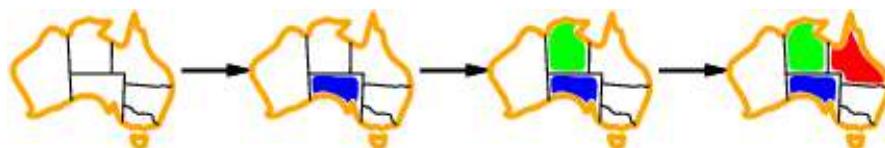
Heuristic: Minimum Remaining Values

- ▷ **Definition 9.4.2** 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 9.4.3** in step 3, there is only one remaining value for SA!



Degree heuristic

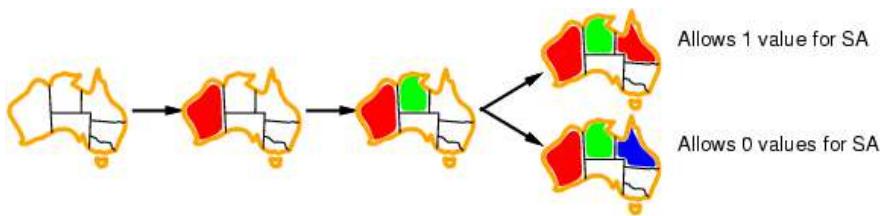
- ▷ **Problem:** Need a tie-breaker among MRV variables (there was no preference in step 1,2)
- ▷ **Definition 9.4.4** The **degree heuristic** in backtracking search always chooses the variable with the most constraints on remaining variables, ie. always pick a v with $\#\{\{v \in (V \setminus \text{dom}(a)) \mid C_{uv} \in C\}\}$ 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 9.4.5**



Degree Heuristic: SA = 5, T = 0, All others 2 or 3

Value Ordering: Least Constraining Value Heuristic

- ▷ **Definition 9.4.6** 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 9.4.7**



- ▷ Combining these heuristics makes 1000 queens feasible



Summary & Preview

- ▷ Summary of “CSP as Search”:
- ▷ **Constraint networks** γ consist of **variables**, associated with finite **domains**, and **constraints** which are binary relations specifying permissible value pairs.
- ▷ A **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.



Suggested Reading:

- ▷ *Chapter 6: Constraint Satisfaction Problems*, Sections 6.1 and 6.3 [RN09].
- **Content:** Compared to our treatment of the topic “Constraint Satisfaction Problems” (Chapters 7 and 8), RN covers much more material, but less formally and in much less detail (in

particular, my slides contain many additional in-depth examples). Nice background/additional reading, can't replace the lecture.

- Section 6.1: Similar to my “Introduction” and “Constraint Networks”, less/different examples, much less detail, more discussion of extensions/variations.
- Section 6.3: Similar to my “Naïve Backtracking” and “Variable- and Value Ordering”, with less examples and details; contains part of what I cover in Chapter 8 (RN does inference first, then backtracking). Additional discussion of *backjumping*.

Chapter 10

Constraint Propagation

10.1 Introduction

Illustration: Inference

▷ **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.



©: Michael Kohlhase

239



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.



10.2 Inference

Inference: Basic Facts

- ▷ **Definition 10.2.1** Inference consists in deducing additional constraints (unary or binary), that follow from the already known constraints, i.e. that are satisfied in all solutions.
- ▷ **Example 10.2.2** 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
7	6	2	4	9	8	1
8	4	1	6	5	3	7
1	8	4	3	6	9	5
5	7	6	1	4	2	8
9	2	3	5	8	7	6
						1

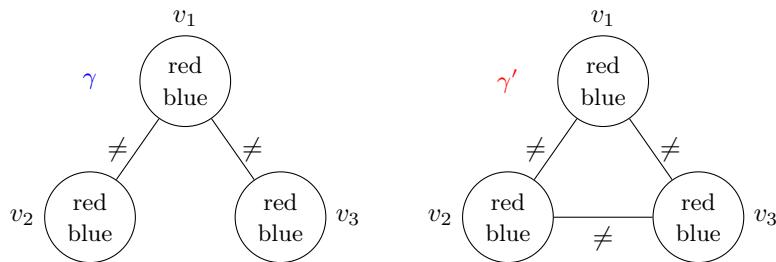
Formally: Replace γ by an equivalent and strictly tighter constraint network γ' .



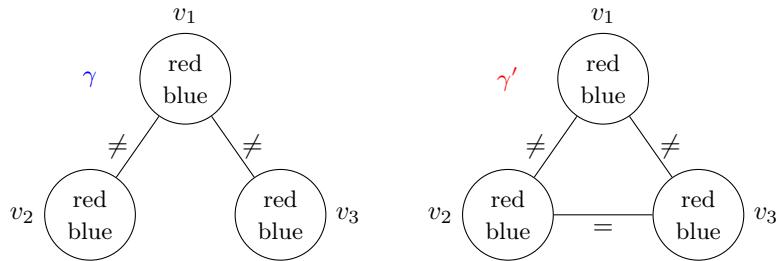
▷ Equivalent Constraint Networks

▷ **Definition 10.2.3** 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 10.2.4**



Are these constraint networks equivalent? No.



Are these constraint networks equivalent? Yes.



Tightness

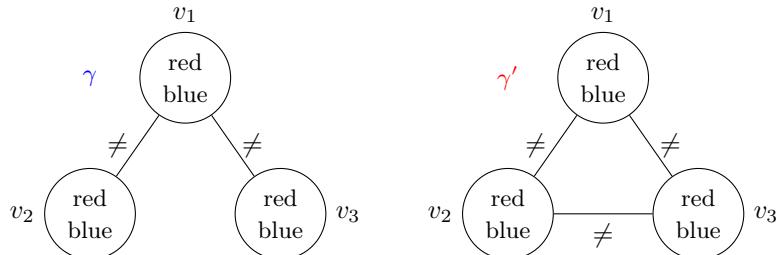
▷ **Definition 10.2.5 (Tightness)** Let $\gamma = \langle V, D, C \rangle$ and $\gamma' = \langle V, D', C' \rangle$ be

constraint networks sharing the same set of variables. We say that γ' is **tighter** than γ , (write $\gamma' \sqsubseteq \gamma$), if:

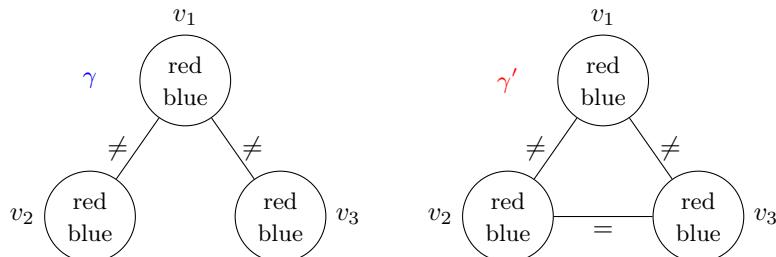
- (i) For all $v \in V$: $D'_v \subseteq D_v$.
- (ii) For all $u \neq v \in V$: either $C_{uv} \not\subseteq C$ or $C'_{uv} \subseteq C_{uv}$.

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

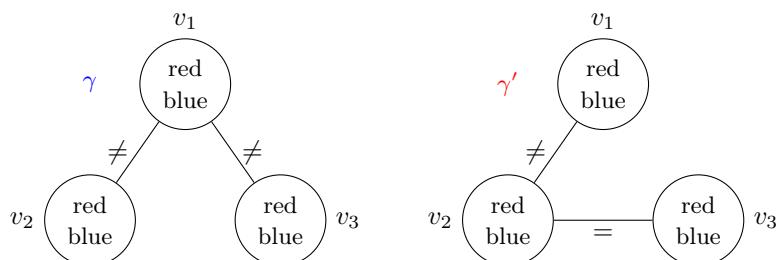
▷ **Example 10.2.6**



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



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



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

▷ **Tightness:** “ γ' has the same constraints as γ , plus some”.

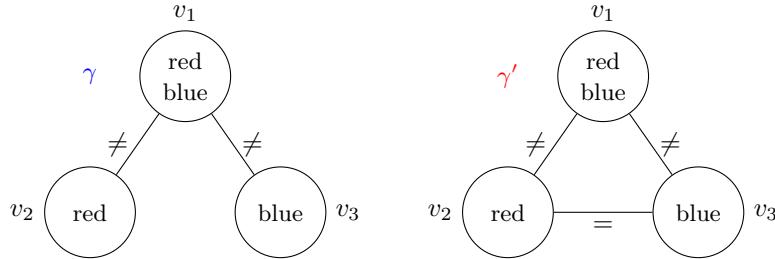


Equivalence + Tightness = Inference

▷ **Theorem 10.2.7** Let γ and γ' be constraint networks s.t. $\gamma' \equiv \gamma$ and $\gamma' \sqsubset \gamma$. Then γ' has the same solutions as, but fewer consistent partial assignments than, γ .

- ▷ $\leadsto \gamma'$ is a better encoding of the underlying problem.

▷ Example 10.2.8

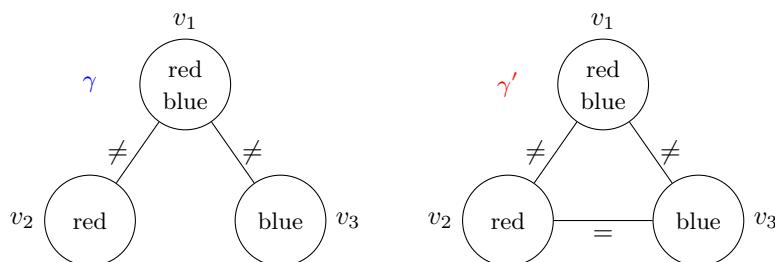


a cannot be extended to a solution (neither in γ nor in γ' because they're equivalent). a is consistent with γ , but not with γ' .



Equivalence + Tightness = Inference

- ▷ **Theorem 10.2.9** Let γ and γ' be constraint networks s.t. $\gamma' \equiv \gamma$ and $\gamma' \sqsubset \gamma$. Then γ' has the same solutions as, but fewer consistent partial assignments than, γ .
 - ▷ $\sim \gamma'$ is a better encoding of the underlying problem.



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?

- ## ▷ Inference as a pre-process:

- ▷ Just once before search starts.
 - ▷ Little runtime overhead, little pruning power. Not considered here.

- #### ▷ Inference during search:

- ▷ At every recursive call of backtracking.
- ▷ 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.

- ▷ 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

- ▷ The general algorithm for backtracking with inference

```
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  $\gamma$  /*  $\gamma' = (V, D', C')$  */
   $\gamma' :=$  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 */
     $a'' :=$  BacktrackingWithInference( $\gamma', a'$ )
    if  $a'' \neq$  "inconsistent" then return  $a''$ 
  return "inconsistent"
```

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



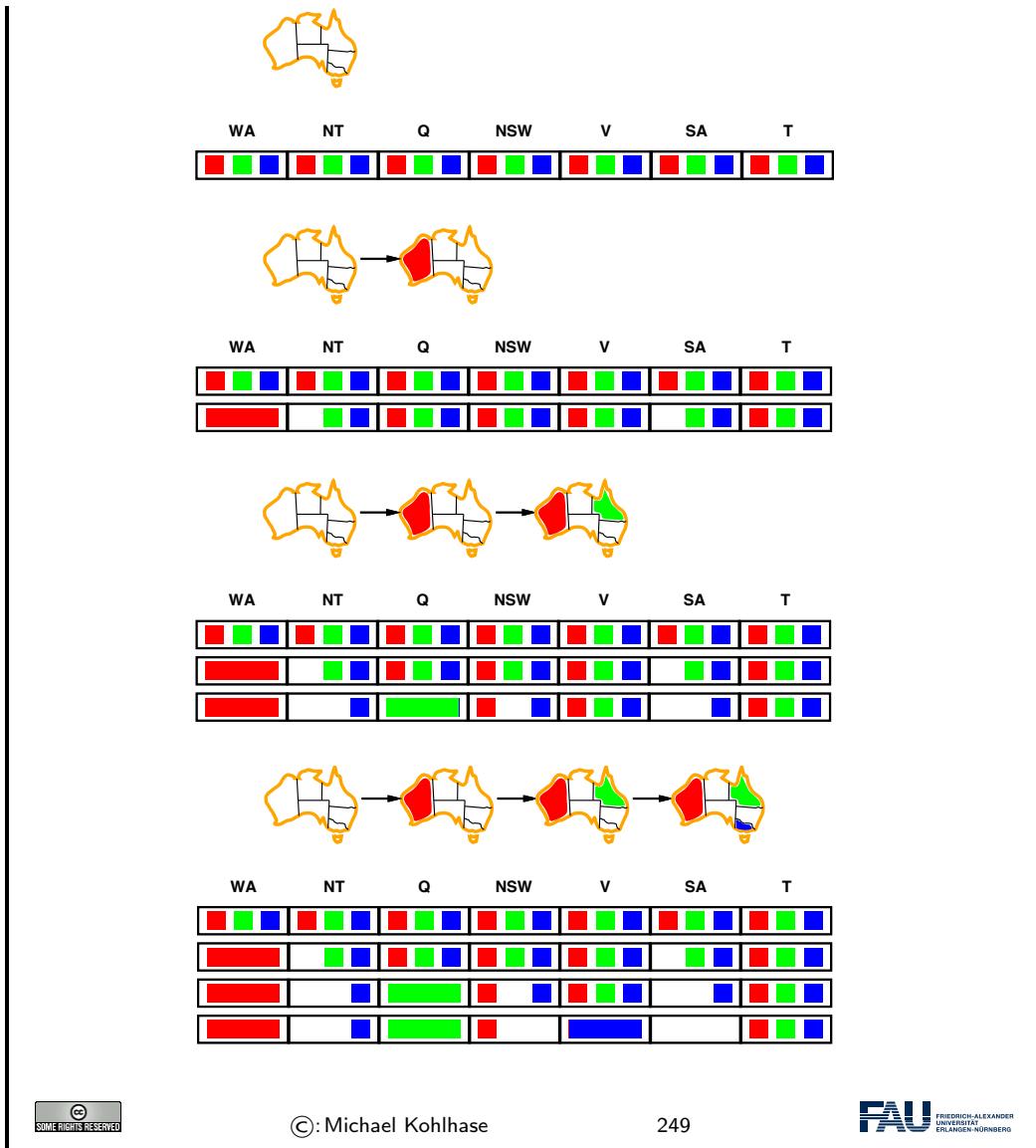
10.3 Forward Checking

Forward Checking

- ▷ **Inference, version 1:** Forward Checking

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

- ▷ Example 10.3.1



Forward Checking: Discussion

▷ **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 “ $\text{Inference}(\gamma')$ ” on slide 248.
- ▷ Incremental computation: Instead of the first for-loop, use only the 2nd one every time a new assignment $a(v) = d'$ is added.

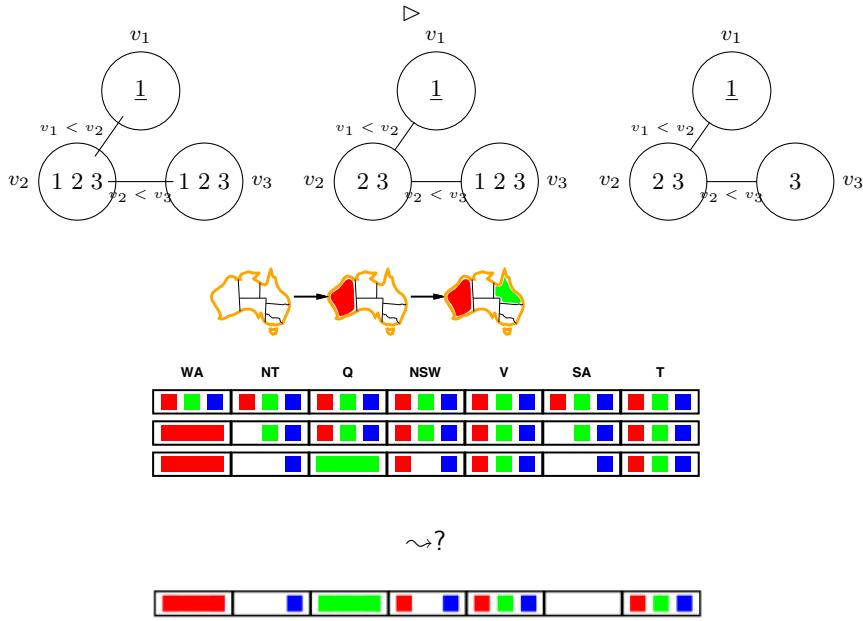
▷ **Practice:**

- ▷ 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).



10.4 Arc Consistency

When Forward Checking is Not Good Enough



- ▷ Forward checking makes inferences only “from assigned to unassigned” variables.



Arc Consistency: Definition

- ▷ **Definition 10.4.1 (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 $C_{uv} \notin C$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d, d') \in C_{uv}$.
 - (ii) The network γ is **arc consistent** if every variable $u \in V$ is arc consistent relative to every other variable $v \in V$.
- ▷ Arc consistency = 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”.
- ▷ **Examples:** (previous slide)
 - ▷ On top, middle, is v_3 arc consistent relative to v_2 ?
 - ▷ No. For values 1 and 2, D_{v_2} does not have a value that works.
 - ▷ And on the right?
 - ▷ Yes. (But v_2 is not arc consistent relative to v_3 .)
 - ▷ SA is not arc consistent relative to NT in the Australia example, 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: $AC(\gamma) \sqsubseteq ForwardChecking(\gamma)$.
(Recall from slide 244 that $\gamma' \sqsubseteq \gamma$ means γ' is tighter than γ .)
- ▷ Forward checking (c.f. slide 249) removes d from D_u only if there is a constraint C_{uv} such that $D_v = \{d'\}$ (when v was assigned the value d'), and $(d, d') \notin C_{uv}$. Clearly, enforcing arc consistency of u relative to v removes d from D_u as well.



Enforcing Arc Consistency for One Pair of Variables

- ▷ **Algorithm enforcing consistency of u relative to v :**

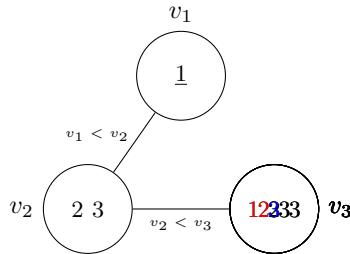
```

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  $\gamma$ 

```

- ▷ Runtime, if k is maximal domain size: $\mathcal{O}(k^2)$, based on implementation where the test “ $(d, d') \in C_{uv}$?” is constant time.

- ▷ **Example 10.4.2** $Revise(\gamma, v_3, v_2)$



AC-1

▷ Idea: Apply pairwise revisions up to a fixed point.

```
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  $\gamma$ 
```

▷ Obviously, this does indeed enforce arc consistency.

▷ Runtime, if n variables, m constraints, k maximal domain size: $\mathcal{O}(m k^2 n k)$:
 $m k^2$ for each inner loop, fixed point reached at the latest once all $n k$ variable values have been removed.

▷ Redundant computations: u and v are revised even if their domains haven't changed since the last time.

▷ Better algorithm avoiding this: AC-3 (coming up)

AC-3

▷ Idea: Remember the potentially inconsistent variable pairs.

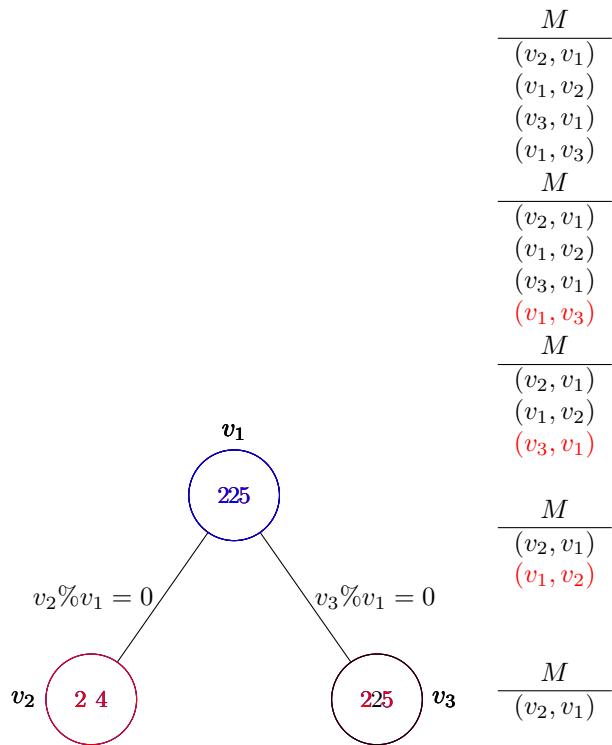
```
function AC-3( $\gamma$ ) returns modified  $\gamma$ 
 $M := \emptyset$ 
for each constraint  $C_{uv} \in C$  do
     $M := M \cup \{(u, v), (v, u)\}$ 
while  $M \neq \emptyset$  do
    remove any element  $(u, v)$  from  $M$ 
    Revise( $\gamma, u, v$ )
    if  $D_u$  has changed in the call to Revise then
        for each constraint  $C_{wu} \in C$  where  $w \neq v$  do
             $M := M \cup \{(w, u)\}$ 
return  $\gamma$ 
```

- ▷ AC-3(γ) enforces arc consistency because?
- ▷ At any time during the while-loop, if $(u, v) \notin M$ then u is arc consistent relative to v .
- ▷ Why only “where $w \neq v$ ”?
- ▷ If $w = v$ is the reason why D_u changed, then w is still arc consistent relative to u : the values just removed from D_u did not match any values from D_w anyway.



AC-3: Example

- ▷ **Example 10.4.3** $y \% x = 0$: y modulo x is 0, i.e., y can be divided by x



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

$$(v_3, v_1)$$

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

$$(v_3, v_1)$$

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

$$\underline{\underline{M}}$$

- ▷ **Theorem 10.4.4 (Runtime of AC-3)** Let $\gamma = \langle V, D, C \rangle$ be a constraint network with m constraints, and maximal domain size k . Then $AC-3(\gamma)$ runs in time $\mathcal{O}(m k^3)$.
- ▷ **Proof:** by counting how often Revise is called.
- P.1** Each call to $\text{Revise}(\gamma, u, v)$ takes time $\mathcal{O}(k^2)$ so it suffices to prove that at most $\mathcal{O}(m k)$ of these calls are made.
- P.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 .
- P.3** Consider any constraint C_{uv} .
- P.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
- P.5** beforehand the domain of either u or v was reduced, which happens at most $2k$ times.
- P.6** Thus we have $\mathcal{O}(k)$ insertions per constraint, and $O(mk)$ insertions overall, as desired. \square



10.5 Decomposition: Constraint Graphs, and Two Simple Cases

Reminder: The Big Picture

- ▷ 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, or at least ameliorate, this explosion in practice.
 - ▷ Often, from an assignment to some variables, we can easily make inferences regarding other variables.
- ▷ **Decomposition:** Another method to try to avoid, or at least 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.
 - ▷ **What is “structure”, and how to “decompose”?**



“Decomposition” 1.0: Disconnected Constraint Graphs

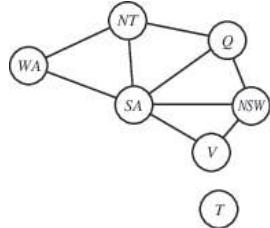
- ▷ **Theorem 10.5.1 (Disconnected Constraint Graphs)** Let $\gamma = \langle V, D, C \rangle$ be a constraint network. Let a_i be a solution to each connected component V_i of the constraint graph of γ . Then $a := \bigcup_i a_i$ is a solution to γ .
- ▷ **Proof:**

P.1 a satisfies all C_{uv} where u and v are inside the same connected component.

P.2 The latter is the case for all C_{uv} .

P.3 If two parts of γ are not connected, then they are independent. \square

▷ **Example 10.5.2**



Color Tasmania separately.

▷ 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 decomposition: $4 \cdot 2^{10}$. Gain: 2^{28} .



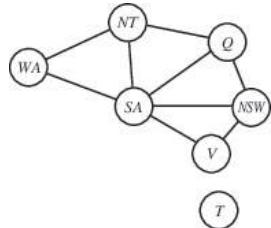
“Decomposition” 2.0: Acyclic Constraint Graphs

▷ **Theorem 10.5.3 (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 $\mathcal{O}(n k^2)$.

▷ **Proof Sketch:** See the algorithm on the next slide \square

▷ Constraint networks with acyclic constraint graphs can be solved in (low-order) polynomial time.

▷ **Example 10.5.4**



Not acyclic. But: see next section.

▷ γ with $n = 40$ variables, each domain size $k = 2$. Acyclic constraint graph.

▷ Reduction of worst-case when using decomposition:

▷ No decomposition: 2^{40} . With decomposition: $40 \cdot 2^2$. Gain: 2^{32} .



Acyclic Constraint Graphs: How To

▷ **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.^a

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**:
 - (a) $\text{Revise}(\gamma, v_{\text{parent}(i)}, v_i)$.
 - (b) **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 .

▷ This algorithm will find a solution without ever having to backtrack!

▷ Proof: Exercises



©: Michael Kohlhase

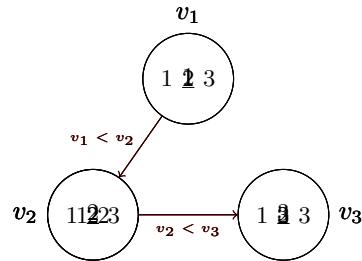
262



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

AcyclicCG(γ): Example

▷ Example 10.5.5 (AcyclicCG() execution)



Step 2: Order v_1, v_2, v_3 . Step 3: After $\text{Revise}(\gamma, v_2, v_3)$. Step 3: After $\text{Revise}(\gamma, v_1, v_2)$. Step 4: After $a(v_2) := 2$ and forward checking. Step 4: After $a(v_2) := 2$ and forward checking. Step 4: After $a(v_3) := 3$ (and forward checking).



©: Michael Kohlhase

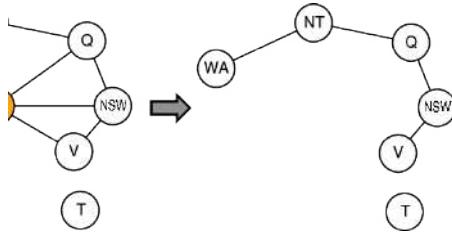
263



10.6 Cutset Conditioning

“Almost” Acyclic Constraint Graphs

▷ Example 10.6.1 (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 10.6.2 (Cutset)** Let $\gamma = \langle V, D, C \rangle$ be a constraint network, and $V_0 \subseteq V$. 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 **optimal** if its size is minimal among all cutsets for γ .

▷ Algorithm, compute $\text{CutsetConditioning}(\gamma, V_0, \emptyset)$, where V_0 is a cutset.

```

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  $a''$ 
  return "inconsistent"
```

▷ Forward Checking required so that " $a \cup \text{AcyclicCG}(\gamma')$ " is consistent in γ .

▷ Runtime is exponential only in $\#(V_0)$, not in $\#(V) \dots !$

▷ Finding optimal cutsets is **NP-hard**, but practical approximations exist.



10.7 Constraint Propagation with Local Search

Iterative algorithms for CSPs

- ▷ 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



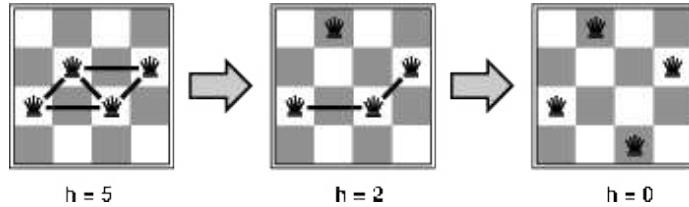
©: Michael Kohlhase

266



Example: 4-Queens

- ▷ **States:** 4 queens in 4 columns ($4^4 = 256$ states)
- ▷ **Operators:** move queen in column
- ▷ **Goal test:** no attacks
- ▷ **Evaluation:** $h(n)$ = number of attacks



©: Michael Kohlhase

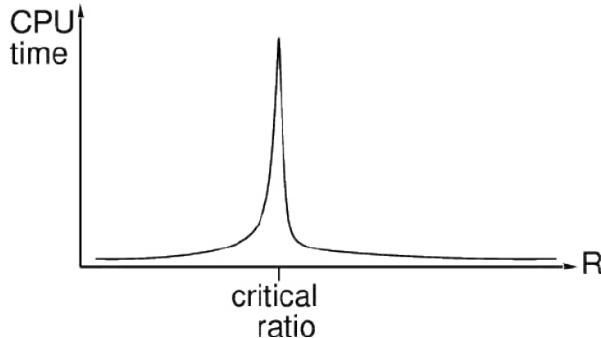
267



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

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



10.8 Conclusion & Summary

Conclusion & Summary

- ▷ γ and γ' are equivalent if they have the same solutions. γ' is tighter than γ if it is more constrained.
- ▷ Inference tightens γ without losing equivalence, during backtracking. 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: Generalizes arc consistency to size- k subsets of variables.
- ▷ 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 Chapter 7)
- ▷ **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.



Reading

- *Chapter 6: Constraint Satisfaction Problems* in [RN09], in particular Sections 6.2, 6.3.2, and 6.5.
- **Content:** Compared to our treatment of the topic “Constraint Satisfaction Problems” (Chapter 9 and Chapter 10), RN covers much more material, but less formally and in much less detail (in particular, my slides contain many additional in-depth examples). Nice background/additional reading, can’t replace the lecture.
- Section 6.3.2: Somewhat comparable to my “Inference” (except that equivalence and tightness are not made explicit in RN) together with “Forward Checking”.
- Section 6.2: Similar to my “Arc Consistency”, less/different examples, much less detail, additional discussion of path consistency and global constraints.
- Section 6.5: Similar to my “Decomposition: Constraint Graphs, and Two Simple Cases” and “Cutset Conditioning”, less/different examples, much less detail, additional discussion of tree decomposition.

Part III

Knowledge and Inference

This part of the course introduces representation languages and inference methods for structured state representations for agents: In contrast to the atomic and factored state representations from Part I, we look at state representations where the relations between objects are not determined by the problem statement, but can be determined by inference-based methods, where the knowledge about the environment is represented in a formal language and new knowledge is derived by transforming expressions of this language.

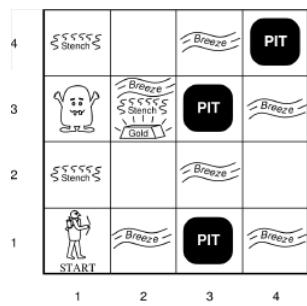
We look at propositional logic – a rather weak representation language – and first-order logic – a much stronger one – and study the respective inference procedures. In the end we show that computation in Prolog is just an inference problem as well.

Chapter 11

Propositional Reasoning, Part I: Principles

11.1 Introduction

The Wumpus World



▷ **Actions:** *GoForward*, *TurnRight* (by 90°), *TurnLeft* (by 90°), *Grab* object in current cell, *Shoot* arrow in direction you're facing (you got exactly one arrow), *Leave cave* if you're in cell [1,1].

▷ **Fall down Pit, meet live Wumpus:** Game Over.

▷ **Initial knowledge:** You're in cell [1,1] facing east. There's a Wumpus, and there's gold.

▷ **Goal:** Have the gold and be outside the cave.

▷ **Percepts:** [*Stench*, *Breeze*, *Glitter*, *Bump*, *Scream*]

- ▷ 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 11.1.1 (Reasoning in the Wumpus World)** A: Agent, V: Vis-

ited, **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
A OK	V OK	B OK	

(1) Initial state

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
A OK	V OK	A B OK	P? OK

(2) One step to right

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
V OK	A B OK	P? OK	

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

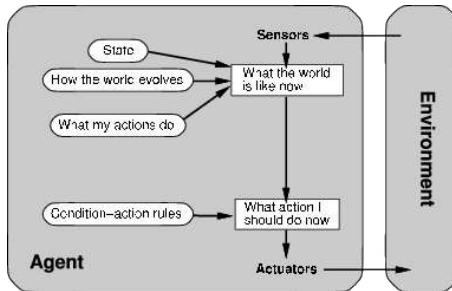
- ▷ The Wumpus is in [1,3]! How do we know?
- ▷ Because in [2,1] we perceived no Stench, the Stench in [1,2] can only come from [1,3].
- ▷ There's a Pit in [3,1]! How do we know?
- ▷ Because in [1,2] we perceived no Breeze, the Breeze in [2,1] can only come from [3,1].



Agents that Think Rationally

▷ Think Before You Act!

A **stateful reflex 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.



Logic: Basic Concepts

▷ **Representing Knowledge:**

- ▷ **Definition 11.1.2 Syntax:** What are legal statements (**formulas**) **A** in the logic?
- ▷ **Example 11.1.3** “W” and “ $W \Rightarrow S$ ”. $(W \hat{=} \text{Wumpus is here}, S \hat{=} \text{it stinks})$
- ▷ **Definition 11.1.4 Semantics:** Which formulas **A** are true under which **assignment** φ , written $\varphi \models A$?
- ▷ **Example 11.1.5** 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 **formulass**, **interpretations** evaluate them in the current world **(they should turn out true!)**

▷ **Reasoning about Knowledge:**

- ▷ **Definition 11.1.6 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 11.1.7 Deduction:** Which statements **B** can be **derived** from **A** using a set C of inference rules (a **calculus**), written $A \vdash_C B$?
- ▷ **Example 11.1.8** If C contains $\frac{A \quad A \Rightarrow B}{B}$ then $P, P \Rightarrow Q \vdash_C Q$
- ▷ **Intuition:** Deduction $\hat{=}$ process in an actual computer trying to reason about entailment. E.g. a mechanical process attempting to determine Wumpus position.



Logic: Basic Concepts (Properties of Deduction)

- ▷ **Definition 11.1.9** Calculus **soundness**: whenever $A \vdash_C B$, we also have $A \models B$.
- ▷ **Definition 11.1.10** Calculus **completeness**: whenever $A \models B$, we also have $A \vdash_C B$.
- ▷ **Question:** Can you give the intuition about soundness and completeness in the Wumpus example?
- ▷ **Answer (if they fail):**
 - ▷ **unsound** $\hat{=}$ conclude **Wumpus is here!** although it isn't;

▷ **incomplete:** unable to figure out where Wumpus is, even though we already have enough information to do so \leadsto I am unnecessarily stuck!

▷ **Answer (ethics):**

▷ **Soundness:** *I don't pretend to know where the Wumpus is, if I don't know*

▷ **Completeness:** *Whenever I have sufficient knowledge to know where the Wumpus is, my thinking process is strong enough to make that deduction*



General Problem Solving using Logic

▷ **Idea:** Any problem that can be formulated as reasoning about logic. \leadsto use off-the-shelf reasoning tool

▷ *Very successful using propositional logic and modern SAT solvers!* (Propositional satisfiability testing; Chapter 12)



Propositional Logic and Its Applications

▷ Propositional logic = canonical form of knowledge + reasoning.

▷ **Syntax:** Atomic **propositions** that can be either true or false, connected by “**and, or, 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 Chapter 9)

▷ Chapter 10 gives an example for verification.



Our Agenda for This Topic

▷ This Section: Basic definitions and concepts; tableaux, resolution.

▷ Sets up the framework. Resolution is the quintessential reasoning procedure

underlying most successful solvers.

- ▷ Chapter 12 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.



11.2 Propositional Logic (Syntax/Semantics)

Propositional Logic (Syntax)

- ▷ propositional logic (write PL^0) is made up from
 - ▷ propositional variables: $\mathcal{V}_o := \{P, Q, R, P^1, P^2, \dots\}$ (countably infinite)
 - ▷ connectives: $\Sigma_o := \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \dots\}$

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

- ▷ negations $\neg A$
- ▷ conjunctions $A \wedge B$
- ▷ disjunctions $A \vee B$
- ▷ implications $A \Rightarrow B$
- ▷ equivalences (or biimplications) $A \Leftrightarrow B$

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

- ▷ **Example 11.2.1** $P \wedge Q, P \vee Q, (\neg P \vee Q) \Leftrightarrow (P \Rightarrow Q) \in wff_o(\mathcal{V}_o)$

- ▷ **Definition 11.2.2** propositional formulae without connectives are called **atomic** (or **atoms**) and **complex** otherwise.



Alternative Notations for Connectives

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



Semantics (PL^0)

- ▷ **Definition 11.2.3** A **model** $\mathcal{M} := \langle \mathcal{D}_o, \mathcal{I} \rangle$ for propositional logic consists of

- ▷ the **Universe** $\mathcal{D}_o = \{\top, \perp\}$
- ▷ the **Interpretation** \mathcal{I} that assigns values to essential connectives
 - ▷ $\mathcal{I}(\neg): \mathcal{D}_o \rightarrow \mathcal{D}_o; \top \mapsto \perp, \perp \mapsto \top$
 - ▷ $\mathcal{I}(\wedge): \mathcal{D}_o \times \mathcal{D}_o \rightarrow \mathcal{D}_o; \langle \alpha, \beta \rangle \mapsto \top, \text{ iff } \alpha = \beta = \top$

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

▷ A **variable assignment** $\varphi: \mathcal{V}_o \rightarrow \mathcal{D}_o$ assigns values to propositional variables

▷ **Definition 11.2.4** The **value function** $\mathcal{I}_\varphi: wff_o(\mathcal{V}_o) \rightarrow \mathcal{D}_o$ assigns values to formulae.

- ▷ Recursively defined, base case: $\mathcal{I}_\varphi(P) = \varphi(P)$
- ▷ $\mathcal{I}_\varphi(\neg A) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(A))$
- ▷ $\mathcal{I}_\varphi(A \wedge B) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(A), \mathcal{I}_\varphi(B))$



We will now use the distribution of values of a Boolean expression under all (variable) assignments to characterize them semantically. The intuition here is that we want to understand theorems, examples, counterexamples, and inconsistencies in mathematics and everyday reasoning¹.

The idea is to use the formal language of Boolean expressions as a model for mathematical language. Of course, we cannot express all of mathematics as Boolean expressions, but we can at

¹Here (and elsewhere) we will use mathematics (and the language of mathematics) as a test tube for understanding reasoning, since mathematics has a long history of studying its own reasoning processes and assumptions.

least study the interplay of mathematical statements (which can be true or false) with the copula “and”, “or” and “not”.

Semantic Properties of Propositional Formulae

- ▷ **Definition 11.2.5** 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 $\mathcal{I}_\varphi(\mathbf{A}) = \top$ (write $\mathcal{M} \models^\varphi \mathbf{A}$)
 - ▷ false under φ (φ falsifies **A**) in \mathcal{M} , iff $\mathcal{I}_\varphi(\mathbf{A}) = \perp$ (write $\mathcal{M} \not\models^\varphi \mathbf{A}$)
 - ▷ satisfiable in \mathcal{M} , iff $\mathcal{I}_\varphi(\mathbf{A}) = \top$ for some assignment φ
 - ▷ valid in \mathcal{M} , iff $\mathcal{M} \models^\varphi \mathbf{A}$ for all assignments φ (write $\mathcal{M} \models \mathbf{A}$)
 - ▷ falsifiable in \mathcal{M} , iff $\mathcal{I}_\varphi(\mathbf{A}) = \perp$ for some assignments φ
 - ▷ unsatisfiable in \mathcal{M} , iff $\mathcal{I}_\varphi(\mathbf{A}) = \perp$ for all assignments φ
- ▷ **Example 11.2.6** $x \vee x$ is satisfiable and falsifiable.
- ▷ **Example 11.2.7** $x \vee \neg x$ is valid and $x \wedge \neg x$ is unsatisfiable.
- ▷ **Notation 11.2.8** (alternative) Write $[\mathbf{A}]_\varphi^{\mathcal{M}}$ for $\mathcal{I}_\varphi(\mathbf{A})$, if $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$. (and $[\mathbf{A}]^{\mathcal{M}}$, if **A** is ground, and $[\mathbf{A}]$, if \mathcal{M} is clear)
- ▷ **Definition 11.2.9 (Entailment)** (aka. logical consequence)

We say that **A** entails **B** ($\mathbf{A} \models \mathbf{B}$), iff $\mathcal{I}_\varphi(\mathbf{B}) = \top$ for all φ with $\mathcal{I}_\varphi(\mathbf{A}) = \top$
(i.e. all assignments that make **A** true also make **B** true)



Let us now see how these semantic properties model mathematical practice.

In mathematics we are interested in assertions that are true in all circumstances. In our model of mathematics, we use variable assignments to stand for circumstances. So we are interested in Boolean expressions which are true under all variable assignments; we call them valid. We often give examples (or show situations) which make a conjectured assertion false; we call such examples counterexamples, and such assertions “falsifiable”. We also often give examples for certain assertions to show that they can indeed be made true (which is not the same as being valid yet); such assertions we call “satisfiable”. Finally, if an assertion cannot be made true in any circumstances we call it “unsatisfiable”; such assertions naturally arise in mathematical practice in the form of refutation proofs, where we show that an assertion (usually the negation of the theorem we want to prove) leads to an obviously unsatisfiable conclusion, showing that the negation of the theorem is unsatisfiable, and thus the theorem valid.

A better mouse-trap: Truth Tables

- ▷ Truth tables to visualize truth functions:

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



Questionnaire

▷ There are three persons, Stefan (S), Nicole (N), and Jochen (J).

1. Their hair colors are black (*bla*), red (*red*), or green (*gre*).
2. Their study subjects are informatics (*inf*), physics (*phy*), or chinese (*chi*) at least one studies informatics.
 - (a) Persons with red or green hair do not study informatics.
 - (b) Neither the physics nor the chinese students have black hair.
 - (c) Of the two male persons, one studies physics, and the other studies chinese.

Question: Who studies informatics?

- (A) Stefan (B) Nicole (C) Jochen (C) Nobody

▷ **Answer:** You can solve this using propositional logic. We first express what we know: For every $x \in \{S, N, J\}$ we have

1. $\text{bla}(x) \vee \text{red}(x) \vee \text{gre}(x)$;
2. $\text{inf}(x) \vee \text{phy}(x) \vee \text{chi}(x)$ and $\text{inf}(S) \vee \text{inf}(N) \vee \text{inf}(J)$
 - (a) $\text{inf}(x) \Rightarrow \neg \text{red}(x) \wedge \neg \text{gre}(x)$.
 - (b) $\text{phy}(x) \Rightarrow \neg \text{bla}(x)$ and $\text{chi}(x) \Rightarrow \neg \text{bla}(x)$.
 - (c) $\text{phy}(S) \wedge \text{chi}(J) \vee \text{phy}(J) \wedge \text{chi}(S)$.

Now, we obtain new knowledge via entailment

3. 1. together with 2.(a) entails that $\text{inf}(x) \Rightarrow \text{bla}(x)$ for every $x \in \{S, N, J\}$,
4. thus $\neg \text{bla}(S) \wedge \neg \text{bla}(J)$ by 3. and 2.(b) and
5. so $\neg \text{inf}(S) \wedge \neg \text{inf}(J)$ by 3. and 4.
6. With 2.(c) the latter entails $\text{inf}(N)$.

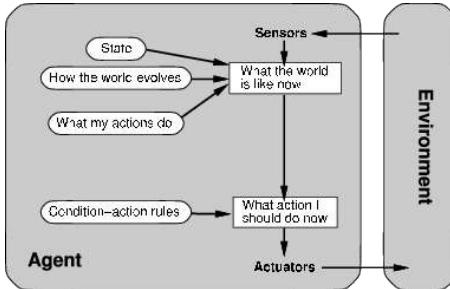


We have now defined syntax (the language agents can use to represent knowledge) and its semantics (how expressions of this language relate to the world the agent's environment). Theoretically, an agent could use the entailment relation to derive new knowledge percepts and the existing state representation – in the **MAKE–PERCEPT–SENTENCE** and **MAKE–ACTION–SENTENCE** subroutines below. But as we have seen in above, this is very tedious. A much better way would be to have a set of rules that directly act on the state representations.

Agents that Think Rationally

▷ Think Before You Act!:

A **stateful reflex 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-PERCEP-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.

Before we do that in the form of logical calculi, we will take a more abstract view and introduce the necessary prerequisites of abstract rule systems. We will also take the opportunity to discuss the quality criteria for such rule systems.

11.3 Formal Systems (Syntax and Semantics in General)

Recap: General Aspects of Propositional Logic

▷ There are many ways to define Propositional Logic:

- ▷ We chose \wedge and \neg as primitive, and many others as defined.
- ▷ We could have used \vee and \neg just as well.
- ▷ We could even have used only one connective – e.g. negated conjunction \uparrow or disjunction \downarrow and defined \wedge , \vee , and \neg via \uparrow and \downarrow respectively.

\uparrow	T	F	\downarrow	T	F	$\neg a$	$a \uparrow a$	$a \downarrow a$
F	T	T	F	T	F	$\neg a$	$(a \uparrow b) \uparrow (a \uparrow b)$	$(a \downarrow a) \downarrow (b \downarrow b)$
T	T	F	T	F	F	$a \vee b$	$(a \uparrow a) \uparrow (b \uparrow b)$	$(a \downarrow b) \downarrow (a \downarrow b)$

▷ Observation: The set $wff_o(\mathcal{V}_o)$ of well-formed propositional formulas 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 $\mathbf{A} \models \mathbf{B}$.
- ▷ **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.



The notion of a **logical system** is at the basis of the field of logic. In its most abstract form, a logical system consists of a formal language, a class of models, and a satisfaction relation between models and expressions of the formal language. The satisfaction relation tells us when an expression is deemed true in this model.

Logical Systems

- ▷ **Definition 11.3.1** 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 **formulae** of \mathcal{S} , members of \mathcal{K} **models** for \mathcal{S} , and \models the **satisfaction relation**.
- ▷ **Example 11.3.2 (Propositional Logic)**
 $\langle wff_o(\mathcal{V}_o), \mathcal{K}, \models \rangle$ is a logical system, if we define $\mathcal{K} := \mathcal{V}_o \rightharpoonup \mathcal{D}_o$ (the set of variable assignments) and $\varphi \models \mathbf{A} :\Leftrightarrow \mathcal{I}_\varphi(\mathbf{A}) = \top$.
- ▷ **Definition 11.3.3** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, $\mathcal{M} \in \mathcal{K}$ be a model and $\mathbf{A} \in \mathcal{L}$ a formula, then we call \mathbf{A}
 - ▷ **satisfied by** \mathcal{M} , iff $\mathcal{M} \models \mathbf{A}$
 - ▷ **falsified by** \mathcal{M} , iff $\mathcal{M} \not\models \mathbf{A}$
 - ▷ **satisfiable** in \mathcal{K} , iff $\mathcal{M} \models \mathbf{A}$ for some model $\mathcal{M} \in \mathcal{K}$.
 - ▷ **valid** in \mathcal{K} (write $\models \mathcal{M}$), iff $\mathcal{M} \models \mathbf{A}$ for all models $\mathcal{M} \in \mathcal{K}$
 - ▷ **falsifiable** in \mathcal{K} , iff $\mathcal{M} \not\models \mathbf{A}$ for some $\mathcal{M} \in \mathcal{K}$.
 - ▷ **unsatisfiable** in \mathcal{K} , iff $\mathcal{M} \not\models \mathbf{A}$ for all $\mathcal{M} \in \mathcal{K}$.



Let us now turn to the syntactical counterpart of the entailment relation: derivability in a calculus. Again, we take care to define the concepts at the general level of logical systems.

The intuition of a calculus is that it provides a set of syntactic rules that allow to reason by considering the form of propositions alone. Such rules are called inference rules, and they can be strung together to derivations — which can alternatively be viewed either as sequences of formulae where all formulae are justified by prior formulae or as trees of inference rule applications. But we can also define a calculus in the more general setting of logical systems as an arbitrary relation on formulae with some general properties. That allows us to abstract away from the homomorphic setup of logics and calculi and concentrate on the basics.

Derivation Systems and Inference Rules

- ▷ **Definition 11.3.4** 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. $\mathcal{H} \vdash \mathbf{A}$, if $\mathbf{A} \in \mathcal{H}$;

- ▷ is **proof-transitive**, i.e. if $\mathcal{H} \vdash \mathbf{A}$ and $\mathcal{H}' \cup \{\mathbf{A}\} \vdash \mathbf{B}$, then $\mathcal{H} \cup \mathcal{H}' \vdash \mathbf{B}$;
 - ▷ **monotonic** (or **admits weakening**), i.e. $\mathcal{H} \vdash \mathbf{A}$ and $\mathcal{H} \subseteq \mathcal{H}'$ imply $\mathcal{H}' \vdash \mathbf{A}$.
 - ▷ **Definition 11.3.5** 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 11.3.6** Let \mathcal{L} be a formal language, then an **inference rule** over \mathcal{L}
- $$\frac{\mathbf{A}_1 \quad \dots \quad \mathbf{A}_n}{\mathbf{C}} \mathcal{N}$$
- where $\mathbf{A}_1, \dots, \mathbf{A}_n$ and \mathbf{C} are formula schemata for \mathcal{L} and \mathcal{N} is a name. The \mathbf{A}_i are called **assumptions**, and \mathbf{C} is called **conclusion**.
- ▷ **Definition 11.3.7** An inference rule without assumptions is called an **axiom** (schema).
 - ▷ **Definition 11.3.8** 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} .



With formula schemata we mean representations of sets of formulae, we use boldface uppercase letters as (meta)-variables for formulae, for instance the formula schema $\mathbf{A} \Rightarrow \mathbf{B}$ represents the set of formulae whose head is \Rightarrow .

Derivations and Proofs

- ▷ **Definition 11.3.9** 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 $\mathbf{C} \in \mathcal{L}$ from a set $\mathcal{H} \subseteq \mathcal{L}$ of **hypotheses** (write $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{C}$) is a sequence $\mathbf{A}_1, \dots, \mathbf{A}_m$ of \mathcal{L} -formulae, such that
 - ▷ $\mathbf{A}_m = \mathbf{C}$, (derivation culminates in \mathbf{C})
 - ▷ for all $1 \leq i \leq m$, either $\mathbf{A}_i \in \mathcal{H}$, or (hypothesis)
 - ▷ there is an inference rule $\frac{\mathbf{A}_{l_1} \quad \dots \quad \mathbf{A}_{l_k}}{\mathbf{A}_i}$ in \mathcal{C} with $l_j < i$ for all $j \leq k$. (rule application)

Observation: We can also see a derivation as a tree, where the \mathbf{A}_{l_j} are the children of the node \mathbf{A}_k .

▷ Example 11.3.10

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.

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



Inference rules are relations on formulae represented by formula schemata (where boldface, uppercase letters are used as meta-variables for formulae). For instance, in Example 11.3.10 the inference rule $\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}$ was applied in a situation, where the meta-variables \mathbf{A} and \mathbf{B} were instantiated by the formulae P and $Q \Rightarrow P$.

As axioms do not have assumptions, they can be added to a derivation at any time. This is just what we did with the axioms in Example 11.3.10.

Formal Systems

- ▷ **Observation 11.3.11** 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 $\vdash_{\mathcal{D}}$ defined in Definition 11.3.9 is a derivation relation in the sense of Definition 11.3.4.²
- ▷ **Definition 11.3.12** We 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 11.3.13** A derivation $\emptyset \vdash_{\mathcal{C}} \mathbf{A}$ is called a **proof** of \mathbf{A} and if one exists (write $\vdash_{\mathcal{C}} \mathbf{A}$) then \mathbf{A} is called a \mathcal{C} -**theorem**.
- ▷ **Definition 11.3.14** 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.



©: Michael Kohlhase

291



^bEDNOTE: MK: this should become a view!

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

- ▷ **Formulae:** built from **prop. variables**: P, Q, R, \dots and **implication**: \Rightarrow
- ▷ **Semantics:** $\mathcal{I}_{\varphi}(P) = \varphi(P)$ and $\mathcal{I}_{\varphi}(\mathbf{A} \Rightarrow \mathbf{B}) = \top$, iff $\mathcal{I}_{\varphi}(\mathbf{A}) = \mathsf{F}$ or $\mathcal{I}_{\varphi}(\mathbf{B}) = \top$.
- ▷ $\mathbf{K} := P \Rightarrow Q \Rightarrow P$, $\mathbf{S} := (P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R$

$$\triangleright \frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}} \text{ MP} \qquad \frac{\mathbf{A}}{[\mathbf{B}/X](\mathbf{A})} \text{ Subst}$$

▷ Let us look at a \mathcal{H}^0 theorem (with a proof)

▷ $\mathbf{C} \Rightarrow \mathbf{C}$ (Tertium non datur)

▷ **Proof:**

P.1 $(\mathbf{C} \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C}) \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}$ (S with $[\mathbf{C}/P], [\mathbf{C} \Rightarrow \mathbf{C}/Q], [\mathbf{C}/R]$)

P.2 $\mathbf{C} \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C}$ (K with $[\mathbf{C}/P], [\mathbf{C} \Rightarrow \mathbf{C}/Q]$)

P.3 $(\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}$ (MP on P.1 and P.2)

P.4 $\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}$ (K with $[\mathbf{C}/P], [\mathbf{C}/Q]$)

P.5 $\mathbf{C} \Rightarrow \mathbf{C}$ (MP on P.3 and P.4)

P.6 We have shown that $\emptyset \vdash_{\mathcal{H}^0} \mathbf{C} \Rightarrow \mathbf{C}$ (i.e. $\mathbf{C} \Rightarrow \mathbf{C}$ is a **theorem**) (is also valid?)

□



©: Michael Kohlhase

292



This is indeed a very simple formal system, but it has all the required parts:

- A formal language: expressions built up from variables and implications.
- A semantics: given by the obvious interpretation function
- A calculus: given by the two axioms and the two inference rules.

The calculus gives us a set of rules with which we can derive new formulae from old ones. The axioms are very simple rules, they allow us to derive these two formulae in any situation. The inference rules are slightly more complicated: we read the formulae above the horizontal line as assumptions and the (single) formula below as the conclusion. An inference rule allows us to derive the conclusion, if we have already derived the assumptions.

Now, we can use these inference rules to perform a proof. A proof is a sequence of formulae that can be derived from each other. The representation of the proof in the slide is slightly compactified to fit onto the slide: We will make it more explicit here. We first start out by deriving the formula

$$(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R \quad (11.1)$$

which we can always do, since we have an axiom for this formula, then we apply the rule *subst*, where **A** is this result, **B** is **C**, and *X* is the variable *P* to obtain

$$(\mathbf{C} \Rightarrow Q \Rightarrow R) \Rightarrow (\mathbf{C} \Rightarrow Q) \Rightarrow \mathbf{C} \Rightarrow R \quad (11.2)$$

Next we apply the rule *subst* to this where **B** is **C** \Rightarrow **C** and *X* is the variable *Q* this time to obtain

$$(\mathbf{C} \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow R) \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C} \Rightarrow R \quad (11.3)$$

And again, we apply the rule *subst* this time, **B** is **C** and *X* is the variable *R* yielding the first formula in our proof on the slide. To conserve space, we have combined these three steps into one in the slide. The next steps are done in exactly the same way.

In general formulae can be used to represent facts about the world as propositions; they have a semantics that is a mapping of formulae into the real world (propositions are mapped to truth values.) We have seen two relations on formulae: the entailment relation and the deduction relation. The first one is defined purely in terms of the semantics, the second one is given by a calculus, i.e. purely syntactically. Is there any relation between these relations?

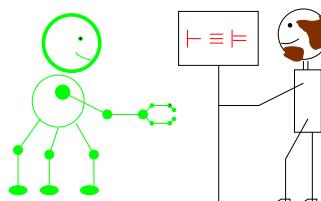
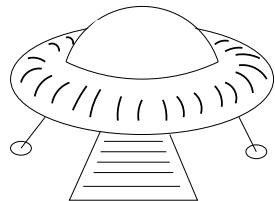
Soundness and Completeness

▷ **Definition 11.3.15** 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 $\mathcal{H} \models \mathbf{A}$, whenever $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$, and
- ▷ **complete**, iff $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$, whenever $\mathcal{H} \models \mathbf{A}$.

▷ Goal: $\vdash \mathbf{A}$ iff $\models \mathbf{A}$ (provability and validity coincide)

▷ **To TRUTH through PROOF** (CALCULEMUS [Leibniz ~1680])





Ideally, both relations would be the same, then the calculus would allow us to infer all facts that can be represented in the given formal language and that are true in the real world, and only those. In other words, our representation and inference is faithful to the world.

A consequence of this is that we can rely on purely syntactical means to make predictions about the world. Computers rely on formal representations of the world; if we want to solve a problem on our computer, we first represent it in the computer (as data structures, which can be seen as a formal language) and do syntactic manipulations on these structures (a form of calculus). Now, if the provability relation induced by the calculus and the validity relation coincide (this will be quite difficult to establish in general), then the solutions of the program will be correct, and we will find all possible ones.

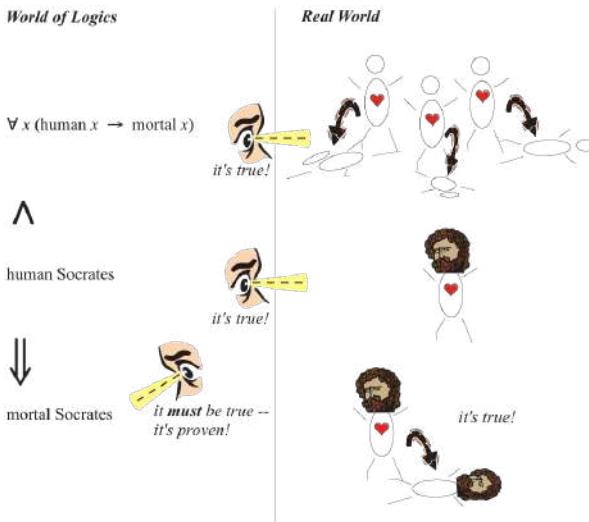
Of course, the logics we have studied so far are very simple, and not able to express interesting facts about the world, but we will study them as a simple example of the fundamental problem of Computer Science: How do the formal representations correlate with the real world.

Within the world of logics, one can derive new propositions (the *conclusions*, here: *Socrates is mortal*) from given ones (the *premises*, here: *Every human is mortal* and *Sokrates is human*). Such derivations are *proofs*.

In particular, logics can describe the internal structure of real-life facts; e.g. individual things, actions, properties. A famous example, which is in fact as old as it appears, is illustrated in the slide below.

The miracle of logics

▷ Purely formal derivations are true in the real world!



If a logic is correct, the conclusions one can prove are true (= hold in the real world) whenever the premises are true. This is a miraculous fact (think about it!)

11.4 Propositional Natural Deduction Calculus

We will now introduce the “natural deduction” calculus for propositional logic. The calculus was

created in order to model the natural mode of reasoning e.g. in everyday mathematical practice. This calculus was intended as a counter-approach to the well-known Hilbert style calculi, which were mainly used as theoretical devices for studying reasoning in principle, not for modeling particular reasoning styles.

Rather than using a minimal set of inference rules, the natural deduction calculus provides two/three inference rules for every connective and quantifier, one “introduction rule” (an inference rule that derives a formula with that symbol at the head) and one “elimination rule” (an inference rule that acts on a formula with this head and derives a set of subformulae).

Calculi: Natural Deduction (\mathcal{ND}^0 ; Gentzen [Gen34])

▷ Idea: \mathcal{ND}^0 tries to mimic human theorem proving behavior (non-minimal)

▷ Definition 11.4.1 The propositional natural deduction calculus \mathcal{ND}^0 has rules for the introduction and elimination of connectives

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

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



The most characteristic rule in the natural deduction calculus is the $\Rightarrow I$ rule. It corresponds to the mathematical way of proving an implication $\mathbf{A} \Rightarrow \mathbf{B}$: We assume that \mathbf{A} is true and show \mathbf{B} from this assumption. When we can do this we discharge (get rid of) the assumption and conclude $\mathbf{A} \Rightarrow \mathbf{B}$. This mode of reasoning is called **hypothetical reasoning**. Note that the local hypothesis is **discharged** by the rule $\Rightarrow I$, i.e. it cannot be used in any other part of the proof. As the $\Rightarrow I$ rules may be nested, we decorate both the rule and the corresponding assumption with a marker (here the number 1).

Let us now consider an example of hypothetical reasoning in action.

Natural Deduction: Examples

▷ Example 11.4.2 (Inference with Local Hypotheses)

$$\begin{array}{c}
 \frac{[\mathbf{A} \wedge \mathbf{B}]^1}{\mathbf{B}} \wedge E_r \quad \frac{[\mathbf{A} \wedge \mathbf{B}]^1}{\mathbf{A}} \wedge E_l \\
 \hline
 \frac{}{\mathbf{B} \wedge \mathbf{A}} \wedge I \\
 \hline
 \mathbf{A} \wedge \mathbf{B} \Rightarrow \mathbf{B} \wedge \mathbf{A} \Rightarrow I^1
 \end{array}
 \qquad
 \begin{array}{c}
 \frac{[A]^1}{[B]^2} \\
 \frac{A}{B \Rightarrow A} \Rightarrow I^2 \\
 \hline
 \frac{}{A \Rightarrow B \Rightarrow A} \Rightarrow I^1
 \end{array}$$



Here we see reasoning with local hypotheses at work. In the left example, we assume the formula $\mathbf{A} \wedge \mathbf{B}$ and can use it in the proof until it is discharged by the rule $\wedge E_l$ on the bottom – therefore we decorate the hypothesis and the rule by corresponding numbers (here the label “1”). Note the assumption $\mathbf{A} \wedge \mathbf{B}$ is *local to the proof fragment* delineated by the corresponding hypothesis and the discharging rule, i.e. even if this proof is only a fragment of a larger proof, then we cannot use its hypothesis anywhere else. Note also that we can use as many copies of the local hypothesis as we need; they are all discharged at the same time.

In the right example we see that local hypotheses can be nested as long as hypotheses are kept local. In particular, we may not use the hypothesis \mathbf{B} after the $\Rightarrow I^2$, e.g. to continue with a $\Rightarrow E$.

One of the nice things about the natural deduction calculus is that the deduction theorem is almost trivial to prove. In a sense, the triviality of the deduction theorem is the central idea of the calculus and the feature that makes it so natural.

A Deduction Theorem for \mathcal{ND}^0

▷ **Theorem 11.4.3** $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$, iff $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$.

▷ **Proof:** We show the two directions separately

P.1 If $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$, then $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$ by $\Rightarrow I$, and

P.2 If $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$, then $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$ by weakening and $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$ by $\Rightarrow E$. □



Another characteristic of the natural deduction calculus is that it has inference rules (introduction and elimination rules) for all connectives. So we extend the set of rules from Definition 13.3.1 for disjunction, negation and falsity.

More Rules for Natural Deduction

▷ **Definition 11.4.4** \mathcal{ND}^0 has the following additional rules for the remaining connectives.

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

$$\frac{\begin{array}{c} [\mathbf{A}]^1 \\ \vdots \\ \mathbf{F} \end{array}}{\neg \mathbf{A}} \neg I^1 \quad \frac{\neg \neg \mathbf{A}}{\mathbf{A}} \neg E$$

$$\frac{\neg \mathbf{A} \quad \mathbf{A}}{F} FI \quad \frac{F}{\mathbf{A}} FE$$



Natural Deduction in Sequent Calculus Formulation

- ▷ Idea: Explicit representation of hypotheses (lift calculus to judgments)
- ▷ **Definition 11.4.5** A judgment is a meta-statement about the provability of propositions
- ▷ **Definition 11.4.6** A sequent is a judgment of the form $\mathcal{H} \vdash \mathbf{A}$ about the provability of the formula \mathbf{A} from the set \mathcal{H} of hypotheses.
Write $\vdash \mathbf{A}$ for $\emptyset \vdash \mathbf{A}$.
- ▷ Idea: Reformulate ND rules so that they act on sequents
- ▷ **Example 11.4.7** We give the sequent-style version of Example 13.3.2

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

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 11.4.8** The following inference rules make up the propositional sequent-style natural deduction calculus \mathcal{ND}_\vdash^0 :

$$\begin{array}{ccc}
 \frac{}{\Gamma, \mathbf{A} \vdash \mathbf{A}} \text{ Ax} & \frac{\Gamma \vdash \mathbf{B}}{\Gamma, \mathbf{A} \vdash \mathbf{B}} \text{ weaken} & \frac{}{\Gamma \vdash \mathbf{A} \vee \neg \mathbf{A}} \text{ TND} \\
 \frac{\Gamma \vdash \mathbf{A} \quad \Gamma \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}} \wedge I & \frac{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}{\Gamma \vdash \mathbf{A}} \wedge E_l & \frac{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}{\Gamma \vdash \mathbf{B}} \wedge E_r \\
 \frac{\Gamma \vdash \mathbf{A}}{\Gamma \vdash \mathbf{A} \vee \mathbf{B}} \vee I_l & \frac{\Gamma \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \vee \mathbf{B}} \vee I_r & \frac{\Gamma \vdash \mathbf{A} \vee \mathbf{B} \quad \Gamma, \mathbf{A} \vdash \mathbf{C} \quad \Gamma, \mathbf{B} \vdash \mathbf{C}}{\Gamma \vdash \mathbf{C}} \vee E \\
 \frac{\Gamma, \mathbf{A} \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \Rightarrow \mathbf{B}} \Rightarrow I & \frac{\Gamma \vdash \mathbf{A} \Rightarrow \mathbf{B} \quad \Gamma \vdash \mathbf{A}}{\Gamma \vdash \mathbf{B}} \Rightarrow E & \\
 \frac{\Gamma, \mathbf{A} \vdash F}{\Gamma \vdash \neg \mathbf{A}} \neg I & \frac{\Gamma \vdash \neg \neg \mathbf{A}}{\mathbf{A}} \neg E & \\
 \frac{\Gamma \vdash \neg \mathbf{A} \quad \Gamma \vdash \mathbf{A}}{\Gamma \vdash F} FI & \frac{\Gamma \vdash F}{\Gamma \vdash \mathbf{A}} FE &
 \end{array}$$



Linearized Notation for (Sequent-Style) ND Proofs

▷ Linearized notation for sequent-style ND proofs

1. $\mathcal{H}_1 \vdash \mathbf{A}_1$ (\mathcal{J}_1)
 2. $\mathcal{H}_2 \vdash \mathbf{A}_2$ (\mathcal{J}_2)
 3. $\mathcal{H}_3 \vdash \mathbf{A}_3$ ($\mathcal{R}1, 2$)
- corresponds to
$$\frac{\mathcal{H}_1 \vdash \mathbf{A}_1 \quad \mathcal{H}_2 \vdash \mathbf{A}_2}{\mathcal{H}_3 \vdash \mathbf{A}_3} \mathcal{R}$$

▷ **Example 11.4.9** We show a linearized version of Example 13.3.7

#	hyp	\vdash	formula	NDjust	#	hyp	\vdash	formula	NDjust
1.	1	\vdash	$\mathbf{A} \wedge \mathbf{B}$	Ax	1.	1	\vdash	\mathbf{A}	Ax
2.	1	\vdash	\mathbf{B}	$\wedge E_r 1$	2.	2	\vdash	\mathbf{B}	Ax
3.	1	\vdash	\mathbf{A}	$\wedge E_l 1$	3.	1, 2	\vdash	\mathbf{A}	weaken 1, 2
4.	1	\vdash	$\mathbf{B} \wedge \mathbf{A}$	$\wedge I2, 1$	4.	1	\vdash	$\mathbf{B} \Rightarrow \mathbf{A}$	$\Rightarrow I3$
5.		\vdash	$\mathbf{A} \wedge \mathbf{B} \Rightarrow \mathbf{B} \wedge \mathbf{A}$	$\Rightarrow I4$	5.		\vdash	$\mathbf{A} \Rightarrow \mathbf{B} \Rightarrow \mathbf{A}$	$\Rightarrow I4$



Each line in the table represents one inference step in the proof. It consists of line number (for referencing), a formula for the asserted property, a justification via a ND rules (and the lines this one is derived from), and finally a list of line numbers of proof steps that are local hypotheses in effect for the current line.

11.5 Machine-Oriented Calculi for Propositional Logic

Automated Deduction as an Inference Procedure

▷ **Recall:** Our knowledge of the cave entails a definite Wumpus position! (slide ??)

▷ **Problem:** How to find out about this?

▷ **Answer:** Inference, here resolution/tableau



The following theorem is simple, but will be crucial later on.

Unsatisfiability Theorem

▷ **Theorem 11.5.1 (Unsatisfiability Theorem)** $\mathcal{H} \models \mathbf{A}$ iff $\mathcal{H} \cup \{\neg \mathbf{A}\}$ is unsatisfiable.

▷ **Proof:**

P.1 We prove both directions separately

P.1.1 \Rightarrow : Say $\mathcal{H} \models \mathbf{A}$: For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \models \mathbf{A}$ and thus $\varphi \not\models (\neg \mathbf{A})$. \square

P.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 11.5.2** Entailment can be tested via satisfiability.



Normal Forms

▷ **The two quintessential normal forms:** (there are others as well)

▷ **Definition 11.5.3** 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 11.5.4** 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 11.5.5** Every formula has equivalent formulas in CNF and DNF.



11.5.1 Calculi for Automated Theorem Proving: Analytical Tableaux

Analytical Tableaux

Before we can start, we will need to recap some nomenclature on formulae.

Recap: Atoms and Literals

▷ **Definition 11.5.6** 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 11.5.7** We call a pair A^α a **labeled formula**, if $\alpha \in \{T, F\}$. A labeled atom is called **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 11.5.8** For a literal A^α , we call the literal A^β with $\alpha \neq \beta$ the **opposite literal** (or **partner literal**).

▷ **Definition 11.5.9** Let Φ be a set of formulae, then we use $\Phi^\alpha := \{A^\alpha \mid A \in \Phi\}$.



The idea about literals is that they are atoms (the simplest formulae) that carry around their intended truth value.

Now we will also review some propositional identities that will be useful later on. Some of them we have already seen, and some are new. All of them can be proven by simple truth table arguments.

Test Calculi: Tableaux and Model Generation

▷ **Idea:** instead of showing $\emptyset \vdash Th$, show $\neg Th \vdash trouble$ (use \perp for trouble)

▷ **Example 11.5.10** Tableau Calculi try to construct models.

Tableau Refutation (Validity)	Model generation (Satisfiability)
$\models P \wedge Q \Rightarrow Q \wedge P$	$\models P \wedge (Q \vee \neg R) \wedge \neg Q$
$P \wedge Q \Rightarrow Q \wedge P^F$	$P \wedge (Q \vee \neg R) \wedge \neg Q^T$
$P \wedge Q^T$	$P \wedge (Q \vee \neg R)^T$
$Q \wedge P^F$	$\neg Q^T$
P^T	Q^F
Q^T	P^T
$P^F \quad Q^F$	$Q \vee \neg R^T$
$\perp \quad \perp$	$Q^T \quad \neg R^T$
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)



Tableau calculi develop a formula in a tree-shaped arrangement that represents a case analysis on when a formula can be made true (or false). Therefore the formulae are decorated with exponents that hold the intended truth value.

On the left we have a refutation tableau that analyzes a negated formula (it is decorated with the intended truth value F). Both branches contain an elementary contradiction \perp .

On the right we have a model generation tableau, which analyzes a positive formula (it is decorated with the intended truth value T). This tableau uses the same rules as the refutation tableau, but makes a case analysis of when this formula can be satisfied. In this case we have a closed branch and an open one, which corresponds a model).

Now that we have seen the examples, we can write down the tableau rules formally.

Analytical Tableaux (Formal Treatment of \mathcal{T}_0)

▷ formula is analyzed in a tree to determine satisfiability

▷ branches correspond to valuations (models)

▷ one per connective

$$\frac{\mathbf{A} \wedge \mathbf{B}^T}{\begin{array}{c} \mathbf{A}^T \\ \mathbf{B}^T \end{array}} \mathcal{T}_0 \wedge \quad \frac{\mathbf{A} \wedge \mathbf{B}^F}{\begin{array}{c} \mathbf{A}^F \\ \mathbf{B}^F \end{array}} \mathcal{T}_0 \vee \quad \frac{\neg \mathbf{A}^T}{\mathbf{A}^F} \mathcal{T}_0^T \quad \frac{\neg \mathbf{A}^F}{\mathbf{A}^T} \mathcal{T}_0^F \quad \frac{\begin{array}{c} \mathbf{A}^\alpha \\ \mathbf{A}^\beta \end{array} \quad \alpha \neq \beta}{\perp} \mathcal{T}_0 \text{cut}$$

▷ Use rules exhaustively as long as they contribute new material

- ▷ **Definition 11.5.11** Call a tableau **saturated**, iff no rule applies, and a branch **closed**, iff it ends in \perp , else **open**. (open branches in saturated tableaux yield models)
- ▷ **Definition 11.5.12 (\mathcal{T}_0 -Theorem/Derivability)** \mathbf{A} is a \mathcal{T}_0 -theorem ($\vdash_{\mathcal{T}_0} \mathbf{A}$), iff there is a closed tableau with \mathbf{A}^F at the root.
- $\Phi \subseteq wff_o(\mathcal{V}_o)$ **derives** \mathbf{A} in \mathcal{T}_0 ($\Phi \vdash_{\mathcal{T}_0} \mathbf{A}$), iff there is a closed tableau starting with \mathbf{A}^F and Φ^T .



These inference rules act on tableaux have to be read as follows: if the formulae over the line appear in a tableau branch, then the branch can be extended by the formulae or branches below the line. There are two rules for each primary connective, and a branch closing rule that adds the special symbol \perp (for unsatisfiability) to a branch.

We use the tableau rules with the convention that they are only applied, if they contribute new material to the branch. This ensures termination of the tableau procedure for propositional logic (every rule eliminates one primary connective).

Definition 11.5.13 We will call a closed tableau with the signed formula \mathbf{A}^α at the root a **tableau refutation** for \mathcal{A}^α .

The saturated tableau represents a full case analysis of what is necessary to give \mathbf{A} the truth value α ; since all branches are closed (contain contradictions) this is impossible.

Definition 11.5.14 We will call a tableau refutation for \mathbf{A}^F a **tableau proof** for \mathbf{A} , since it refutes the possibility of finding a model where \mathbf{A} evaluates to F . Thus \mathbf{A} must evaluate to T in all models, which is just our definition of validity.

Thus the tableau procedure can be used as a calculus for propositional logic. In contrast to the calculus in **?sec.hilbert?** it does not prove a theorem \mathbf{A} by deriving it from a set of axioms, but it proves it by refuting its negation. Such calculi are called negative or test calculi. Generally negative calculi have computational advantages over positive ones, since they have a built-in sense of direction.

We have rules for all the necessary connectives (we restrict ourselves to \wedge and \neg , since the others can be expressed in terms of these two via the propositional identities above. For instance, we can write $\mathbf{A} \vee \mathbf{B}$ as $\neg(\neg \mathbf{A} \wedge \neg \mathbf{B})$, and $\mathbf{A} \Rightarrow \mathbf{B}$ as $\neg \mathbf{A} \vee \mathbf{B}, \dots$)

We will now look at an example. Following our introduction of propositional logic **?impsem-ex?** we look at a formulation of propositional logic with fancy variable names. Note that `love(mary, bill)` is just a variable name like `P` or `X`, which we have used earlier.

A Valid Real-World Example

- ▷ **Example 11.5.15** If Mary loves Bill and John loves Mary, then John loves

Mary

$$\begin{aligned}
 & \text{love(mary, bill)} \wedge \text{love(john, mary)} \Rightarrow \text{love(john, mary)}^F \\
 \neg & (\neg \neg (\text{love(mary, bill)} \wedge \text{love(john, mary)}) \wedge \neg \text{love(john, mary)})^F \\
 \neg & \neg (\text{love(mary, bill)} \wedge \text{love(john, mary)}) \wedge \neg \text{love(john, mary)}^T \\
 & \neg \neg (\text{love(mary, bill)} \wedge \text{love(john, mary)})^T \\
 & \neg (\text{love(mary, bill)} \wedge \text{love(john, mary)})^F \\
 & \text{love(mary, bill)} \wedge \text{love(john, mary)}^T \\
 & \neg \text{love(john, mary)}^T \\
 & \text{love(mary, bill)}^T \\
 & \text{love(john, mary)}^T \\
 & \text{love(john, mary)}^F \\
 & \perp
 \end{aligned}$$

This is a closed tableau, so the $\text{love(mary, bill)} \wedge \text{love(john, mary)} \Rightarrow \text{love(john, mary)}$ is a \mathcal{T}_0 -theorem.

As we will see, \mathcal{T}_0 is sound and complete, so

$$\text{love(mary, bill)} \wedge \text{love(john, mary)} \Rightarrow \text{love(john, mary)}$$

is valid.



©: Michael Kohlhase

308



We could have used the entailment theorem (?entl-thm-cor?) here to show that *If Mary loves Bill and John loves Mary entails John loves Mary*. But there is a better way to show entailment: we directly use derivability in \mathcal{T}_0

Deriving Entailment in \mathcal{T}_0

▷ **Example 11.5.16** Mary loves Bill and John loves Mary together entail that John loves Mary

$$\begin{aligned}
 & \text{love(mary, bill)}^T \\
 & \text{love(john, mary)}^T \\
 & \text{love(john, mary)}^F \\
 & \perp
 \end{aligned}$$

This is a closed tableau, so the $\{\text{love(mary, bill)}, \text{love(john, mary)}\} \vdash_{\mathcal{T}_0} \text{love(john, mary)}$, again, as \mathcal{T}_0 is sound and complete we have

$$\{\text{love(mary, bill)}, \text{love(john, mary)}\} \models \text{love(john, mary)}$$



©: Michael Kohlhase

309



Note: that we can also use the tableau calculus to try and show entailment (and fail). The nice thing is that the failed proof, we can see what went wrong.

A Falsifiable Real-World Example

▷ **Example 11.5.17** *If Mary loves Bill or John loves Mary, then John loves Mary

Try proving the implication

(this fails)

$$\begin{array}{c}
 (\text{love(mary, bill)} \vee \text{love(john, mary)}) \Rightarrow \text{love(john, mary)}^F \\
 \neg(\neg(\neg(\text{love(mary, bill)} \vee \text{love(john, mary)})) \wedge \neg\text{love(john, mary)})^F \\
 \neg(\neg(\text{love(mary, bill)} \vee \text{love(john, mary)})) \wedge \neg\text{love(john, mary)}^T \\
 \quad \neg\text{love(john, mary)}^T \\
 \quad \text{love(john, mary)}^F \\
 \neg(\neg(\text{love(mary, bill)} \vee \text{love(john, mary)}))^T \\
 \neg(\text{love(mary, bill)} \vee \text{love(john, mary)})^F \\
 \quad \text{love(mary, bill)}^T \vee \text{love(john, mary)}^T \\
 \quad \text{love(mary, bill)}^T \mid \text{love(john, mary)}^T \\
 \quad \perp
 \end{array}$$

Indeed we can make $\mathcal{I}_\varphi(\text{love(mary, bill)}) = T$ but $\mathcal{I}_\varphi(\text{love(john, mary)}) = F$.



Obviously, the tableau above is saturated, but not closed, so it is not a tableau proof for our initial entailment conjecture. We have marked the literal on the open branch green, since they allow us to read off the conditions of the situation, in which the entailment fails to hold. As we intuitively argued above, this is the situation, where Mary loves Bill. In particular, the open branch gives us a variable assignment (marked in green) that satisfies the initial formula. In this case, *Mary loves Bill*, which is a situation, where the entailment fails.

Again, the derivability version is much simpler

Testing for Entailment in \mathcal{T}_0

- ▷ **Example 11.5.18** Does *Mary loves Bill or John loves Mary* entail that *John loves Mary*?

$$\begin{array}{c}
 \text{love(mary, bill)} \vee \text{love(john, mary)}^T \\
 \quad \text{love(john, mary)}^F \\
 \text{love(mary, bill)}^T \mid \text{love(john, mary)}^T \\
 \quad \perp
 \end{array}$$

This saturated tableau has an open branch that shows that the interpretation with $\mathcal{I}_\varphi(\text{love(mary, bill)}) = T$ but $\mathcal{I}_\varphi(\text{love(john, mary)}) = F$ falsifies the derivability/entailment conjecture.



Practical Enhancements for Tableaux

Propositional Identities

- ▷ **Definition 11.5.19** 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
Idempotence	$\varphi \wedge \varphi = \varphi$	$\varphi \vee \varphi = \varphi$
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	$\varphi \Rightarrow \psi = \neg\varphi \vee \psi$	$\varphi \Leftrightarrow \psi = (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$



We have seen in the examples above that while it is possible to get by with only the connectives \vee and \neg , it is a bit unnatural and tedious, since we need to eliminate the other connectives first. In this section, we will make the calculus less frugal by adding rules for the other connectives, without losing the advantage of dealing with a small calculus, which is good making statements about the calculus.

The main idea is to add the new rules as derived rules, i.e. inference rules that only abbreviate deductions in the original calculus. Generally, adding derived inference rules does not change the derivability relation of the calculus, and is therefore a safe thing to do. In particular, we will add the following rules to our tableau system.

We will convince ourselves that the first rule is a derived rule, and leave the other ones as an exercise.

Derived Rules of Inference

▷ **Definition 11.5.20** Let \mathcal{C} be a calculus, a rule of inference $\frac{\mathbf{A}_1 \cdots \mathbf{A}_n}{\mathbf{C}}$ is called a **derived inference rule** in \mathcal{C} , iff there is a \mathcal{C} -proof of $\mathbf{A}_1, \dots, \mathbf{A}_n \vdash \mathbf{C}$.

▷ **Definition 11.5.21** We have the following derived rules of inference

$$\begin{array}{cccc}
 \frac{\mathbf{A} \Rightarrow \mathbf{B}^T}{\mathbf{A}^F \mid \mathbf{B}^T} & \frac{\mathbf{A} \Rightarrow \mathbf{B}^F}{\mathbf{A}^T \mid \mathbf{B}^F} & \frac{\mathbf{A} \Rightarrow \mathbf{B}^T}{\mathbf{B}^T} & \frac{\mathbf{A}^T}{\neg \mathbf{A} \vee \mathbf{B}^T} \\
 & & & \frac{\mathbf{A}^T}{\neg(\neg \mathbf{A} \wedge \neg \mathbf{B})^T} \\
 & & & \frac{\mathbf{A}^T}{\neg \neg \mathbf{A} \wedge \neg \mathbf{B}^F} \\
 \frac{\mathbf{A} \vee \mathbf{B}^T}{\mathbf{A}^T \mid \mathbf{B}^T} & \frac{\mathbf{A} \vee \mathbf{B}^F}{\mathbf{A}^F \mid \mathbf{B}^F} & \frac{\mathbf{A} \Leftrightarrow \mathbf{B}^T}{\mathbf{A}^T \mid \mathbf{A}^F} & \frac{\mathbf{A} \Leftrightarrow \mathbf{B}^F}{\mathbf{A}^T \mid \mathbf{A}^F} \\
 & & & \frac{\mathbf{A} \Leftrightarrow \mathbf{B}^F}{\neg \neg \mathbf{A}^F \mid \neg \mathbf{B}^F} \\
 & & & \frac{\mathbf{A}^T}{\neg \mathbf{A}^T \mid \mathbf{B}^F} \\
 & & & \frac{\mathbf{A}^T}{\mathbf{A}^F \mid \mathbf{B}^F} \\
 & & & \frac{\mathbf{A}^T}{\perp}
 \end{array}$$



With these derived rules, theorem proving becomes quite efficient. With these rules, the tableau (?tab:firsttab?) would have the following simpler form:

Tableaux with derived Rules (example)

Example 11.5.22

$$\begin{array}{c}
 \text{love(mary, bill)} \wedge \text{love(john, mary)} \Rightarrow \text{love(john, mary)}^F \\
 \text{love(mary, bill)} \wedge \text{love(john, mary)}^T \\
 \quad \text{love(john, mary)}^F \\
 \quad \text{love(mary, bill)}^T \\
 \quad \text{love(john, mary)}^T \\
 \quad \perp
 \end{array}$$



Another thing that was awkward in (?tab:firsttab?) was that we used a proof for an implication to prove logical consequence. Such tests are necessary for instance, if we want to check consistency or informativity of new sentences³. Consider for instance a discourse $\Delta = D^1, \dots, D^n$, where n EdN:3 is large. To test whether a hypothesis H is a consequence of Δ ($\Delta \models H$) we need to show that $C := (D^1 \wedge \dots) \wedge D^n \Rightarrow H$ is valid, which is quite tedious, since C is a rather large formula, e.g. if Δ is a 300 page novel. Moreover, if we want to test entailment of the form ($\Delta \models H$) often, – for instance to test the informativity and consistency of every new sentence H , then successive Δ s will overlap quite significantly, and we will be doing the same inferences all over again; the entailment check is not incremental.

Fortunately, it is very simple to get an incremental procedure for entailment checking in the model-generation-based setting: To test whether $\Delta \models H$, where we have interpreted Δ in a model generation tableau T , just check whether the tableau closes, if we add $\neg H$ to the open branches. Indeed, if the tableau closes, then $\Delta \wedge \neg H$ is unsatisfiable, so $\neg(\Delta \wedge \neg H)$ is valid, but this is equivalent to $\Delta \Rightarrow H$, which is what we wanted to show.

Example 11.5.23 Consider for instance the following entailment in natural language.

Mary loves Bill. John loves Mary \models John loves Mary

⁴ We obtain the tableau

$$\begin{array}{c}
 \text{love(mary, bill)}^T \\
 \text{love(john, mary)}^T \\
 \neg(\text{love(john, mary)})^T \\
 \text{love(john, mary)}^F \\
 \perp
 \end{array}$$

EdN:4

which shows us that the conjectured entailment relation really holds.

Soundness and Termination of Tableaux

As always we need to convince ourselves that the calculus is sound, otherwise, tableau proofs do not guarantee validity, which we are after. Since we are now in a refutation setting we cannot just show that the inference rules preserve validity: we care about unsatisfiability (which is the dual notion to validity), as we want to show the initial labeled formula to be unsatisfiable. Before we can do this, we have to ask ourselves, what it means to be (un)-satisfiable for a labeled formula or a tableau.

Soundness (Tableau)

- ▷ **Idea:** A test calculus is sound, iff it preserves satisfiability and the goal formulae are unsatisfiable.

³EDNOTE: add reference to presupposition stuff

⁴EDNOTE: need to mark up the embedding of NL strings into Math

- ▷ **Definition 11.5.24** A labeled formula \mathbf{A}^α is valid under φ , iff $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$.
- ▷ **Definition 11.5.25** 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 11.5.26** *Tableau rules transform satisfiable tableaux into satisfiable ones.*
- ▷ **Theorem 11.5.27 (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.

P.1 then the initial tableau is satisfiable (Φ^F satisfiable)

P.2 so \mathcal{T} is satisfiable, by Lemma 11.5.26.

P.3 there is a satisfiable branch (by definition)

P.4 but all branches are closed (\mathcal{T} closed)

□



Thus we only have to prove Lemma 11.5.26, this is relatively easy to do. For instance for the first rule: if we have a tableau that contains $\mathbf{A} \wedge \mathbf{B}^T$ and is satisfiable, then it must have a satisfiable branch. If $\mathbf{A} \wedge \mathbf{B}^T$ is not on this branch, the tableau extension will not change satisfiability, so we can assume that it is on the satisfiable branch and thus $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = T$ for some variable assignment φ . Thus $\mathcal{I}_\varphi(\mathbf{A}) = T$ and $\mathcal{I}_\varphi(\mathbf{B}) = T$, so after the extension (which adds the formulae \mathbf{A}^T and \mathbf{B}^T to the branch), the branch is still satisfiable. The cases for the other rules are similar.

The next result is a very important one, it shows that there is a procedure (the tableau procedure) that will always terminate and answer the question whether a given propositional formula is valid or not. This is very important, since other logics (like the often-studied first-order logic) does not enjoy this property.

Termination for Tableaux

- ▷ **Lemma 11.5.28** *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.*
- ▷ Let us call a labeled formulae \mathbf{A}^α **worked off** in a tableau \mathcal{T} , if a tableau rule has already been applied to it.
- ▷ **Proof:**
 - P.1** It is easy to see that applying rules to worked off formulae will only add formulae that are already present in its branch.
 - P.2** Let $\mu(\mathcal{T})$ be the number of connectives in labeled formulae in \mathcal{T} that are not worked off.
 - P.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)
 - P.4** At some point the tableau only contains worked off formulae and literals.
 - P.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. □



The Tableau calculus basically computes the disjunctive normal form: every branch is a disjunct that is a conjunct of literals. The method relies on the fact that a DNF is unsatisfiable, iff each monomial is, i.e. iff each branch contains a contradiction in form of a pair of complementary literals.

11.5.2 Resolution for Propositional Logic

The next calculus is a test calculus based on the conjunctive normal form: the [resolution calculus](#). In contrast to the tableau method, it does not compute the normal form as it goes along, but has a pre-processing step that does this and a single inference rule that maintains the normal form. The goal of this calculus is to derive the [empty clause](#) (the empty disjunction), which is unsatisfiable.

Another Test Calculus: Resolution

▷ **Definition 11.5.29 (Resolution Calculus)** The [resolution calculus](#) operates a 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 11.5.30 (Resolution Refutation)** Let S be a [clause set](#), then we call a \mathcal{R} derivation $\mathcal{D}: S \vdash_{\mathcal{R}} \square$ [resolution refutation](#).

▷ **Definition 11.5.31 (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 11.5.32** A [clause](#) is a disjunction of literals. We will use \square for the empty disjunction (no disjuncts) and call it the [empty clause](#).

▷ **Definition 11.5.33** 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 11.5.34 (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 11.5.35** We write $CNF^0(A^\alpha)$ for the set of all clauses derivable from A^α via the rules above.



Note that the **C**-terms in the definition of the resolution calculus are necessary, since we assumed that the assumptions of the inference rule must match full formulae. The **C**-terms are used with the convention that they are optional. So that we can also simplify $(\mathbf{A} \vee \mathbf{B})^T$ to $\mathbf{A}^T \vee \mathbf{B}^T$.

Background: The background behind this notation is that \mathbf{A} and $T \vee \mathbf{A}$ are equivalent for any \mathbf{A} . That allows us to interpret the **C**-terms in the assumptions as T and thus leave them out.

The clause normal form translation as we have formulated it here is quite frugal; we have left out rules for the connectives \vee , \Rightarrow , and \Leftrightarrow , relying on the fact that formulae containing these connectives can be translated into ones without before CNF transformation. The advantage of having a calculus with few inference rules is that we can prove meta-properties like soundness and completeness with less effort (these proofs usually require one case per inference rule). On the other hand, adding specialized inference rules makes proofs shorter and more readable.

Fortunately, there is a way to have your cake and eat it. Derived inference rules have the property that they are formally redundant, since they do not change the expressive power of the calculus. Therefore we can leave them out when proving meta-properties, but include them when actually using the calculus.

Derived Rules of Inference

▷ **Definition 11.5.36** Let \mathcal{C} be a calculus, a rule of inference $\frac{\mathbf{A}_1 \quad \dots \quad \mathbf{A}_n}{\mathbf{C}}$ is

called a **derived inference rule** in \mathcal{C} , iff there is a \mathcal{C} -proof of $\mathbf{A}_1, \dots, \mathbf{A}_n \vdash \mathbf{C}$.

$$\begin{array}{c} \frac{\mathbf{C} \vee (\mathbf{A} \Rightarrow \mathbf{B})^T}{\mathbf{C} \vee (\neg \mathbf{A} \vee \mathbf{B})^T} \\ \frac{\mathbf{C} \vee \neg \mathbf{A}^T \vee \mathbf{B}^T}{\mathbf{C} \vee \mathbf{A}^F \vee \mathbf{B}^T} \end{array} \rightsquigarrow \frac{\mathbf{C} \vee (\mathbf{A} \Rightarrow \mathbf{B})^T}{\mathbf{C} \vee \mathbf{A}^F \vee \mathbf{B}^T}$$

▷ **Others:**

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



With these derived rules, theorem proving becomes quite efficient. To get a better understanding of the calculus, we look at an example: we prove an axiom of the Hilbert Calculus we have studied above.

Example: Proving Axiom S

▷ **Example 11.5.38** Clause Normal Form transformation

$$\begin{array}{c}
 \dfrac{(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R^F}{P \Rightarrow Q \Rightarrow R^T; (P \Rightarrow Q) \Rightarrow P \Rightarrow R^F} \\
 \hline
 \dfrac{P^F \vee (Q \Rightarrow R)^T; P \Rightarrow Q^T; P \Rightarrow R^F}{P^F \vee Q^F \vee R^T; P^F \vee Q^T; P^T; R^F}
 \end{array}$$

$$CNF = \{P^F \vee Q^F \vee R^T, P^F \vee Q^T, P^T, R^F\}$$

▷ **Example 11.5.39** Resolution Proof

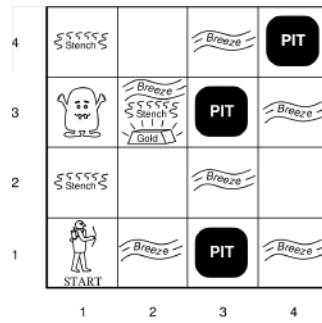
1	$P^F \vee Q^F \vee R^T$	initial
2	$P^F \vee Q^T$	initial
3	P^T	initial
4	R^F	initial
5	$P^F \vee Q^F$	resolve 1.3 with 4.1
6	Q^F	resolve 5.1 with 3.1
7	P^F	resolve 2.2 with 6.1
8	□	resolve 7.1 with 3.1



Excuse: A full analysis of any calculus needs a completeness proof. We will not cover this in the course, but provide one for the calculi introduced so far in Chapter A.

11.6 Killing a Wumpus with Propositional Inference

Where is the Wumpus? The Situation



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



Where is the Wumpus? Our Knowledge

- ▷ 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, amongst many others:

$$\begin{aligned} R_1 &:= \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1} \\ R_2 &:= \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1} \\ R_3 &:= \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3} \\ R_4 &:= S_{1,2} \Rightarrow W_{1,3} \vee W_{2,2} \vee W_{1,1} \end{aligned}$$
- ▷ **To show:** $R_1, R_2, R_3, R_4 \models W_{1,3}$



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!

- ▷ **Derivation proving that the Wumpus 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 .



11.7 Conclusion

Summary

- ▷ 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**.
- ▷ **Tableaux** and **Resolution** 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 $KB \models A$ by showing that $KB \cup \{\neg A\}$ 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 ...**
- ▷ 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 \sim further expansion of the rules.
- ▷ **Can we design a more human-like logic?** Yep:

- ▷ **Predicate logic:** Quantification of variables ranging over objects. (cf. Chapter 13 **and** Chapter 14)
- ▷ ... 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.



Reading:

- *Chapter 7: Logical Agents*, Sections 7.1 – 7.5 [RN09].
- **Content:** Sections 7.1 and 7.2 roughly correspond to my “Introduction”, Section 7.3 roughly corresponds to my “Logic (in AI)”, Section 7.4 roughly corresponds to my “Propositional Logic”, Section 7.5 roughly corresponds to my “Resolution” and “Killing a Wumpus”.
- Overall, the content is quite similar. I have tried to add some additional clarifying illustrations. RN gives many complementary explanations, nice as additional background reading.
- I would note that RN’s presentation of resolution seems a bit awkward, and Section 7.5 contains some additional material that is imho not interesting (alternate inference rules, forward and backward chaining). Horn clauses and unit resolution (also in Section 7.5), on the other hand, are quite relevant.

Chapter 12

Propositional Reasoning: SAT Solvers

12.1 Introduction

Reminder: Our Agenda for Propositional Logic

- ▷ Chapter 11: Basic definitions and concepts; machine-oriented calculi
- ▷ Sets up the framework. Resolution is the quintessential reasoning procedure underlying most successful solvers.
- ▷ Here: 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.



©: Michael Kohlhase

327



SAT

- ▷ **Definition 12.1.1** The **SAT Problem**: Given a propositional formula **A**, decide whether or not **A** is satisfiable. We denote the class of all SAT problems with **SAT**
- ▷ 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 in polynomial time be transformed into a satisfiability-equivalent CNF formula (cf. Chapter 11).
- ▷ Active research area, annual SAT conference, lots of tools etc. available: <http://www.satlive.org/>
- ▷ **Definition 12.1.2** Tools addressing SAT are commonly referred to as **SAT solvers**.

▷ **Recall:** To decide whether $\text{KB} \models \mathbf{A}$, decide satisfiability of $\theta := \text{KB} \cup \{\neg \mathbf{A}\}$: θ is unsatisfiable iff $\text{KB} \models \mathbf{A}$.

▷ **Consequence:** Deduction can be performed using SAT solvers.



SAT vs. CSP

▷ **Recall:** Constraint network $\gamma = (V, D, C)$ 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 12.1.3 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 $\mathbf{A}(\gamma)$ that is satisfiable iff γ is solvable.

▷ How to construct $\mathbf{A}(\gamma)$? Exercises.

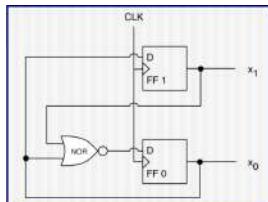
▷ **Upshot:** Anything we can do with CSP, we can (in principle) do with SAT.



Example Application: Hardware Verification

▷ **Example 12.1.4 (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 x'_1, x'_0 (value in next cycle).

▷ **Transition relation:** $x'_1 \Leftrightarrow x_0; x'_0 \Leftrightarrow (\neg(x_1 \vee x_0))$.

▷ **Initial state:** $\neg(x_1 \wedge x_0)$.

▷ **Error property:** $x_1 \wedge x'_0$.

▷ **Step 2:** Transform to CNF, encode as set Δ of clauses.

▷ **Clauses:** $x'_1 \vee x_0^T$, $x'_1 \vee x_0^F$, $x'_0 \vee x_1^T \vee x_0^T$, $x'_0 \vee x_1^F$, $x'_0 \vee x_0^F$, $x_1^F \vee x_0^F$,
 x'_1 , x_0^T .

▷ **Step 3:** Call a SAT solver (up next).



©: Michael Kohlhase

330



Our Agenda for This Chapter

- ▷ **The Davis-Putnam (Logemann-Loveland) Procedure:** How to systematically test satisfiability?
 - ▷ The quintessential SAT solving procedure, DPLL.
- ▷ **DPLL = (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.



©: Michael Kohlhase

331



But – What About Local Search for SAT?

- ▷ **There’s a wealth of research on local search for SAT, e.g.:**
- ▷ **Definition 12.1.5** The **GSAT algorithm** **OUTPUT:** a satisfying truth assignment of Δ , if found

```
function GSAT ( $\Delta$ , Max-Flips, Max-Tries)
for i:=1 to Max-Tries
  l:= a randomly-generated truth assignment
  for j:=1 to Max-Flips
    if I satisfies  $\Delta$  then return I
    X:= a proposition reversing whose truth assignment gives
        the largest increase in the number of satisfied clauses
    l:= 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 Section 7.5 (Not covered here)



12.2 The Davis-Putnam (Logemann-Loveland) Procedure

The DPLL Procedure

- ▷ **Definition 12.2.1** The **DPLL** Procedure is called on a clause set Δ and the empty partial interpretation φ :

```
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  $C = l$  do
    extend  $I'$  with the respective truth value for the proposition underlying  $l$ 
    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  $I''' :=$  DPLL( $\Delta'', I''$ )  $\neq$  "unsatisfiable" then return  $I'''$ 
   $I'' := I'$  extended with the other truth value for  $P$ ;  $\Delta'' := \Delta'$ ; simplify  $\Delta''$ 
  return DPLL( $\Delta'', I''$ )
```

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



DPLL: Example (Vanilla1)

- ▷ **Example 12.2.2** 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. $P \mapsto F$
 $Q^T; Q^F$

3a. UP Rule: $Q \mapsto T$

2b. $\overline{P} \mapsto T$
 Q^F

3b. UP Rule: $Q \mapsto F$



DPLL: Example (Vanilla2)

▷ **Example 12.2.3** 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: $P \mapsto T$
□

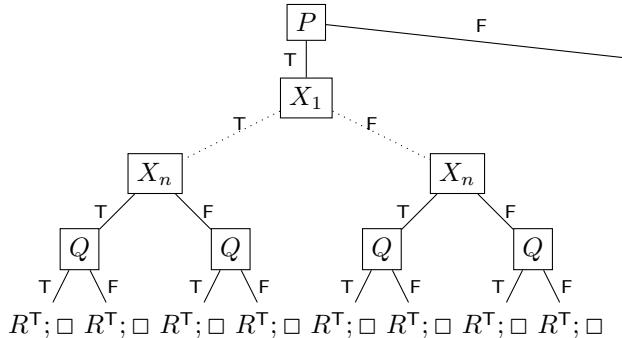


DPLL: Example (Redundance1)

▷ **Example 12.2.4** 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 Δ ; Θ with $\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$



Properties of DPLL

▷ **Unsatisfiable case:**

- ▷ What can we say if “unsatisfiable” is returned?
- ▷ In this case, we know that Δ is unsatisfiable: Unit propagation is *sound*, in the sense that it does not reduce the set of solutions.

▷ **Satisfiable case:**

- ▷ What can we say when a partial interpretation I is returned?

- ▷ Any extension of I to a complete interpretation satisfies Δ . (By construction, I suffices to satisfy all clauses.)

▷ **Déjà Vu, Anybody?**

- ▷ DPLL $\hat{=}$ BacktrackingWithInference, with 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.



UP = Unit Resolution

- ▷ The Unit Propagation (UP) Rule ...

```
while  $\Delta'$  contains a unit clause  $\{l\}$  do
    extend  $I'$  with the respective truth value for the proposition underlying  $l$ 
    simplify  $\Delta'$  /* remove false literals */
```

... corresponds to a calculus:

- ▷ **Definition 12.2.5 (Unit Resolution)** Unit resolution (UR) is the calculus consisting of the following inference rule:

$$\frac{C \vee P^F \quad P^T}{C}$$

- ▷ Unit propagation = Resolution restricted to the case where one of the parent clauses is unit.
- ▷ **Observation 12.2.6 (Soundness)** UR is sound (since resolution is)
- ▷ **Observation 12.2.7 (Completeness)** UR is not refutation complete
- ▷ **Example 12.2.8** $P^T \vee Q^T; P^T \vee Q^F; P^F \vee Q^T; P^F \vee Q^F$ is unsatisfiable but UP cannot derive the empty clause \square .
- ▷ UP makes only limited inferences, as long as there are unit clauses. It does not guarantee to infer everything that can be inferred.



12.3 DPLL = (A Restricted Form of) Resolution

DPLL vs. Resolution

- ▷ **Definition 12.3.1** 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 the splitting rule.
- ▷ **Theorem 12.3.2** If DPLL returns “unsatisfiable” on Δ , then $\Delta \vdash_{\mathcal{R}} \square$ with a

resolution derivation whose length is at most the number of decisions.

▷ **Proof:** Consider first DPLL without UP

P.1 Consider any leaf node N , for proposition X , both of whose truth values directly result in a clause C that has become empty.

P.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 .

P.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.

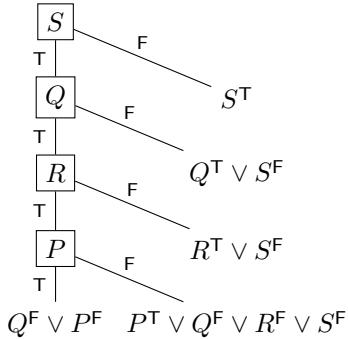
P.4 Unit propagation can be simulated via applications of the splitting rule, choosing a proposition that is constrained by a unit clause: One of the two truth values then immediately yields an empty clause. □



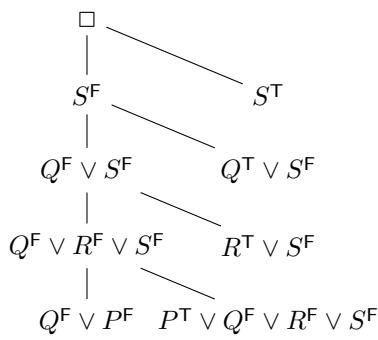
DPLL vs. Resolution: Example (Vanilla2)

▷ **Example 12.3.3** $\Delta := Q^F \vee P^F; P^T \vee Q^F \vee R^F \vee S^F; Q^T \vee S^F; R^T \vee S^F; S^T$

DPLL: (Without UP; leaves annotated with clauses that became empty)



Resolution Proof from that DPLL Tree:



For reference, we give the full proof here.

Theorem 12.3.4 If DPLL returns “unsatisfiable” on Δ , then $\mathcal{R}: S \not\vdash_{\mathcal{R}} \square$ with a resolution derivation whose length is at most the number of decisions.

Proof: Consider first DPLL with no unit propagation.

P.1 If the search tree is not empty, then there exists a leaf node N , i.e., a node associated to proposition X so that, for each value of X , the partial assignment directly results in an empty clause.

P.2 Denote the parent decisions of N by l_1, \dots, l_k , where l_i is a literal for proposition X_i and the search node containing X_i is N_i .

P.3 Denote the empty clause for X by $C(N, X)$, and denote the empty clause for X^F by $C(N, X^F)$.

P.4 For each $x \in \{X^T, X^F\}$ we have the following properties:

1. $x^F \in C(N, x)$; and
2. $C(N, x) \subseteq \{x^F, \overline{l_1}, \dots, \overline{l_k}\}$.

Due to , we can resolve $C(N, X)$ with $C(N, X^F)$; denote the outcome clause by $C(N)$.

P.5 We obviously have that (1) $C(N) \subseteq \{\overline{l_1}, \dots, \overline{l_k}\}$.

P.6 The proof now proceeds by removing N from the search tree and attaching $C(N)$ at the l_k branch of N_k , in the role of $C(N_k, l_k)$ as above. Then we select the next leaf node N' and iterate the argument; once the tree is empty, by (1) we have derived the empty clause. What we need to show is that, in each step of this iteration, we preserve the properties (a) and (b) for all leaf nodes. Since we did not change anything in other parts of the tree, the only node we need to show this for is $N' := N_k$.

P.7 Due to (1), we have (b) for N_k . But we do not necessarily have (a): $C(N) \subseteq \{\overline{l_1}, \dots, \overline{l_k}\}$, but there are cases where $\overline{l_k} \notin C(N)$ (e.g., if X_k is not contained in any clause and thus branching over it was completely unnecessary). If so, however, we can simply remove N_k and all its descendants from the tree as well. We attach $C(N)$ at the l_{k-1} branch of N_{k-1} , in the role of $C(N_{k-1}, l_{k-1})$. If $\overline{l_{k-1}} \in C(N)$ then we have (a) for $N' := N_{k-1}$ and can stop. If $\overline{l_{k-1}} \notin C(N)$, then we remove N_{k-1} and so forth, until either we stop with (a), or have removed N_1 and thus must already have derived the empty clause (because $C(N) \subseteq \{\overline{l_1}, \dots, \overline{l_k}\} \setminus \{\overline{l_1}, \dots, \overline{l_k}\}$).

P.8 Unit propagation can be simulated via applications of the splitting rule, choosing a proposition that is constrained by a unit clause: One of the two truth values then immediately yields an empty clause. \square

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 = tree resolution.

- ▷ This is a fundamental weakness! There are inputs Δ whose shortest tree-resolution proof is exponentially longer than their shortest (general) resolution proof.
- ▷ In a tree resolution, each derived clause C is used only once (at its parent).
The same C is derived anew every time it is used!

▷ DPLL “makes the same mistakes over and over again”.

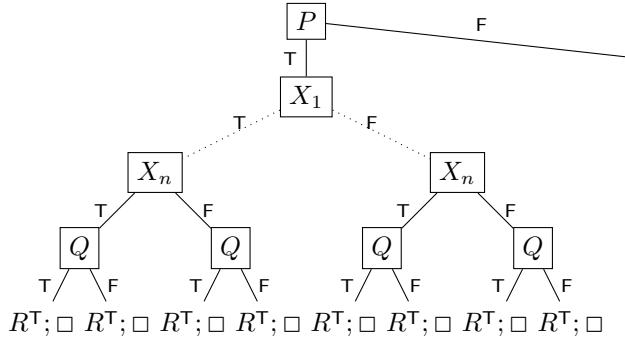
▷ To the rescue: **clause learning**.



12.4 Why Did Unit Propagation Yield a Conflict?

DPLL: Example (Redundance1)

▷ **Example 12.4.1** We introduce some nasty redundancy to make DPLL slow.
 $\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^T \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$
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 12.4.2** 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 by the UP rule to α .

▷ **Definition 12.4.3 (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 .

Where $(l_1 \vee \dots \vee l_k \vee l') \in \Delta$ 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) .

- ▷ How do we know that \bar{l}_i are vertices in G_β^{impl} ?
- ▷ Because $l_1 \vee \dots \vee l_k \vee l'$ became unit/empty.
- ▷ **Observation 12.4.4** G_β^{impl} is acyclic
- ▷ **Proof Sketch:** UP can't derive l' whose value was already set beforehand. \square
- ▷ **Vertices with in-degree 0:** Choice literals, and unit clauses of Δ .



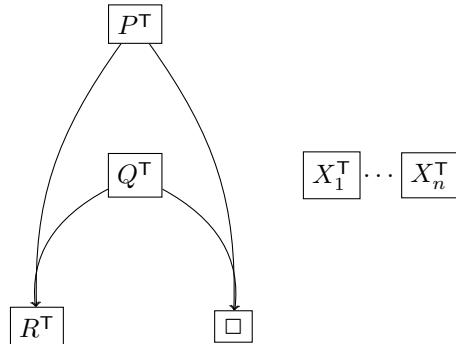
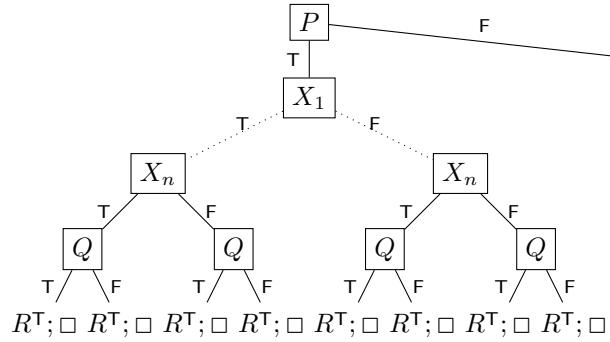
Implication Graphs: Example (Vanilla1) in Detail

- ▷ **Example 12.4.5** Let $\Delta := P^\top \vee Q^\top \vee R^\text{F}; P^\text{F} \vee Q^\text{F}; R^\top; P^\top \vee Q^\text{F}$
 1. UP Rule: $R \mapsto \top$
Implied literal R^\top .
 $P^\top \vee Q^\top; P^\text{F} \vee Q^\text{F}; P^\top \vee Q^\text{F}$
 2. Splitting Rule:
 - 2a. $P \mapsto \text{F}$
Choice literal P^F .
 $Q^\top; Q^\text{F}$
 - 3a. UP Rule: $Q \mapsto \top$
Implied literal Q^\top , edges (R^\top, Q^\top) and (P^F, Q^\top) .
 \square
Conflict vertex $\square_{P^\top \vee Q^\text{F}}$, edges $(P^\text{F}, \square_{P^\top \vee Q^\text{F}})$ and $(Q^\top, \square_{P^\top \vee Q^\text{F}})$.



Implication Graphs: Example (Redundance1)

- ▷ **Example 12.4.6** Continuing from Example 12.4.5:
 $\Delta := P^\text{F} \vee Q^\text{F} \vee R^\top; P^\text{F} \vee Q^\text{F} \vee R^\text{F}; P^\text{F} \vee Q^\top \vee R^\text{T}; P^\text{F} \vee Q^\text{T} \vee R^\text{F}$
DPLL on Δ ; Θ with $\Theta := X_1^\top \vee \dots \vee X_{100}^\top; X_1^\text{F} \vee \dots \vee X_{100}^\text{F}$
Choice: $P^\top, X_1^\top, \dots, X_{100}^\top, Q^\top$. Implied: R^\top .

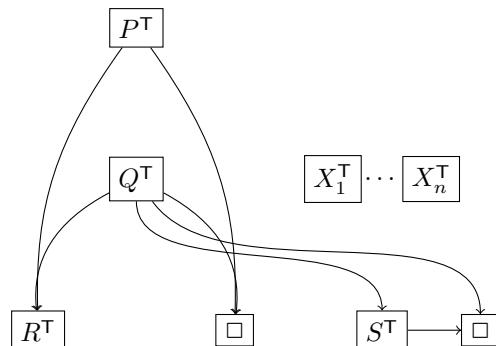


Implication Graphs: Example (Redundance2)

▷ **Example 12.4.7** Continuing from Example 12.4.6:

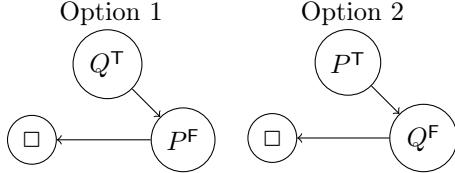
$$\begin{aligned}\Delta &:= P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F \\ \Theta &:= X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F\end{aligned}$$

DPLL on $\Delta; \Theta; \Phi$ with $\Phi := Q^F \vee S^T; Q^F \vee S^F$
 Choice: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied: R^T .



Implication Graphs: A Remark

- ▷ The implication graph is *not* uniquely determined by the choice literals.
- ▷ It depends on “ordering decisions” made during UP: Which unit clause is picked first.
- ▷ **Example 12.4.8** $\Delta = P^F \vee Q^T; Q^T; P^T$



©: Michael Kohlhase

348



Conflict Graphs

- ▷ A conflict graph captures “what went wrong” in a failed node.
- ▷ **Definition 12.4.9 (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 l' is a vertex in C , then all parents of l' , i.e. vertices \bar{l}_i with a I edge (\bar{l}_i, l') , 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.



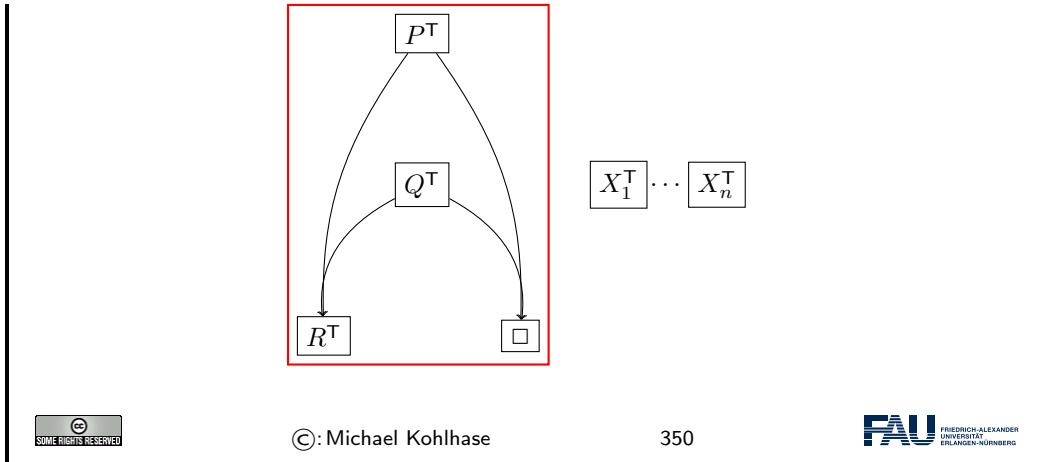
©: Michael Kohlhase

349



Conflict-Graphs: Example (Redundance1)

- ▷ **Example 12.4.10** Continuing from Example 12.4.6:
 $\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$
DPLL on Δ ; Θ with $\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$
Choice: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied: R^T .

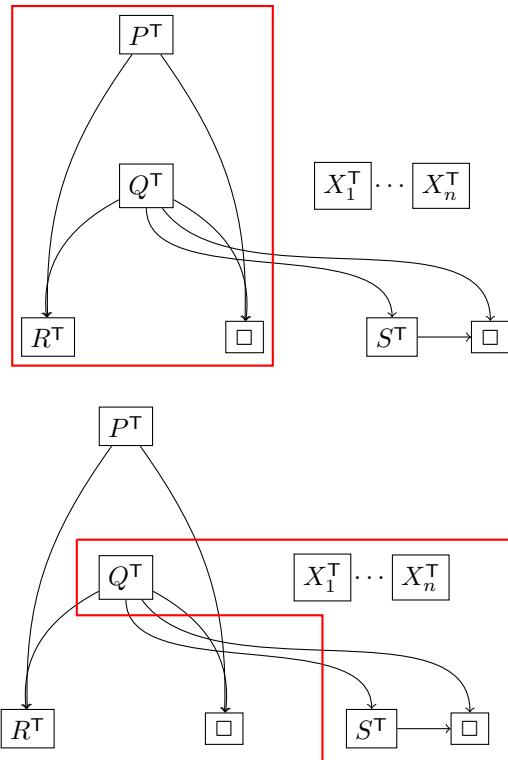


Conflict Graphs: Example (Redundance2)

▷ **Example 12.4.11** Continuing from Example 12.4.7 and Example 12.4.10:

$$\begin{aligned}\Delta &:= P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F \\ \Theta &:= X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F\end{aligned}$$

DPLL on $\Delta; \Theta; \Phi$ with $\Phi := Q^F \vee S^T; Q^F \vee S^F$
 Choice: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied: R^T .





12.5 Clause Learning

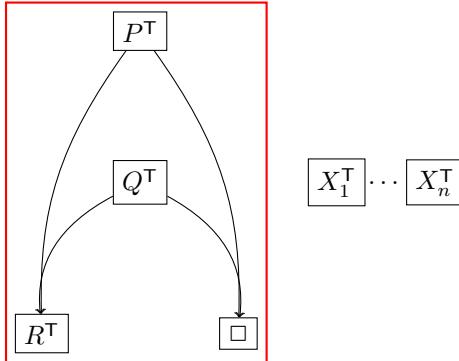
Clause Learning

- ▷ **Observe:** Conflict graphs encode *logical entailments*
- ▷ **Proposition 12.5.1 (Clause Learning)** Let Δ be a set of clauses, and let C be a conflict graph at some time point during a run of DPLL on Δ . Let L be the choice literals in C . Then $\Delta \models \bigvee_{l \in L} \bar{l}$
- ▷ **Intuition:** The negation of the choice literals in a conflict graph is a valid clause.



Clause Learning: Example (Redundance1)

- ▷ **Example 12.5.2** Continuing from Example 12.4.10:
 $\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$
DPLL on Δ ; Θ with $\Theta := X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F$
Choice: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied: R^T .



Learned clause: $P^F \vee Q^F$



The Effect of Learned Clauses (in Redundance1)

- ▷ What happens after we learned a new clause C ?
- 1. **We add C into Δ .** e.g. $C = P^F \vee Q^F$.
- 2. **We retract the last choice l' .** e.g. the choice $l' = Q$.
- ▷ **Observation:** $C = \bigvee_{l \in L} \bar{l}$, where L is the set of conflict literals in conflict graph G

Before we learn the clause, G must contain the most recent choice l' : 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 12.5.3** $l_1 = P$, $C = P^F \vee Q^F$, $l' = Q$.
- ▷ **Observation:** Given the earlier choices l_1, \dots, l_k , after we learned the new clause $C = \bar{l}_1 \vee \dots \vee \bar{l}_k \vee \bar{l}'$, \bar{l}' is now set by UP!
- 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.

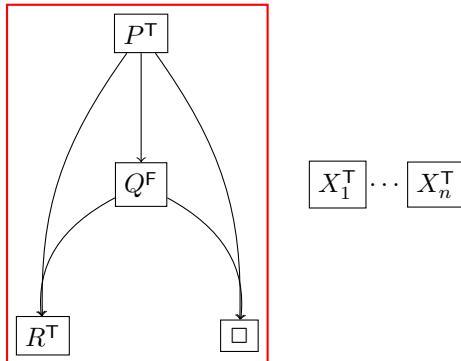


The Effect of Learned Clauses: Example (Redundance1)

- ▷ **Example 12.5.4** Continuing from Example 12.5.2:

$$\begin{aligned}\Delta &:= P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F \\ \Theta &:= X_1^T \vee \dots \vee X_{100}^T; X_1^F \vee \dots \vee X_{100}^F\end{aligned}$$

DPLL on $\Delta; \Theta; \Phi$ with $\Phi := P^F \vee Q^F$
 Choice: $P^T, X_1^T, \dots, X_{100}^T, Q^T$. Implied: Q^F, R^T .

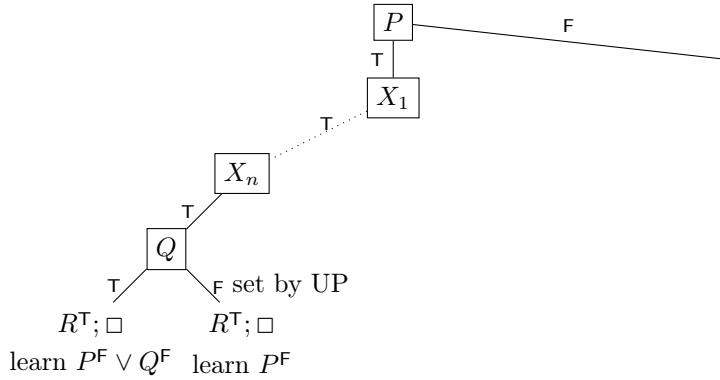


Learned clause: P^F



NOT the same Mistakes over Again: (Redundance1)

- ▷ **Example 12.5.5** Continuing from Example 12.4.10:
 $\Delta := P^F \vee Q^F \vee R^T; P^F \vee Q^F \vee R^F; P^F \vee Q^T \vee R^T; P^F \vee Q^T \vee R^F$
 DPLL on $\Delta; \Theta$ 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.
This is not so in general! (see next slide)

▷ Clause Learning vs. Resolution

▷ Remember (slide 341):

1. **DPLL = tree resolution:** Each derived clause C (not in Δ) is derived anew every time it is used.
2. There exist Δ whose shortest tree-resolution proof is exponentially longer than their shortest (general) resolution proof.

▷ 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 ??, 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 354 – 356 are merely meant to give a *rough intuition* on “backjumping”.
- ▷ **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.

- ▷ But 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  $C = \overline{l_1} \vee \dots \vee \overline{l_k} \vee \overline{l'}$ 
    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  $\{\text{it level}\}$   $L$ 
     $L := L + 1$ 

```



Remarks

▷ WHICH clause(s) to learn?

- ▷ While we only select choice literals, much more can be done.
- ▷ For any **cut** through the conflict graph, with choice literals on the “left-hand” side of the cut and the conflict literals on the right-hand side, the literals on the left border of the cut yield a learnable clause.
- ▷ Must take care to *not learn too many clauses* ...

▷ Origins of clause learning:

- ▷ Clause learning originates from **explanation-based (no-good) learning** developed in the CSP community.
- ▷ The distinguishing feature here is that the “no-good” is a clause:
 - ▷ The exact same type of constraint as the rest of Δ .



12.6 Phase Transitions: Where the *Really* Hard Problems Are

Where Are the Hard Problems?

▷ Err, what?

- ▷ **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 $n > 1000000$.



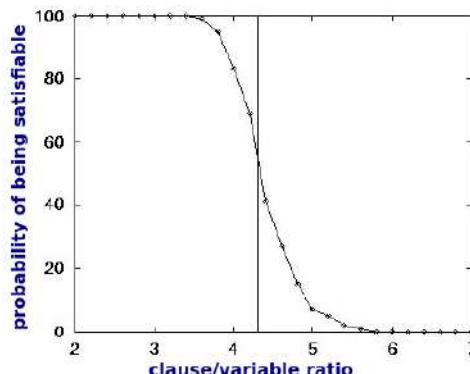
Where Are the Hard Problems?

- ▷ **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]

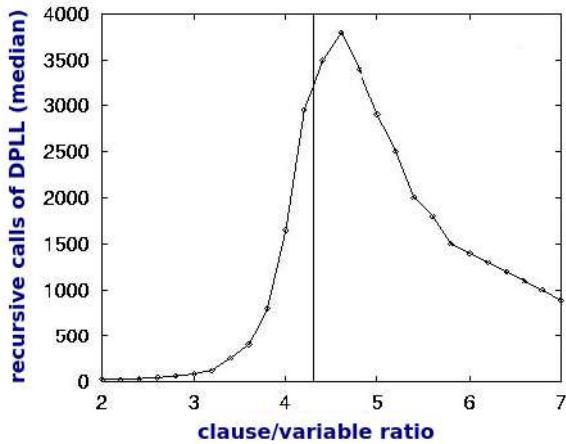
- ▷ **Fixed clause length model:** Fix *clause length k*; *n variables*. Generate *m clauses*, by uniformly choosing *k* variables *P* for each clause *C*, and for each variable *P* deciding uniformly whether to add *P* or *P^F* into *C*.
- ▷ **Order parameter:** Clause/variable ratio $\frac{m}{n}$.
- ▷ **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?

▷ Intuitive explanation:

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

▷ **Conjecture 12.6.1 (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.



Why Should We Care?

▷ **Enlightenment:**

- ▷ Phase transitions contribute to the fundamental understanding of the behavior of search, even if it's only in random distributions.
- ▷ There are interesting theoretical connections to phase transition phenomena in physics. (See [GS05] for a short summary.)

▷ **Ok, but what can we use these results for?**

- ▷ **Benchmark design:** Choose instances from phase transition region.
- ▷ Commonly used in competitions etc. (In SAT, random phase transition formulas are the most difficult for DPLL-style searches.)
- ▷ **Predicting solver performance:** Yes, but very limited because:
- ▷ All this works only for the particular considered *distributions of instances!* Not meaningful for any other instances.



12.7 Conclusion

Summary

- ▷ **SAT solvers** decide satisfiability of CNF formulas. This can be used for deduction, and is highly successful as a general problem solving technique (e.g., in Verification).
- ▷ **DPLL** = backtracking with inference performed by **unit propagation (UP)**, which iteratively instantiates unit clauses and simplifies the formula.
- ▷ DPLL proofs of unsatisfiability correspond to a restricted form of resolution. The restriction forces DPLL to "makes the same mistakes over again".
- ▷ **Implication graphs** capture how UP derives conflicts. Their analysis enables us to do **clause learning**. DPLL with clause learning is called **CDCL**. It corresponds to full resolution, not "making the same mistakes over again".
- ▷ CDCL is state of the art in applications, routinely solving formulas with millions of propositions.
- ▷ In particular random formula distributions, typical problem hardness is characterized by **phase transitions**.



State of the Art in SAT

▷ **SAT competitions:**

- ▷ Since beginning of the 90s <http://www.satcompetition.org/>
- ▷ Distinguish **random** vs. **industrial** vs. **handcrafted** benchmarks.
- ▷ Largest industrial instances: > 1000000 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.



©: Michael Kohlhase

368



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 332) 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)?



©: Michael Kohlhase

369



Reading:

- *Chapter 7: Logical Agents*, Section 7.6.1 [RN09].
 - **Content:** Here, RN describe DPLL, i.e., basically what I cover under “The Davis-Putnam (Logemann-Loveland) Procedure”.

- That's the only thing they cover of this Chapter's material. (And they even mark it as “can be skimmed on first reading”.)
 - This does not do the state of the art in SAT any justice.
- *Chapter 7: Logical Agents*, Sections 7.6.2, 7.6.3, and 7.7 [RN09].
 - **Content:** Sections 7.6.2 and 7.6.3 say a few words on local search for SAT, which I recommend as additional background reading. Section 7.7 describes in quite some detail how to build an agent using propositional logic to take decisions; nice background reading as well.

Chapter 13

First Order Predicate Logic

13.1 Introduction

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,

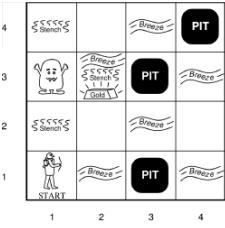
...

▷ **Example 13.1.1** " $\forall x.\text{block}(x) \Rightarrow \text{red}(x)$ "; " $\text{block}(\mathbf{A})$ "



Let's Talk About the Wumpus Instead?

Percepts: [Stench, Breeze, Glitter, Bump, Scream]



▷

- ▷ Cell adjacent to Wumpus: *Stench* (else: *None*).
- ▷ Cell adjacent to Pit: *Breeze* (else: *None*).
- ▷ Cell that contains gold: *Glitter* (else: *None*).
- ▷ You walk into a wall: *Bump* (else: *None*).
- ▷ Wumpus shot by arrow: *Scream* (else: *None*).

▷ **Say, in propositional logic:** “Cell adjacent to Wumpus: *Stench*.”

- ▷ $W_{1,1} \Rightarrow S_{1,2} \wedge S_{2,1}$
- ▷ $W_{1,2} \Rightarrow S_{2,2} \wedge S_{1,1} \wedge S_{1,3}$
- ▷ $W_{1,3} \Rightarrow S_{2,3} \wedge S_{1,2} \wedge S_{1,4}$
- ▷ ...

Note: Even when we *can* describe the problem suitably, for the desired reasoning, the propositional formulation typically is way too large to write (by hand).

▷ **PL1 solution:** “ $\forall x.\text{wumpus}(x) \Rightarrow (\forall y.\text{adjacent}(x, y) \Rightarrow \text{stench}(y))$ ”



©: Michael Kohlhase

371



Blocks/Wumpus, Who Cares? Let's Talk About Numbers!

- ▷ Even worse!
- ▷ **Example 13.1.2 (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 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: “ $\forall x.\text{even}(x) \Rightarrow \text{even}(\succ \succ x)$ ”.



©: Michael Kohlhase

372



Now We're Talking

▷ **Example 13.1.3**

$$\forall n.\text{gt}(n, 2) \Rightarrow \neg (\exists a, b, c.\text{eq}(\text{plus}(\text{pow}(a, n), \text{pow}(b, n)), \text{pow}(c, n)))$$

▷ **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



©: Michael Kohlhase

373



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.)



©: Michael Kohlhase

374



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.*”



©: Michael Kohlhase

375

(A Few) Semantic Technology Applications

Web Queries



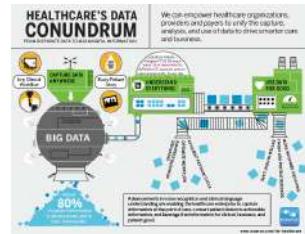
Jeopardy (IBM Watson)



Context-Aware Apps



Healthcare



©: Michael Kohlhase

376

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.



©: Michael Kohlhase

377

13.2 First-Order Logic

First-order logic is the most widely used formal system for modelling knowledge and inference processes. It strikes a very good bargain in the trade-off between expressivity and conceptual and computational complexity. To many people first-order logic is “the logic”, i.e. the only logic worth considering, its applications range from the foundations of mathematics to natural language semantics.

First-Order Predicate Logic (PL^1)

- ▷ **Coverage:** We can talk about (All humans are mortal)
 - ▷ individual things and denote them by variables or constants
 - ▷ properties of individuals, (e.g. being human or mortal)
 - ▷ relations of individuals, (e.g. sibling_of relationship)
 - ▷ functions on individuals, (e.g. the father_of function)

- We can also state the **existence** of an individual with a certain property, or the **universality** of a property.

- ▷ But we cannot state assertions like
 - ▷ There is a surjective function from the natural numbers into the reals.

- ▷ First-Order Predicate Logic has many good properties (complete calculi, compactness, unitary, linear unification, ...)

- ▷ But too weak for formalizing: (at least directly)
 - ▷ natural numbers, torsion groups, calculus, ...
 - ▷ generalized quantifiers (most, at least three, some, ...)



13.2.1 First-Order Logic: Syntax and Semantics

The syntax and semantics of first-order logic is systematically organized in two distinct layers: one for truth values (like in propositional logic) and one for individuals (the new, distinctive feature of first-order logic).

The first step of defining a formal language is to specify the alphabet, here the first-order signatures and their components.

PL^1 Syntax (Signature and Variables)

- ▷ **Definition 13.2.1** First-order logic (PL^1), 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.

- ▷ PL^1 talks about two kinds of objects: (so we have two kinds of symbols)
 - ▷ truth values; sometimes annotated by type σ (like in PL^0)
 - ▷ individuals; sometimes annotated by type ι (numbers, foxes, Pokémons, ...)

▷ **Definition 13.2.2** A **first-order signature** consists of (all disjoint; $k \in \mathbb{N}$)

- ▷ **connectives:** $\Sigma^o = \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \dots\}$ (functions on truth values)
- ▷ **function constants:** $\Sigma_k^f = \{f, g, h, \dots\}$ (functions on individuals)
- ▷ **predicate constants:** $\Sigma_k^p = \{p, q, r, \dots\}$ (relations among inds.)
- ▷ (**Skolem constants:** $\Sigma_k^{sk} = \{f_1^k, f_2^k, \dots\}$) (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 $\Sigma := \Sigma_\iota \cup \Sigma^o$.

We assume a set of **individual variables**: $\mathcal{V}_\iota = \{X_\iota, Y_\iota, Z, X^1_\iota, X^2_\iota\}$ (countably ∞)



We make the deliberate, but non-standard design choice here to include Skolem constants into the signature from the start. These are used in inference systems to give names to objects and construct witnesses. Other than the fact that they are usually introduced by need, they work exactly like regular constants, which makes the inclusion rather painless. As we can never predict how many Skolem constants we are going to need, we give ourselves countably infinitely many for every arity. Our supply of individual variables is countably infinite for the same reason.

The formulae of first-order logic is built up from the signature and variables as terms (to represent individuals) and propositions (to represent propositions). The latter include the propositional connectives, but also quantifiers.

▷ PL¹ Syntax (Formulae)

▷ **Definition 13.2.3 Terms:** $\mathbf{A} \in wff_\iota(\Sigma_\iota)$ (denote individuals: type ι)

- ▷ $\mathcal{V}_\iota \subseteq wff_\iota(\Sigma_\iota)$,
- ▷ if $f \in \Sigma_k^f$ and $\mathbf{A}^i \in wff_\iota(\Sigma_\iota)$ for $i \leq k$, then $f(\mathbf{A}^1, \dots, \mathbf{A}^k) \in wff_\iota(\Sigma_\iota)$.

▷ **Definition 13.2.4 Propositions:** $\mathbf{A} \in wff_o(\Sigma)$ (denote truth values: type o)

- ▷ if $p \in \Sigma_k^p$ and $\mathbf{A}^i \in wff_\iota(\Sigma_\iota)$ for $i \leq k$, then $p(\mathbf{A}^1, \dots, \mathbf{A}^k) \in wff_o(\Sigma)$,
- ▷ if $\mathbf{A}, \mathbf{B} \in wff_o(\Sigma)$ and $X \in \mathcal{V}_\iota$, then $T, \mathbf{A} \wedge \mathbf{B}, \neg \mathbf{A}, \forall X . \mathbf{A} \in wff_o(\Sigma)$.

▷ **Definition 13.2.5** We define the connectives $F, \vee, \Rightarrow, \Leftrightarrow$ via the abbreviations $\mathbf{A} \vee \mathbf{B} := \neg(\neg \mathbf{A} \wedge \neg \mathbf{B})$, $\mathbf{A} \Rightarrow \mathbf{B} := \neg \mathbf{A} \vee \mathbf{B}$, $\mathbf{A} \Leftrightarrow \mathbf{B} := (\mathbf{A} \Rightarrow \mathbf{B}) \wedge (\mathbf{B} \Rightarrow \mathbf{A})$, and $F := \neg T$. We will use them like the primary connectives \wedge and \neg

▷ **Definition 13.2.6** We use $\exists X . \mathbf{A}$ as an abbreviation for $\neg(\forall X . \neg \mathbf{A})$. (existential quantifier)

▷ **Definition 13.2.7** Call formulae without connectives or quantifiers **atomic** else **complex**.



Note: that we only need e.g. conjunction, negation, and universal quantification, all other logical constants can be defined from them (as we will see when we have fixed their interpretations).

Alternative Notations for Quantifiers

Here	Elsewhere
$\forall x . \mathbf{A}$	$\bigwedge x . \mathbf{A}$ $(x) . \mathbf{A}$
$\exists x . \mathbf{A}$	$\bigvee x . \mathbf{A}$



©: Michael Kohlhase

381



The introduction of quantifiers to first-order logic brings a new phenomenon: variables that are under the scope of a quantifiers will behave very differently from the ones that are not. Therefore we build up a vocabulary that distinguishes the two.

Free and Bound Variables

▷ **Definition 13.2.8** We call an occurrence of a variable X **bound** in a formula \mathbf{A} , iff it occurs in a sub-formula $\forall X . \mathbf{B}$ of \mathbf{A} . We call a variable occurrence **free** otherwise.

For a formula \mathbf{A} , we will use $BVar(\mathbf{A})$ (and $free(\mathbf{A})$) for the set of bound (free) variables of \mathbf{A} , i.e. variables that have a free/bound occurrence in \mathbf{A} .

▷ **Definition 13.2.9** We define the set $free(\mathbf{A})$ of **free variables** of a formula \mathbf{A} :

$$\begin{aligned} free(X) &:= \{X\} \\ free(f(\mathbf{A}_1, \dots, \mathbf{A}_n)) &:= \bigcup_{1 \leq i \leq n} free(\mathbf{A}_i) \\ free(p(\mathbf{A}_1, \dots, \mathbf{A}_n)) &:= \bigcup_{1 \leq i \leq n} free(\mathbf{A}_i) \\ free(\neg \mathbf{A}) &:= free(\mathbf{A}) \\ free(\mathbf{A} \wedge \mathbf{B}) &:= free(\mathbf{A}) \cup free(\mathbf{B}) \\ free(\forall X . \mathbf{A}) &:= free(\mathbf{A}) \setminus \{X\} \end{aligned}$$

▷ **Definition 13.2.10** We call a formula \mathbf{A} **closed** or **ground**, iff $free(\mathbf{A}) = \emptyset$. We call a closed proposition a **sentence**, and denote the set of all ground terms with $cwff_t(\Sigma_t)$ and the set of sentences with $cwff_o(\Sigma_t)$.

▷ **Axiom 13.2.11** **Bound variables** can be renamed, i.e. any subterm $\forall X . \mathbf{B}$ of a formula \mathbf{A} can be replaced by $\mathbf{A}' := (\forall Y . \mathbf{B}')$, where \mathbf{B}' arises from \mathbf{B} by replacing all $X \in free(\mathbf{B})$ with a new variable Y that does not occur in \mathbf{A} . We call \mathbf{A}' an **alphabetical variant** of \mathbf{A} .



©: Michael Kohlhase

382



We will be mainly interested in (sets of) sentences – i.e. closed propositions – as the representations of meaningful statements about individuals. Indeed, we will see below that free variables do not give us expressivity, since they behave like constants and could be replaced by them in all situations, except the recursive definition of quantified formulae. Indeed in all situations where variables occur freely, they have the character of meta-variables, i.e. syntactic placeholders that can be instantiated with terms when needed in an inference calculus.

The semantics of first-order logic is a Tarski-style set-theoretic semantics where the atomic syntactic entities are interpreted by mapping them into a well-understood structure, a first-order universe that is just an arbitrary set.

Semantics of PL¹ (Models)

- ▷ We fix the **Universe** $\mathcal{D}_o = \{\top, \perp\}$ of **truth values**.
- ▷ We assume an arbitrary **universe** $\mathcal{D}_t \neq \emptyset$ of **individuals** (this choice is a **parameter to the semantics**)
- ▷ **Definition 13.2.12** An **interpretation** \mathcal{I} assigns values to constants, e.g.
 - ▷ $\mathcal{I}(\neg): \mathcal{D}_o \rightarrow \mathcal{D}_o$ with $\top \mapsto \perp, \perp \mapsto \top$, and $\mathcal{I}(\wedge) = \dots$ (as in PL⁰)
 - ▷ $\mathcal{I}: \Sigma_k^f \rightarrow \mathcal{D}_t^k \rightarrow \mathcal{D}_t$ (interpret function symbols as arbitrary functions)
 - ▷ $\mathcal{I}: \Sigma_k^p \rightarrow \mathcal{P}(\mathcal{D}_t^k)$ (interpret predicates as arbitrary relations)
- ▷ **Definition 13.2.13** A **variable assignment** $\varphi: \mathcal{V}_t \rightarrow \mathcal{D}_t$ maps variables into the universe.

A first-order **Model** $\mathcal{M} = \langle \mathcal{D}_t, \mathcal{I} \rangle$ consists of a universe \mathcal{D}_t and an interpretation \mathcal{I} .



©: Michael Kohlhase

383



We do not have to make the universe of truth values part of the model, since it is always the same; we determine the model by choosing a universe and an interpretation function.

Given a first-order model, we can define the evaluation function as a homomorphism over the construction of formulae.

Semantics of PL¹ (Evaluation)

- ▷ Given a model $\langle \mathcal{D}, \mathcal{I} \rangle$, the **value function** \mathcal{I}_φ is recursively defined: (two parts: **terms & propositions**)
 - ▷ $\mathcal{I}_\varphi: wff_t(\Sigma_t) \rightarrow \mathcal{D}_t$ assigns values to terms.
 - ▷ $\mathcal{I}_\varphi(X) := \varphi(X)$ and
 - ▷ $\mathcal{I}_\varphi(f(\mathbf{A}_1, \dots, \mathbf{A}_k)) := \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1), \dots, \mathcal{I}_\varphi(\mathbf{A}_k))$
- ▷ $\mathcal{I}_\varphi: wff_o(\Sigma) \rightarrow \mathcal{D}_o$ assigns values to formulae:
 - ▷ $\mathcal{I}_\varphi(T) = \mathcal{I}(T) = \top$,
 - ▷ $\mathcal{I}_\varphi(\neg \mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathbf{A}))$
 - ▷ $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathbf{A}), \mathcal{I}_\varphi(\mathbf{B}))$ (just as in PL⁰)
 - ▷ $\mathcal{I}_\varphi(p(\mathbf{A}^1, \dots, \mathbf{A}^k)) := \top$, iff $\langle \mathcal{I}_\varphi(\mathbf{A}^1), \dots, \mathcal{I}_\varphi(\mathbf{A}^k) \rangle \in \mathcal{I}(p)$
 - ▷ $\mathcal{I}_\varphi(\forall X. \mathbf{A}) := \top$, iff $\mathcal{I}_{\varphi, [a/X]}(\mathbf{A}) = \top$ for all $a \in \mathcal{D}_t$.



©: Michael Kohlhase

384



The only new (and interesting) case in this definition is the quantifier case, there we define the value of a quantified formula by the value of its scope – *but with an extended variable assignment*. Note that by passing to the scope \mathbf{A} of $\forall x. \mathbf{A}$, the occurrences of the variable x in \mathbf{A} that were bound in $\forall x. \mathbf{A}$ become free and are amenable to evaluation by the variable assignment $\psi := \varphi, [a/X]$. Note that as an extension of φ , the assignment ψ supplies exactly the right value for x in \mathbf{A} . This variability of the variable assignment in the definition value function justifies the somewhat complex setup of first-order evaluation, where we have the (static) interpretation function for the symbols from the signature and the (dynamic) variable assignment for the variables.

Note furthermore, that the value $\mathcal{I}_\varphi(\exists x.\mathbf{A})$ of $\exists x.\mathbf{A}$, which we have defined to be $\neg(\forall x.\neg\mathbf{A})$ is true, iff it is not the case that $\mathcal{I}_\varphi(\forall x.\neg\mathbf{A}) = \mathcal{I}_\psi(\neg\mathbf{A}) = \top$ for all $a \in \mathcal{D}_t$ and $\psi := \varphi, [a/X]$. This is the case, iff $\mathcal{I}_\psi(\mathbf{A}) = \top$ for some $a \in \mathcal{D}_t$. So our definition of the existential quantifier yields the appropriate semantics.

13.2.2 First-Order Substitutions

We will now turn our attention to substitutions, special formula-to-formula mappings that operationalize the intuition that (individual) variables stand for arbitrary terms.

Substitutions on Terms

- ▷ **Intuition:** If \mathbf{B} is a term and X is a variable, then we denote the result of systematically replacing all occurrences of X in a term \mathbf{A} by \mathbf{B} with $[\mathbf{B}/X](\mathbf{A})$.
- ▷ **Problem:** What about $[Z/Y], [Y/X](X)$, is that Y or Z ?
- ▷ **Folklore:** $[Z/Y], [Y/X](X) = Y$, but $[Z/Y]([Y/X](X)) = Z$ of course.
(Parallel application)
- ▷ **Definition 13.2.14** We call $\sigma: wff_t(\Sigma_t) \rightarrow wff_t(\Sigma_t)$ a **substitution**, iff $\sigma(f(\mathbf{A}_1, \dots, \mathbf{A}_n)) = f(\sigma(\mathbf{A}_1), \dots, \sigma(\mathbf{A}_n))$ and the **support** $\text{supp}(\sigma) := \{X \mid \sigma(X) \neq X\}$ of σ is finite.
- ▷ **Observation 13.2.15** Note that a substitution σ is determined by its values on variables alone, thus we can write σ as $\sigma|_{\mathcal{V}_t} = \{[\sigma(X)/X] \mid X \in \text{supp}(\sigma)\}$.
- ▷ **Notation 13.2.16** We denote the substitution σ with $\text{supp}(\sigma) = \{x^i \mid 1 \leq i \leq n\}$ and $\sigma(x^i) = \mathbf{A}_i$ by $[\mathbf{A}_1/x^1], \dots, [\mathbf{A}_n/x^n]$.
- ▷ **Example 13.2.17** $[a/x], [f(b)/y], [a/z]$ instantiates $g(x, y, h(z))$ to $g(a, f(b), h(a))$.
- ▷ **Definition 13.2.18** We call $\text{intro}(\sigma) := \bigcup_{X \in \text{supp}(\sigma)} \text{free}(\sigma(X))$ the set of variables introduced by σ .



The extension of a substitution is an important operation, which you will run into from time to time. Given a substitution σ , a variable x , and an expression \mathbf{A} , $\sigma, [\mathbf{A}/x]$ extends σ with a new value for x . The intuition is that the values right of the comma overwrite the pairs in the substitution on the left, which already has a value for x , even though the representation of σ may not show it.

Substitution Extension

- ▷ **Notation 13.2.19 (Substitution Extension)** Let σ be a substitution, then we denote with $\sigma, [\mathbf{A}/X]$ the function $\{(Y, \mathbf{A}) \in \sigma \mid Y \neq X\} \cup \{(X, \mathbf{A})\}$.
($\sigma, [\mathbf{A}/X]$ coincides with σ of X , and gives the result \mathbf{A} there.)
- ▷ **Note:** If σ is a substitution, then $\sigma, [\mathbf{A}/X]$ is also a substitution.
- ▷ **Definition 13.2.20** If σ is a substitution, then we call $\sigma, [\mathbf{A}/X]$ the **extension** of σ by $[\mathbf{A}/X]$.

- ▷ We also need the dual operation: removing a variable from the support
- ▷ **Definition 13.2.21** We can **discharge** a variable X from a substitution σ by $\sigma_{-X} := \sigma, [X/X]$.



Note that the use of the comma notation for substitutions defined in Notation 13.2.16 is consistent with substitution extension. We can view a substitution $[a/x], [f(b)/y]$ as the extension of the empty substitution (the identity function on variables) by $[f(b)/y]$ and then by $[a/x]$. Note furthermore, that substitution extension is not commutative in general.

For first-order substitutions we need to extend the substitutions defined on terms to act on propositions. This is technically more involved, since we have to take care of bound variables.

Substitutions on Propositions

- ▷ **Problem:** We want to extend substitutions to propositions, in particular to quantified formulae: What is $\sigma(\forall X . A)$?
- ▷ **Idea:** σ should not instantiate bound variables. $([A/X](\forall X . B) = \forall A . B'$
ill-formed)
- ▷ **Definition 13.2.22** $\sigma(\forall X . A) := (\forall X . \sigma_{-X}(A))$.
- ▷ **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 13.2.23 (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 $\sigma(A) := \sigma(A')$.



We now introduce a central tool for reasoning about the semantics of substitutions: the “substitution-value Lemma”, which relates the process of instantiation to (semantic) evaluation. This result will be the motor of all soundness proofs on axioms and inference rules acting on variables via substitutions. In fact, any logic with variables and substitutions will have (to have) some form of a substitution-value Lemma to get the meta-theory going, so it is usually the first target in any development of such a logic.

We establish the substitution-value Lemma for first-order logic in two steps, first on terms, where it is very simple, and then on propositions.

Substitution Value Lemma for Terms

- ▷ **Lemma 13.2.24** 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 :
- P.1.1 **depth=0:**

P.1.1.1 Then \mathbf{A} is a variable (say Y), or constant, so we have three cases

P.1.1.1.1 $\mathbf{A} = Y = X$: then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](X)) = \mathcal{I}_\varphi(\mathbf{B}) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(\mathbf{A})$.

P.1.1.1.2 $\mathbf{A} = Y \neq X$: then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(\mathbf{A})$.

P.1.1.1.3 \mathbf{A} is a constant: analogous to the preceding case ($Y \neq X$)

P.1.1.2 This completes the base case (depth = 0). \square

P.1.2 depth > 0: then $\mathbf{A} = f(\mathbf{A}_1, \dots, \mathbf{A}_n)$ and we have

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

by inductive hypothesis

P.1.2.2 This completes the inductive case, and we have proven the assertion \square

\square



Substitution Value Lemma for Propositions

▷ **Lemma 13.2.25** $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\psi(\mathbf{A})$, where $\psi = \varphi, [\mathcal{I}_\varphi(\mathbf{B})/X]$.

▷ **Proof:** by induction on the number n of connectives and quantifiers in \mathbf{A}

P.1.1 $n = 0$: then \mathbf{A} is an atomic proposition, and we can argue like in the inductive case of the substitution value lemma for terms.

P.1.2 $n > 0$ and $\mathbf{A} = \neg \mathbf{B}$ or $\mathbf{A} = \mathbf{C} \circ \mathbf{D}$: Here we argue like in the inductive case of the term lemma as well.

P.1.3 $n > 0$ and $\mathbf{A} = \forall Y . \mathbf{C}$ where (wlog) $X \neq Y$:

P.1.3.1 then $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\psi(\forall Y . \mathbf{C}) = \top$, iff $\mathcal{I}_{\psi,[a/Y]}(\mathbf{C}) = \top$ for all $a \in \mathcal{D}_\psi$.

P.1.3.2 But $\mathcal{I}_{\psi,[a/Y]}(\mathbf{C}) = \mathcal{I}_{\varphi,[a/Y]}([\mathbf{B}/X](\mathbf{C})) = \top$, by inductive hypothesis.

P.1.3.3 So $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\varphi(\forall Y . [\mathbf{B}/X](\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\forall Y . \mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}))$

\square

\square



To understand the proof fully, you should think about where the *wlog* – it stands for *without loss of generality* – comes from.

13.3 First-Order Calculi

In this section we will introduce two reasoning calculi for first-order logic, both were invented by Gerhard Gentzen in the 1930's and are very much related. The "natural deduction" calculus was created in order to model the natural mode of reasoning e.g. in everyday mathematical practice.

This calculus was intended as a counter-approach to the well-known Hilbert-style calculi, which were mainly used as theoretical devices for studying reasoning in principle, not for modeling particular reasoning styles.

EdN:5

The “sequent calculus” was a rationalized version and extension of the natural deduction calculus that makes certain meta-proofs simpler to push through⁵.

Both calculi have a similar structure, which is motivated by the human-orientation: rather than using a minimal set of inference rules, they provide two inference rules for every connective and quantifier, one “introduction rule” (an inference rule that derives a formula with that symbol at the head) and one “elimination rule” (an inference rule that acts on a formula with this head and derives a set of subformulae).

This allows us to introduce the calculi in two stages, first for the propositional connectives and then extend this to a calculus for first-order logic by adding rules for the quantifiers.

13.3.1 Propositional Natural Deduction Calculus

We will now introduce the “natural deduction” calculus for propositional logic. The calculus was created in order to model the natural mode of reasoning e.g. in everyday mathematical practice. This calculus was intended as a counter-approach to the well-known Hilbert style calculi, which were mainly used as theoretical devices for studying reasoning in principle, not for modeling particular reasoning styles.

Rather than using a minimal set of inference rules, the natural deduction calculus provides two/three inference rules for every connective and quantifier, one “introduction rule” (an inference rule that derives a formula with that symbol at the head) and one “elimination rule” (an inference rule that acts on a formula with this head and derives a set of subformulae).

Calculi: Natural Deduction (\mathcal{ND}^0 ; Gentzen [Gen34])

▷ Idea: \mathcal{ND}^0 tries to mimic human theorem proving behavior (non-minimal)

▷ Definition 13.3.1 The propositional natural deduction calculus \mathcal{ND}^0 has rules for the introduction and elimination of connectives

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

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



The most characteristic rule in the natural deduction calculus is the $\Rightarrow I$ rule. It corresponds to the mathematical way of proving an implication $\mathbf{A} \Rightarrow \mathbf{B}$: We assume that \mathbf{A} is true and show \mathbf{B} from this assumption. When we can do this we discharge (get rid of) the assumption and conclude $\mathbf{A} \Rightarrow \mathbf{B}$. This mode of reasoning is called **hypothetical reasoning**. Note that the local hypothesis is **discharged** by the rule $\Rightarrow I$, i.e. it cannot be used in any other part of the proof. As the $\Rightarrow I$

⁵EDNOTE: say something about cut elimination/analytical calculi somewhere

rules may be nested, we decorate both the rule and the corresponding assumption with a marker (here the number 1).

Let us now consider an example of hypothetical reasoning in action.

Natural Deduction: Examples

▷ Example 13.3.2 (Inference with Local Hypotheses)

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

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



©: Michael Kohlhase

391



Here we see reasoning with local hypotheses at work. In the left example, we assume the formula $\mathbf{A} \wedge \mathbf{B}$ and can use it in the proof until it is discharged by the rule $\wedge E_l$ on the bottom – therefore we decorate the hypothesis and the rule by corresponding numbers (here the label “1”). Note the assumption $\mathbf{A} \wedge \mathbf{B}$ is *local to the proof fragment* delineated by the corresponding hypothesis and the discharging rule, i.e. even if this proof is only a fragment of a larger proof, then we cannot use its hypothesis anywhere else. Note also that we can use as many copies of the local hypothesis as we need; they are all discharged at the same time.

In the right example we see that local hypotheses can be nested as long as hypotheses are kept local. In particular, we may not use the hypothesis \mathbf{B} after the $\Rightarrow I^2$, e.g. to continue with a $\Rightarrow E$.

One of the nice things about the natural deduction calculus is that the deduction theorem is almost trivial to prove. In a sense, the triviality of the deduction theorem is the central idea of the calculus and the feature that makes it so natural.

A Deduction Theorem for \mathcal{ND}^0

▷ Theorem 13.3.3 $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$, iff $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$.

▷ Proof: We show the two directions separately

P.1 If $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$, then $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$ by $\Rightarrow I$, and

P.2 If $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$, then $\mathcal{H}, \mathcal{A} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$ by weakening and $\mathcal{H}, \mathcal{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$ by $\Rightarrow E$. \square



©: Michael Kohlhase

392



Another characteristic of the natural deduction calculus is that it has inference rules (introduction and elimination rules) for all connectives. So we extend the set of rules from Definition 13.3.1 for disjunction, negation and falsity.

More Rules for Natural Deduction

▷ Definition 13.3.4 \mathcal{ND}^0 has the following additional rules for the remaining

connectives.

$$\begin{array}{c}
 \frac{\mathbf{A}}{\mathbf{A} \vee \mathbf{B}} \vee I_l \quad \frac{\mathbf{B}}{\mathbf{A} \vee \mathbf{B}} \vee I_r \quad \frac{\begin{array}{c} \mathbf{A} \vee \mathbf{B} \\ \vdots \\ \mathbf{C} \\ \mathbf{C} \end{array}}{\mathbf{C}} \vee E^1 \\
 \frac{\mathbf{[A]}^1}{\vdots} \\
 \frac{\mathbf{F}}{\neg \mathbf{A}} \neg I^1 \quad \frac{\neg \neg \mathbf{A}}{\mathbf{A}} \neg E \\
 \frac{\neg \mathbf{A} \quad \mathbf{A}}{\mathbf{F}} FI \quad \frac{\mathbf{F}}{\mathbf{A}} FE
 \end{array}$$



Natural Deduction in Sequent Calculus Formulation

- ▷ Idea: Explicit representation of hypotheses (lift calculus to judgments)
- ▷ Definition 13.3.5 A judgment is a meta-statement about the provability of propositions
- ▷ Definition 13.3.6 A sequent is a judgment of the form $\mathcal{H} \vdash \mathbf{A}$ about the provability of the formula \mathbf{A} from the set \mathcal{H} of hypotheses.
Write $\vdash \mathbf{A}$ for $\emptyset \vdash \mathbf{A}$.
- ▷ Idea: Reformulate ND rules so that they act on sequents
- ▷ Example 13.3.7 We give the sequent-style version of Example 13.3.2

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

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

$$\begin{array}{c}
 \frac{}{\Gamma, \mathbf{A} \vdash \mathbf{A}} \text{Ax} \quad \frac{\Gamma \vdash \mathbf{B}}{\Gamma, \mathbf{A} \vdash \mathbf{B}} \text{weaken} \quad \frac{}{\Gamma \vdash \mathbf{A} \vee \neg \mathbf{A}} \text{TND} \\
 \frac{\Gamma \vdash \mathbf{A} \quad \Gamma \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}} \wedge I \quad \frac{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}{\Gamma \vdash \mathbf{A}} \wedge E_l \quad \frac{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}{\Gamma \vdash \mathbf{B}} \wedge E_r \\
 \frac{\Gamma \vdash \mathbf{A}}{\Gamma \vdash \mathbf{A} \vee \mathbf{B}} \vee I_l \quad \frac{\Gamma \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \vee \mathbf{B}} \vee I_r \quad \frac{\Gamma \vdash \mathbf{A} \vee \mathbf{B} \quad \Gamma, \mathbf{A} \vdash \mathbf{C} \quad \Gamma, \mathbf{B} \vdash \mathbf{C}}{\Gamma \vdash \mathbf{C}} \vee E \\
 \frac{\Gamma, \mathbf{A} \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \Rightarrow \mathbf{B}} \Rightarrow I \quad \frac{\Gamma \vdash \mathbf{A} \Rightarrow \mathbf{B} \quad \Gamma \vdash \mathbf{A}}{\Gamma \vdash \mathbf{B}} \Rightarrow E \\
 \frac{\Gamma, \mathbf{A} \vdash F}{\Gamma \vdash \neg \mathbf{A}} \neg I \quad \frac{\Gamma \vdash \neg \neg \mathbf{A}}{\mathbf{A}} \neg E \\
 \frac{\Gamma \vdash \neg \mathbf{A} \quad \Gamma \vdash \mathbf{A}}{\Gamma \vdash F} FI \quad \frac{\Gamma \vdash F}{\Gamma \vdash \mathbf{A}} FE
 \end{array}$$



Linearized Notation for (Sequent-Style) ND Proofs

▷ Linearized notation for sequent-style ND proofs

$$\begin{array}{lll}
 1. \quad \mathcal{H}_1 \vdash \mathbf{A}_1 \quad (\mathcal{J}_1) & & \\
 2. \quad \mathcal{H}_2 \vdash \mathbf{A}_2 \quad (\mathcal{J}_2) & \text{corresponds to} & \frac{\mathcal{H}_1 \vdash \mathbf{A}_1 \quad \mathcal{H}_2 \vdash \mathbf{A}_2}{\mathcal{H}_3 \vdash \mathbf{A}_3} \mathcal{R} \\
 3. \quad \mathcal{H}_3 \vdash \mathbf{A}_3 \quad (\mathcal{R}1, 2) & &
 \end{array}$$

▷ **Example 13.3.9** We show a linearized version of Example 13.3.7

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



Each line in the table represents one inference step in the proof. It consists of line number (for referencing), a formula for the asserted property, a justification via a ND rules (and the lines this one is derived from), and finally a list of line numbers of proof steps that are local hypotheses in effect for the current line.

To obtain a first-order calculus, we have to extend \mathcal{ND}^0 with (introduction and elimination) rules for the quantifiers.

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

▷ Rules for propositional connectives just as always

▷ **Definition 13.3.10 (New Quantifier Rules)** The first-order natural deduction calculus \mathcal{ND}^1 extends \mathcal{ND}^0 by the following four rules

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

$$[[c/X](\mathbf{A})]^1$$

$$\frac{[\mathbf{B}/X](\mathbf{A})}{\exists X . \mathbf{A}} \exists I \quad \frac{\begin{array}{c} \exists X . \mathbf{A} \\ \vdots \\ \mathbf{C} \end{array}}{\mathbf{C}} \exists E^1$$

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



The intuition behind the rule $\forall I$ is that a formula \mathbf{A} with a (free) variable X can be generalized to $\forall X . \mathbf{A}$, if X stands for an arbitrary object, i.e. there are no restricting assumptions about X . The $\forall E$ rule is just a substitution rule that allows to instantiate arbitrary terms \mathbf{B} for X in \mathbf{A} . The $\exists I$ rule says if we have a witness \mathbf{B} for X in \mathbf{A} (i.e. a concrete term \mathbf{B} that makes \mathbf{A} true), then we can existentially close \mathbf{A} . The $\exists E$ rule corresponds to the common mathematical practice, where we give objects we know exist a new name c and continue the proof by reasoning about this concrete object c . Anything we can prove from the assumption $[c/X](\mathbf{A})$ we can prove outright if $\exists X . \mathbf{A}$ is known.

A Complex \mathcal{ND}^1 Example

▷ **Example 13.3.11** We prove $\neg(\forall X . P(X)) \vdash_{\mathcal{ND}^1} \exists X . \neg P(X)$.

$$\frac{[\neg P(X)]^2}{\frac{[\neg(\exists X . \neg P(X))]^1}{\frac{\exists X . \neg P(X)}{FI}} \exists I} F$$

$$\frac{\neg\neg P(X)}{\neg P(X)} \neg I^2$$

$$\frac{\neg P(X)}{P(X)} \neg E$$

$$\frac{P(X)}{\forall X . P(X)} \forall I$$

$$\frac{\neg(\forall X . P(X)) \quad \forall X . P(X)}{FI} FI$$

$$\frac{F}{\frac{\neg\neg(\exists X . \neg P(X))}{\exists X . \neg P(X)} \neg I^1} \neg E$$



This is the classical formulation of the calculus of natural deduction. To prepare the things we want to do later (and to get around the somewhat un-licensed extension by hypothetical reasoning in the calculus), we will reformulate the calculus by lifting it to the “judgements level”. Instead of postulating rules that make statements about the validity of propositions, we postulate rules that make state about derivability. This move allows us to make the respective local hypotheses

in ND derivations into syntactic parts of the objects (we call them “sequents”) manipulated by the inference rules.

First-Order Natural Deduction in Sequent Formulation

▷ Rules for propositional connectives just as always

▷ **Definition 13.3.12 (New Quantifier Rules)**

$$\frac{\Gamma \vdash A \quad X \notin \text{free}(\Gamma)}{\Gamma \vdash \forall X.A} \forall I \quad \frac{\Gamma \vdash \forall X.A}{\Gamma \vdash [B/X](A)} \forall E$$

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



©: Michael Kohlhase

399



Natural Deduction with Equality

▷ **Definition 13.3.13 (First-Order Logic with Equality)** We extend PL¹ with a new logical symbol for equality $= \in \Sigma_2^p$ and fix its semantics to $\mathcal{I}(=) := \{(x, x) \mid x \in D_\ell\}$. We call the extended logic **first-order logic with equality** (PL₌¹)

▷ We now extend natural deduction as well.

▷ **Definition 13.3.14** For the calculus of natural deduction with equality $\mathcal{ND}_{=}^1$ we add the following two equality rules to \mathcal{ND}^1 to deal with equality:

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

where $\mathbf{C}[\mathbf{A}]_p$ if the formula \mathbf{C} has a subterm \mathbf{A} at position p and $[\mathbf{B}/p]\mathbf{C}$ is the result of replacing that subterm with \mathbf{B} .

▷ In many ways equivalence behaves like equality, so we will use the following derived rules in \mathcal{ND}^1 :

$$\frac{}{\mathbf{A} \Leftrightarrow \mathbf{A}} \Leftrightarrow I \quad \frac{\mathbf{A} \Leftrightarrow \mathbf{B} \quad \mathbf{C}[\mathbf{A}]_p \Leftrightarrow \mathbf{E}}{[\mathbf{B}/p]\mathbf{C}} \Leftrightarrow = E$$



©: Michael Kohlhase

400



Again, we have two rules that follow the introduction/elimination pattern of natural deduction calculi.

To make sure that we understand the constructions here, let us get back to the “replacement at position” operation used in the equality rules.

Positions in Formulae

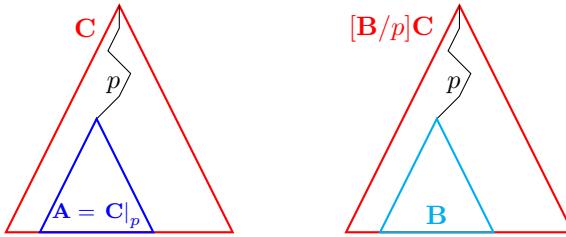
▷ **Idea:** Formulae are (naturally) trees, so we can use tree positions to talk about subformulae

▷ **Definition 13.3.15** A **formula position** p is a list of natural numbers that in each node of a formula (tree) specifies into which child to descend. For a formula \mathbf{A} we denote the **subformula at p** with $\mathbf{A}|_p$.

▷ We will sometimes write a formula \mathbf{C} as $\mathbf{C}[\mathbf{A}]_p$ to indicate that \mathbf{C} the subformula \mathbf{A} at position p .

▷ **Definition 13.3.16** Let p be a position, then $[\mathbf{A}/p]\mathbf{C}$ is the formula obtained from \mathbf{C} by **replacing** the subformula at position p by \mathbf{A} .

▷ **Example 13.3.17 (Schematically)**



The operation of replacing a subformula at position p is quite different from e.g. (first-order) substitutions:

- We are replacing subformulae with subformulae instead of instantiating variables with terms.
- substitutions replace all occurrences of a variable in a formula, whereas formula replacement only affects the (one) subformula at **position p** .

We conclude this Subsection with an extended example: the proof of a classical mathematical result in the natural deduction calculus with equality. This shows us that we can derive strong properties about complex situations (here the real numbers; an uncountably infinite set of numbers).

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

▷ We can do real Maths with \mathcal{ND}_\equiv^1 :

▷ **Theorem 13.3.18** $\sqrt{2}$ is irrational

Proof: We prove the assertion by contradiction

P.1 Assume that $\sqrt{2}$ is rational.

P.2 Then there are numbers p and q such that $\sqrt{2} = p/q$.

P.3 So we know $2q^2 = p^2$.

P.4 But $2q^2$ has an odd number of prime factors while p^2 an even number.

P.5 This is a contradiction (since they are equal), so we have proven the assertion □



If we want to formalize this into \mathcal{ND}_\equiv^1 , we have to write down all the assertions in the proof steps in PL¹ syntax and come up with justifications for them in terms of \mathcal{ND}_\equiv^1 inference rules. The next

two slides show such a proof, where we write $\text{prime}(n)$ to denote that n is prime, use $\#\#(n)$ for the number of prime factors of a number n , and write $\text{irr}(r)$ if r is irrational.

\mathcal{MD}^1_{\equiv} 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		$\forall n, p. \text{tp} \Rightarrow \#\#(pn) = \#\#(n) + 1$	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	$\text{tp} \Rightarrow \#\#(2q^2) = \#\#(q^2) + 1$	$\forall E^2(1)$



Lines 6 and 9 are local hypotheses for the proof (they only have an implicit counterpart in the inference rules as defined above). Finally we have abbreviated the arithmetic simplification of line 9 with the justification “arith” to avoid having to formalize elementary arithmetic.

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

13		tp	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		$\text{irr}(\sqrt{2})$	$\neg E^2(23)$



We observe that the \mathcal{MD}^1 proof is much more detailed, and needs quite a few Lemmata about $\#\#$ to go through. Furthermore, we have added a definition of irrationality (and treat definitional equality via the equality rules). Apart from these artefacts of formalization, the two representations of proofs correspond to each other very directly.

13.4 First-Order Predicate Logic (Conclusion)

Summary

- ▷ [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 .



©: Michael Kohlhase

405



Reading:

- *Chapter 8: First-Order Logic*, Sections 8.1 and 8.2 in [RN09]
 - **Content:** A less formal account of what I cover in “Syntax” and “Semantics”. Contains different examples, and complementary explanations. Nice as additional background reading.
- Sections 8.3 and 8.4 provide additional material on using PL1, and on modeling in PL1, that I don’t cover in this lecture. Nice reading, not required for exam.
- *Chapter 9: Inference in First-Order Logic*, Section 9.5.1 in [RN09]
 - **Content:** A very brief (2 pages) description of what I cover in “Normal Forms”. Much less formal; I couldn’t find where (if at all) RN cover transformation into prenex normal form. Can serve as additional reading, can’t replace the lecture.

Excursion: A full analysis of any calculus needs a completeness proof. We will not cover this in the course, but provide one for the calculi introduced so far in Section B.2.

Chapter 14

First-Order Inference

14.1 First-Order Inference with Tableaux

14.1.1 First-Order Tableaux

Test Calculi: Tableaux and Model Generation

▷ **Idea:** instead of showing $\emptyset \vdash Th$, show $\neg Th \vdash trouble$ (use \perp for trouble)

▷ **Example 14.1.1** Tableau Calculi try to construct models.

Tableau Refutation (Validity)	Model generation (Satisfiability)
$\models P \wedge Q \Rightarrow Q \wedge P$	$\models P \wedge (Q \vee \neg R) \wedge \neg Q$
$P \wedge Q \Rightarrow Q \wedge P^F$ $P \wedge Q^T$ $Q \wedge P^F$ P^T Q^T $P^F \quad Q^F$ $\perp \quad \perp$	$P \wedge (Q \vee \neg R) \wedge \neg Q^T$ $P \wedge (Q \vee \neg R)^T$ $\neg Q^T$ Q^F P^T $Q \vee \neg R^T$ $Q^T \quad \neg R^T$ $\perp \quad R^F$
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)



Tableau calculi develop a formula in a tree-shaped arrangement that represents a case analysis on when a formula can be made true (or false). Therefore the formulae are decorated with exponents that hold the intended truth value.

On the left we have a refutation tableau that analyzes a negated formula (it is decorated with the intended truth value F). Both branches contain an elementary contradiction \perp .

On the right we have a model generation tableau, which analyzes a positive formula (it is decorated with the intended truth value T). This tableau uses the same rules as the refutation tableau, but makes a case analysis of when this formula can be satisfied. In this case we have a closed branch and an open one, which corresponds a model).

Now that we have seen the examples, we can write down the tableau rules formally.

Analytical Tableaux (Formal Treatment of \mathcal{T}_0)

- ▷ formula is analyzed in a tree to determine satisfiability
- ▷ branches correspond to valuations (models)
- ▷ one per connective

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

- ▷ Use rules exhaustively as long as they contribute new material
- ▷ **Definition 14.1.2** Call a tableau **saturated**, iff no rule applies, and a branch **closed**, iff it ends in \perp , else **open**. (open branches in saturated tableaux yield models)
- ▷ **Definition 14.1.3 (\mathcal{T}_0 -Theorem/Derivability)** \mathbf{A} is a \mathcal{T}_0 -theorem ($\vdash_{\mathcal{T}_0} \mathbf{A}$), iff there is a closed tableau with \mathbf{A}^F at the root.
- $\Phi \subseteq wff_o(\mathcal{V}_o)$ **derives** \mathbf{A} in \mathcal{T}_0 ($\Phi \vdash_{\mathcal{T}_0} \mathbf{A}$), iff there is a closed tableau starting with \mathbf{A}^F and Φ^T .



These inference rules act on tableaux have to be read as follows: if the formulae over the line appear in a tableau branch, then the branch can be extended by the formulae or branches below the line. There are two rules for each primary connective, and a branch closing rule that adds the special symbol \perp (for unsatisfiability) to a branch.

We use the tableau rules with the convention that they are only applied, if they contribute new material to the branch. This ensures termination of the tableau procedure for propositional logic (every rule eliminates one primary connective).

Definition 14.1.4 We will call a closed tableau with the signed formula \mathbf{A}^α at the root a **tableau refutation** for \mathcal{A}^α .

The saturated tableau represents a full case analysis of what is necessary to give \mathbf{A} the truth value α ; since all branches are closed (contain contradictions) this is impossible.

Definition 14.1.5 We will call a tableau refutation for \mathbf{A}^F a **tableau proof** for \mathbf{A} , since it refutes the possibility of finding a model where \mathbf{A} evaluates to F . Thus \mathbf{A} must evaluate to T in all models, which is just our definition of validity.

Thus the tableau procedure can be used as a calculus for propositional logic. In contrast to the calculus in **?sec.hilbert?** it does not prove a theorem \mathbf{A} by deriving it from a set of axioms, but it proves it by refuting its negation. Such calculi are called negative or test calculi. Generally negative calculi have computational advantages over positive ones, since they have a built-in sense of direction.

We have rules for all the necessary connectives (we restrict ourselves to \wedge and \neg , since the others can be expressed in terms of these two via the propositional identities above. For instance, we can write $\mathbf{A} \vee \mathbf{B}$ as $\neg(\neg \mathbf{A} \wedge \neg \mathbf{B})$, and $\mathbf{A} \Rightarrow \mathbf{B}$ as $\neg \mathbf{A} \vee \mathbf{B}, \dots$)

We will now extend the propositional tableau techniques to first-order logic. We only have to add two new rules for the universal quantifiers (in positive and negative polarity).

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

- ▷ Refutation calculus based on trees of labeled formulae
- ▷ Tableau-Rules: \mathcal{T}_0 (propositional tableau rules) plus

$$\frac{\forall X \cdot \mathbf{A}^T \quad \mathbf{C} \in \text{cwf}(\Sigma_i)}{[\mathbf{C}/X](\mathbf{A})^T} \mathcal{T}_1:\forall \quad \frac{\forall X \cdot \mathbf{A}^F \quad c \in (\Sigma_0^{sk} \setminus \mathcal{H})}{[c/X](\mathbf{A})^F} \mathcal{T}_1:\exists$$



©: Michael Kohlhase

408



The rule $\mathcal{T}_1:\forall$ rule operationalizes the intuition that a universally quantified formula is true, iff all of the instances of the scope are. To understand the $\mathcal{T}_1:\exists$ rule, we have to keep in mind that $\exists X \cdot \mathbf{A}$ abbreviates $\neg(\forall X \cdot \neg \mathbf{A})$, so that we have to read $\forall X \cdot \mathbf{A}^F$ existentially — i.e. as $\exists X \cdot \neg \mathbf{A}^T$, stating that there is an object with property $\neg \mathbf{A}$. In this situation, we can simply give this object a name: c , which we take from our (infinite) set of witness constants Σ_0^{sk} , which we have given ourselves expressly for this purpose when we defined first-order syntax. In other words $[c/X](\neg \mathbf{A})^T = [c/X](\mathbf{A})^F$ holds, and this is just the conclusion of the $\mathcal{T}_1:\exists$ rule.

Note that the $\mathcal{T}_1:\forall$ rule is computationally extremely inefficient: we have to guess an (i.e. in a search setting to systematically consider all) instance $\mathbf{C} \in \text{wff}(\Sigma_i)$ for X . This makes the rule infinitely branching.

14.1.2 Free Variable Tableaux

In the next calculus we will try to remedy the computational inefficiency of the $\mathcal{T}_1:\forall$ rule. We do this by delaying the choice in the universal rule.

Free variable Tableaux (\mathcal{T}_1^f)

- ▷ Refutation calculus based on trees of labeled formulae

- ▷ \mathcal{T}_0 (propositional tableau rules) plus

- ▷ Quantifier rules:

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

- ▷ Generalized cut rule: $\mathcal{T}_1^f:\perp$ instantiates the whole tableau by σ .

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

- ▷ **Advantage:** no guessing necessary in $\mathcal{T}_1^f:\forall$ -rule

- ▷ **New:** find suitable substitution

(most general unifier)



©: Michael Kohlhase

409



Metavariables: Instead of guessing a concrete instance for the universally quantified variable as in the $\mathcal{T}_1:\forall$ rule, $\mathcal{T}_1^f:\forall$ instantiates it with a new meta-variable Y , which will be instantiated by need

in the course of the derivation.

Skolem terms as witnesses: The introduction of meta-variables makes it necessary to extend the treatment of witnesses in the existential rule. Intuitively, we cannot simply invent a new name, since the meaning of the body \mathbf{A} may contain meta-variables introduced by the $\mathcal{T}_1^f:\forall$ rule. As we do not know their values yet, the witness for the existential statement in the antecedent of the $\mathcal{T}_1^f:\exists$ rule needs to depend on that. So witness it using a witness term, concretely by applying a Skolem function to the meta-variables in \mathbf{A} .

Instantiating Metavariables: Finally, the $\mathcal{T}_1^f:\perp$ rule completes the treatment of meta-variables, it allows to instantiate the whole tableau in a way that the current branch closes. This leaves us with the problem of finding substitutions that make two terms equal.

Multiplicity in Tableaux

- ▷ **Observation 14.1.6** All \mathcal{T}_1^f rules except $\mathcal{T}_1^f:\forall$ only need to be applied once.
- ▷ **Example 14.1.7** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists x.p(x))$.

Start, close branch	use $\mathcal{T}_1^f:\forall$ again
$(p(a) \vee p(b)) \Rightarrow (\exists x.p(x))^F$ $p(a) \vee p(b)^T$ $\exists x.p(x)^F$ $\forall x.\neg p(x)^T$ $\neg p(y)^T$ $p(y)^F$ $p(a)^T \quad \quad p(b)^T$ $\perp : [a/y] \quad \quad$	$(p(a) \vee p(b)) \Rightarrow (\exists x.p(x))^F$ $p(a) \vee p(b)^T$ $\exists x.p(x)^F$ $\forall x.\neg p(x)^T$ $\neg p(a)^T$ $p(a)^F$ $p(a)^T \quad \quad p(b)^T$ $\perp : [a/y] \quad \quad \neg p(z)^T$ $p(z)^F$ $\perp : [b/z]$

- ▷ **Definition 14.1.8** Let \mathcal{T} be a tableau for \mathbf{A} , and a positive occurrence of $\forall x.\mathbf{B}$ in \mathbf{A} , then we call the number of applications of $\mathcal{T}_1^f:\forall$ to $\forall x.\mathbf{B}$ its **multiplicity**.
- ▷ **Observation 14.1.9** 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 . \square
- ▷ **Theorem 14.1.10** \mathcal{T}_1^f is only complete with unbounded multiplicities.
- ▷ **Proof Sketch:** Otherwise validity in PL¹ would be decidable. \square



Treating $\mathcal{T}_1^f:\perp$

- ▷ The $\mathcal{T}_1^f:\perp$ rule instantiates the whole tableau.
- ▷ There may be more than one $\mathcal{T}_1^f:\perp$ opportunity on a branch

▷ **Example 14.1.11** 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.

▷ 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 (later)



Spanning Matings for $\mathcal{T}_1^f : \perp$

▷ **Observation 14.1.12** \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.

▷ **Definition 14.1.13** Let \mathcal{T} be a \mathcal{T}_1^f tableau, then we call a unification problem $\mathcal{E} := \mathbf{A}_1 =? \mathbf{B}_1 \wedge \dots \wedge \mathbf{A}_n =? \mathbf{B}_n$ a **mating** for \mathcal{T} , iff \mathbf{A}_i^T and \mathbf{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 \mathbf{A}_i^T and \mathbf{B}_i^F for some i .

▷ **Theorem 14.1.14** 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.



14.1.3 First-Order Unification

We will now look into the problem of finding a substitution σ that make two terms equal (we say it unifies them) in more detail. The presentation of the unification algorithm we give here “transformation-based” this has been a very influential way to treat certain algorithms in theoretical computer science.

A transformation-based view of algorithms: The “transformation-based” view of algorithms divides two concerns in presenting and reasoning about algorithms according to Kowalski’s slogan⁶

$$\text{computation} = \text{logic} + \text{control}$$

The computational paradigm highlighted by this quote is that (many) algorithms can be thought of as manipulating representations of the problem at hand and transforming them into a form that makes it simple to read off solutions. Given this, we can simplify thinking and reasoning about such algorithms by separating out their “logical” part, which deals with what is concerned with how the problem representations can be manipulated in principle from the “control” part, which is concerned with questions about when to apply which transformations.

It turns out that many questions about the algorithms can already be answered on the “logic” level, and that the “logical” analysis of the algorithm can already give strong hints as to how to optimize control.

In fact we will only concern ourselves with the “logical” analysis of unification here.

The first step towards a theory of unification is to take a closer look at the problem itself. A first set of examples show that we have multiple solutions to the problem of finding substitutions that make two terms equal. But we also see that these are related in a systematic way.

Unification (Definitions)

- ▷ **Problem:** For given terms \mathbf{A} and \mathbf{B} find a substitution σ , such that $\sigma(\mathbf{A}) = \sigma(\mathbf{B})$.
- ▷ **Notation 14.1.15** We write term pairs as $\mathbf{A} = ? \mathbf{B}$ e.g. $f(X) = ? f(g(Y))$
- ▷ **Solutions** (e.g. $[g(a)/X], [a/Y], [g(g(a))/X], [g(a)/Y]$, or $[g(Z)/X], [Z/Y]$) are called **unifiers**, $\mathbf{U}(\mathbf{A} = ? \mathbf{B}) := \{\sigma \mid \sigma(\mathbf{A}) = \sigma(\mathbf{B})\}$
- ▷ **Idea:** find representatives in $\mathbf{U}(\mathbf{A} = ? \mathbf{B})$, that generate the set of solutions
- ▷ **Definition 14.1.16** 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 14.1.17** σ is called a **most general unifier** of \mathbf{A} and \mathbf{B} , iff it is minimal in $\mathbf{U}(\mathbf{A} = ? \mathbf{B})$ wrt. \leq [$\text{free}(\mathbf{A}) \cup \text{free}(\mathbf{B})$].



The idea behind a most general unifier is that all other unifiers can be obtained from it by (further) instantiation. In an automated theorem proving setting, this means that using most general unifiers is the least committed choice — any other choice of unifiers (that would be necessary for completeness) can later be obtained by other substitutions.

Note that there is a subtlety in the definition of the ordering on substitutions: we only compare on a subset of the variables. The reason for this is that we have defined substitutions to be total on (the infinite set of) variables for flexibility, but in the applications (see the definition of a most general unifiers), we are only interested in a subset of variables: the ones that occur in the initial problem formulation. Intuitively, we do not care what the unifiers do off that set. If we did not have the restriction to the set W of variables, the ordering relation on substitutions would become much too fine-grained to be useful (i.e. to guarantee unique most general unifiers in our case).

⁶EDNOTE: find the reference, and see what he really said

Now that we have defined the problem, we can turn to the unification algorithm itself. We will define it in a way that is very similar to logic programming: we first define a calculus that generates “solved forms” (formulae from which we can read off the solution) and reason about control later. In this case we will reason that control does not matter.

Unification (Equational Systems)

- ▷ **Idea:** Unification is equation solving.
- ▷ **Definition 14.1.18** We call a formula $\mathbf{A}^1 =? \mathbf{B}^1 \wedge \dots \wedge \mathbf{A}^n =? \mathbf{B}^n$ an **equational system** iff $\mathbf{A}^i, \mathbf{B}^i \in wff_{\ell}(\Sigma_{\ell}, \mathcal{V}_{\ell})$.
- ▷ We consider equational systems as sets of equations (\wedge is ACI), and equations as two-element multisets ($=?$ is C).



In principle, unification problems are sets of equations, which we write as conjunctions, since all of them have to be solved for finding a unifier. Note that it is not a problem for the “logical view” that the representation as conjunctions induces an order, since we know that conjunction is associative, commutative and idempotent, i.e. that conjuncts do not have an intrinsic order or multiplicity, if we consider two equational problems as equal, if they are equivalent as propositional formulae. In the same way, we will abstract from the order in equations, since we know that the equality relation is symmetric. Of course we would have to deal with this somehow in the implementation (typically, we would implement equational problems as lists of pairs), but that belongs into the “control” aspect of the algorithm, which we are abstracting from at the moment.

Solved forms and Most General Unifiers

- ▷ **Definition 14.1.19** We call a pair $\mathbf{A} =? \mathbf{B}$ **solved** in a unification problem \mathcal{E} , iff $\mathbf{A} = X$, $\mathcal{E} = X =? \mathbf{A} \wedge \mathcal{E}$, and $X \notin (\text{free}(\mathbf{A}) \cup \text{free}(\mathcal{E}))$. We call an unification problem \mathcal{E} a **solved form**, iff all its pairs are solved.
- ▷ **Lemma 14.1.20** *Solved forms are of the form $X^1 =? \mathbf{B}^1 \wedge \dots \wedge X^n =? \mathbf{B}^n$ where the X^i are distinct and $X^i \notin \text{free}(\mathbf{B}^j)$.*
- ▷ **Definition 14.1.21** Any substitution $\sigma = [\mathbf{B}^1/X^1], \dots, [\mathbf{B}^n/X^n]$ induces a solved unification problem $\mathcal{E}_{\sigma} := (X^1 =? \mathbf{B}^1 \wedge \dots \wedge X^n =? \mathbf{B}^n)$.
- ▷ **Lemma 14.1.22** *If $\mathcal{E} = X^1 =? \mathbf{B}^1 \wedge \dots \wedge X^n =? \mathbf{B}^n$ is a solved form, then \mathcal{E} has the unique most general unifier $\sigma_{\mathcal{E}} := [\mathbf{B}^1/X^1], \dots, [\mathbf{B}^n/X^n]$.*
- ▷ **Proof:** Let $\theta \in \mathbf{U}(\mathcal{E})$

P.1 then $\theta(X^i) = \theta(\mathbf{B}^i) = \theta \circ \sigma_{\mathcal{E}}(X^i)$

P.2 and thus $\theta = \theta \circ \sigma_{\mathcal{E}}[\text{supp}(\sigma)]$. □

Note: we can rename the introduced variables in most general unifiers!



It is essential to our “logical” analysis of the unification algorithm that we arrive at equational problems whose unifiers we can read off easily. Solved forms serve that need perfectly as Lemma 14.1.22 shows.

Given the idea that unification problems can be expressed as formulae, we can express the algorithm in three simple rules that transform unification problems into solved forms (or unsolvable ones).

▷ Unification Algorithm

▷ **Definition 14.1.23** Inference system \mathcal{U}

$$\frac{\mathcal{E} \wedge f(\mathbf{A}^1, \dots, \mathbf{A}^n) =? f(\mathbf{B}^1, \dots, \mathbf{B}^n)}{\mathcal{E} \wedge \mathbf{A}^1 =? \mathbf{B}^1 \wedge \dots \wedge \mathbf{A}^n =? \mathbf{B}^n} \mathcal{U}_{\text{dec}} \quad \frac{\mathcal{E} \wedge \mathbf{A} =? \mathbf{A}}{\mathcal{E}} \mathcal{U}_{\text{triv}}$$

$$\frac{\mathcal{E} \wedge X =? \mathbf{A} \quad X \notin \text{free}(\mathbf{A}) \quad X \in \text{free}(\mathcal{E})}{[\mathbf{A}/X](\mathcal{E}) \wedge X =? \mathbf{A}} \mathcal{U}_{\text{elim}}$$

▷ **Lemma 14.1.24** \mathcal{U} is *correct*: $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ implies $\mathbf{U}(\mathcal{F}) \subseteq \mathbf{U}(\mathcal{E})$

▷ **Lemma 14.1.25** \mathcal{U} is *complete*: $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ implies $\mathbf{U}(\mathcal{E}) \subseteq \mathbf{U}(\mathcal{F})$

▷ **Lemma 14.1.26** \mathcal{U} is *confluent*: the order of derivations does not matter

▷ **Corollary 14.1.27** 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 □



The decomposition rule \mathcal{U}_{dec} is completely straightforward, but note that it transforms one unification pair into multiple argument pairs; this is the reason, why we have to directly use unification problems with multiple pairs in \mathcal{U} .

Note furthermore, that we could have restricted the $\mathcal{U}_{\text{triv}}$ rule to variable-variable pairs, since for any other pair, we can decompose until only variables are left. Here we observe, that constant-constant pairs can be decomposed with the \mathcal{U}_{dec} rule in the somewhat degenerate case without arguments.

Finally, we observe that the first of the two variable conditions in $\mathcal{U}_{\text{elim}}$ (the “occurs-in-check”) makes sure that we only apply the transformation to unifiable unification problems, whereas the second one is a termination condition that prevents the rule to be applied twice.

The notion of completeness and correctness is a bit different than that for calculi that we compare to the entailment relation. We can think of the “logical system of unifiability” with the model class of sets of substitutions, where a set satisfies an equational problem \mathcal{E} , iff all of its members are unifiers. This view induces the soundness and completeness notions presented above.

The three meta-properties above are relatively trivial, but somewhat tedious to prove, so we leave the proofs as an exercise to the reader.

We now fortify our intuition about the unification calculus by two examples. Note that we only need to pursue one possible \mathcal{U} derivation since we have confluence.

Unification Examples

Example 14.1.28 Two similar unification problems:

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



We will now convince ourselves that there cannot be any infinite sequences of transformations in \mathcal{U} . Termination is an important property for an algorithm.

The proof we present here is very typical for termination proofs. We map unification problems into a partially ordered set $\langle S, \prec \rangle$ where we know that there cannot be any infinitely descending sequences (we think of this as measuring the unification problems). Then we show that all transformations in \mathcal{U} strictly decrease the measure of the unification problems and argue that if there were an infinite transformation in \mathcal{U} , then there would be an infinite descending chain in S , which contradicts our choice of $\langle S, \prec \rangle$.

The crucial step in coming up with such proofs is finding the right partially ordered set. Fortunately, there are some tools we can make use of. We know that $\langle \mathbb{N}, < \rangle$ is terminating, and there are some ways of lifting component orderings to complex structures. For instance it is well-known that the lexicographic ordering lifts a terminating ordering to a terminating ordering on finite-dimensional Cartesian spaces. We show a similar, but less known construction with multisets for our proof.

Unification (Termination)

▷ **Definition 14.1.29** Let S and T be multisets and \prec a partial ordering on $S \cup T$. Then we define $(S \prec^m T)$, 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 14.1.30** If \prec is total/terminating on S , then \prec^m is total/terminating on $\mathcal{P}(S)$.

▷ **Lemma 14.1.31** \mathcal{U} is terminating (any \mathcal{U} -derivation is finite)

▷ **Proof:** We prove termination by mapping \mathcal{U} transformation into a Noetherian space.

P.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}

P.2 The lexicographic order \prec on pairs $\mu(\mathcal{E})$ is decreased by all inference rules.

P.2.1 \mathcal{U} dec and \mathcal{U} triv decrease the multiset of term depths without increasing the unsolved variables

P.2.2 \mathcal{U} elim decreases the number of unsolved variables (by one), but may increase term depths. \square



But it is very simple to create terminating calculi, e.g. by having no inference rules. So there is one more step to go to turn the termination result into a decidability result: we must make sure that we have enough inference rules so that any unification problem is transformed into solved form if it is unifiable.

Unification (decidable)

▷ **Definition 14.1.32** We call an equational problem \mathcal{E} **\mathcal{U} -reducible**, iff there is a \mathcal{U} -step $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ from \mathcal{E} .

▷ **Lemma 14.1.33** If \mathcal{E} is unifiable but not solved, then it is \mathcal{U} -reducible

▷ **Proof:** We assume that \mathcal{E} is unifiable but unsolved and show the \mathcal{U} rule that applies.

P.1 There is an unsolved pair $\mathbf{A} = ? \mathbf{B}$ in $\mathcal{E} = \mathcal{E}' \wedge \mathbf{A} = ? \mathbf{B}$.

P.2 we have two cases

P.2.1 $\mathbf{A}, \mathbf{B} \notin \mathcal{V}_i$: then $\mathbf{A} = f(\mathbf{A}^1 \dots \mathbf{A}^n)$ and $\mathbf{B} = f(\mathbf{B}^1 \dots \mathbf{B}^n)$, and thus \mathcal{U} dec is applicable

P.2.2 $\mathbf{A} = X \in \text{free}(\mathcal{E})$: then \mathcal{U} elim (if $\mathbf{B} \neq X$) or \mathcal{U} triv (if $\mathbf{B} = X$) is applicable. \square

▷ **Corollary 14.1.34** Unification is decidable in PL¹.

▷ **Proof Idea:** \mathcal{U} -irreducible sets of equations can be obtained in finite time by Lemma 14.1.31 and are either solved or unsolvable by Lemma 14.1.33, so they provide the answer. \square



Excuse: Now that we understand basic unification theory, we can come to the meta-theoretical properties of the tableau calculus. We delegate this discussion to Section B.3.

Tableau Reasons about Blocks

▷ **Example 14.1.35 (Reasoning about Blocks)** returing to slide 370



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

$$\begin{array}{c}
 \forall X . \text{block}(X) \Rightarrow \text{red}(X)^T \\
 \text{block}(\mathbf{A})^T \\
 \text{red}(\mathbf{A})^F \\
 \text{block}(Y) \Rightarrow \text{red}(Y)^T \\
 \text{block}(Y)^F \mid \text{red}(\mathbf{A})^T \\
 \perp : [\mathbf{A}/Y] \quad \perp
 \end{array}$$



14.2 First-Order Resolution

First-Order Resolution (CNF)

▷ **Definition 14.2.1** The **Conjunctive Normal Form Calculus CNF**¹ is given by the inference rules of CNF^0 extended by

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

$$\frac{(\forall X . \mathbf{A})^F \vee \mathbf{C} \ \{X_1, \dots, X_k\} = \text{free}(\forall X . \mathbf{A})}{[f_n^k(X^1, \dots, X^k)/X](\mathbf{A})^F \vee \mathbf{C}}$$

$\text{CNF}^1(\Phi)$ is the set of all clauses that can be derived from Φ .

▷ **Definition 14.2.2 (First-Order Resolution Calculus)** First-order resolution is a refutation calculus that manipulates formulae in conjunctive normal form. \mathcal{R}^1 has two inference rules

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



Excuse: Again, we relegate the meta-theoretical properties of the first-order resolution calculus to Section B.4.

14.2.1 Resolution Examples

Col. West, a Criminal?

▷ **Example 14.2.3** 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 $P^F \vee Q^F \vee R^F \vee S^T$.

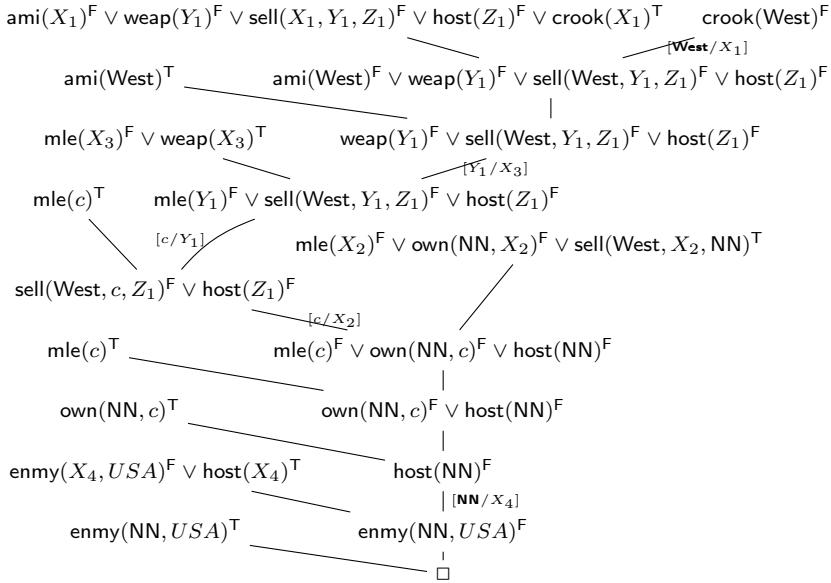


Col. West, a *Criminal*?

- ▷ *It is a crime for an American to sell weapons to hostile nations:*
Clause: $\text{ami}(X_1) \wedge \text{weap}(Y_1) \wedge \text{sell}(X_1, Y_1, Z_1) \wedge \text{host}(Z_1) \Rightarrow \text{crook}(X_1)$
- ▷ *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 14.2.4 From [RN09]

Everyone who loves all animals is loved by someone.

Anyone who kills an animal is loved by noone.

Jack loves all animals.

Cats are animals.

Either Jack or curiosity killed the cat (whose name is “Garfield”).

Prove that curiosity killed the cat.



Curiosity Killed the Cat? Clauses

▷ Everyone who loves all animals is loved by someone:

$\forall X. (\forall Y. \text{animal}(Y) \Rightarrow \text{love}(X, Y)) \Rightarrow (\exists Z. \text{love}(Z, X))$

Clauses: $\text{animal}(g(X_1))^\top \vee \text{love}(g(X_1), X_1)^\top$ and $\text{love}(X_2, f(X_2))^F \vee \text{love}(g(X_2), X_2)^\top$

▷ Anyone who kills an animal is loved by noone:

$\forall X. \exists Y. \text{animal}(Y) \wedge \text{kill}(X, Y) \Rightarrow (\forall Z. \neg \text{love}(Z, X))$

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: $\text{animal}(X_4)^F \vee \text{love(jack}, X_4)^\top$

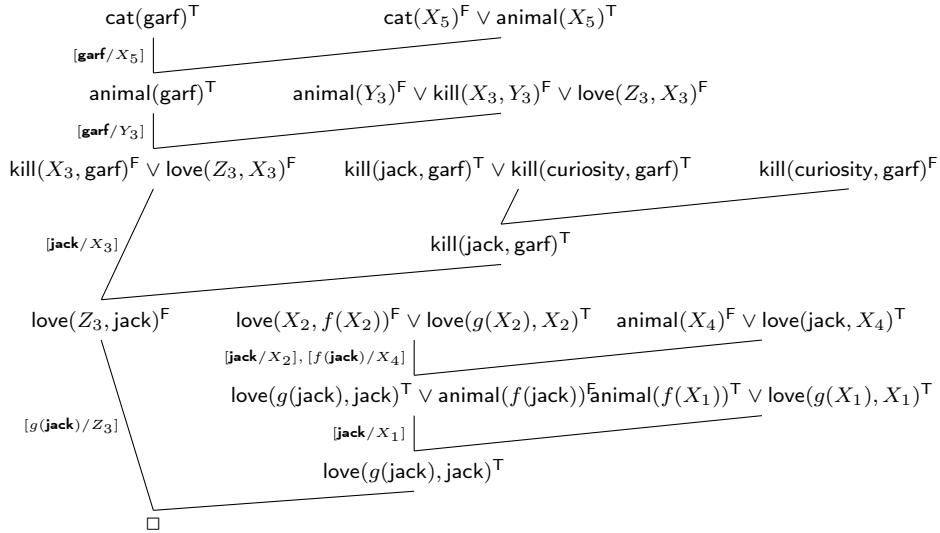
▷ Cats are animals:

Clause: $\text{cat}(X_5)^F \vee \text{animal}(X_5)^\top$

▷ Either Jack or curiosity killed the cat (whose name is “Garfield”):
 Clauses: $\text{kill(jack, garf)}^T \vee \text{kill(curiosity, garf)}^T$ and cat(garf)^T



Curiosity Killed the Cat! PL1 Resolution Proof



Excursion: A full analysis of any calculus needs a completeness proof. We will not cover this in the course, but provide one for the calculi introduced so far in the appendix.

Chapter 15

Logic Programming as Resolution Theorem Proving

To understand ProLog better, we can interpret the language of ProLog as resolution in PL¹

We know all this already

- ▷ Goals, goal-sets, rules, and facts are just clauses. (called “Horn clauses”)
- ▷ **Observation 15.0.1** 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 15.0.2** (goal set) $?- G_1, \dots, G_n.$ corresponds to $G_1^F \vee \dots \vee G_n^F$
- ▷ **Observation 15.0.3** (fact) $F.$ corresponds to the unit clause $F^T.$
- ▷ **Definition 15.0.4** A Horn clause is a clause with at most one positive literal.
- ▷ **Note:** backchaining becomes (hyper)-resolution (special case for rule with facts) $\frac{P^T \vee A \quad P^F \vee B}{A \vee B}$ positive, unit-resulting hyperresolution (PURR)



©: Michael Kohlhase

428



This observation helps us understand ProLog better, and use implementation techniques from theorem proving.

PROLOG (Horn Logic)

- ▷ **Definition 15.0.5** A clause is called a Horn clause, iff contains at most one positive literal, i.e. if it is of the form $B_1^F \vee \dots \vee B_n^F \vee A^T$ ($A:-B_1, \dots, B_n.$)
 - ▷ **Rule clause:** general case, e.g. $\text{fallible}(X) :- \text{human}(X).$
 - ▷ **Fact clause:** no negative literals, e.g. $\text{human}(\text{sokrates}).$
 - ▷ **Program:** set of rule and fact clauses
 - ▷ **Query:** no positive literals: e.g. $?- \text{fallible}(X), \text{greek}(X).$

- ▷ **Definition 15.0.6** Horn logic is the formal system whose language is the set of Horn clauses together with the calculus given by MP, $\wedge I$, and Subst.



PROLOG: Our Example

- ▷ **Program:**

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

- ▷ **Example 15.0.7 (Query)** ?– fallible(X),greek(X).

- ▷ **Answer substitution:** [sokrates/X]



Why Only Horn Clauses?

- ▷ General clauses of the form A₁,...,A_n :- B₁,...,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

- ▷ **Deduction:** knowledge extension

$$\frac{rains \Rightarrow wet_street \quad rains}{wet_street} D$$

- ▷ **Abduction:** explanation

$$\frac{rains \Rightarrow wet_street \quad wet_street}{rains} A$$

- ▷ **Induction:** learning rules

$$\frac{wet_street \quad rains}{rains \Rightarrow wet_street} I$$



Part IV

Planning & Acting

This part covers the AI subfield of “planning”, i.e. search problem solving with a structured representation language for environment state and actions – In planning, the focus is on the latter.

Chapter 16

Planning I: Framework

16.1 Planning: Introduction

Reminder: Classical Search Problems Chapter 7



- ▷ States: Card positions (e.g. position_Jspades=Qhearts).
- ▷ Actions: Card moves (e.g. move_Jspades_Qhearts_freecell4).
- ▷ Initial state: Start configuration.
- ▷ Goal states: All cards “home”.
- ▷ Solutions: Card moves solving this game.

© SOME RIGHTS RESERVED

© Michael Kohlhase 433 FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Planning

- ▷ **Ambition:** Write one program that can solve all classical search problems.
- ▷ **Reminder:** (see Chapter 7)
 - ▷ The **blackbox description** of a problem Π is an API (a programming interface) providing functionality allowing to construct the state space: `InitialState()`, `GoalTest(s)`, ...

- ▷ “Specifying the problem” = 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” = writing a problem description.
- ▷ Here, “problem description language” = **planning language**.



How does a planning language describe a problem?

- ▷ A *logical description* of the possible **states** (vs. Blackbox: data structures).
(E.g.: predicate $Eq(.,.)$.)
- ▷ A *logical description* of the **initial state I** (vs. data structures).
(E.g.: $Eq(x, 1)$.)
- ▷ A *logical description* of the **goal condition G** (vs. a goal-test function).
(E.g.: $Eq(x, 2)$.)
- ▷ A *logical description* of the 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 $Eq(x, 2) \wedge \neg Eq(x, 1)$ ”.)
- ▷ Solution (**plan**) = sequence of actions from A , transforming I into a state that
satisfies G .
(E.g.: “increment x ”.)

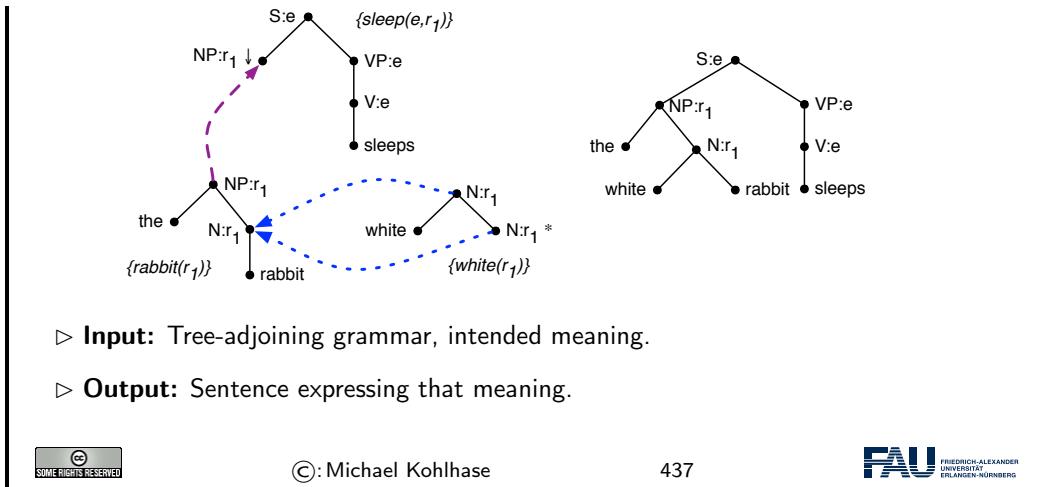


Planning Language Overview

- ▷ **Disclaimer:** Planning languages go way beyond classical search problems. There are variants for inaccessible, stochastic, dynamic, continuous, and multi-agent settings.
- ▷ We focus on classical search for simplicity (and practical relevance).
- ▷ For a comprehensive overview, see [GNT04].

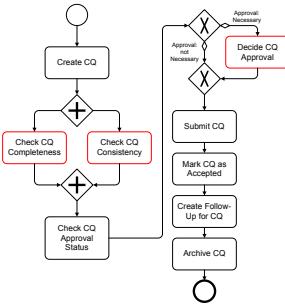


Application: Natural Language Generation



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

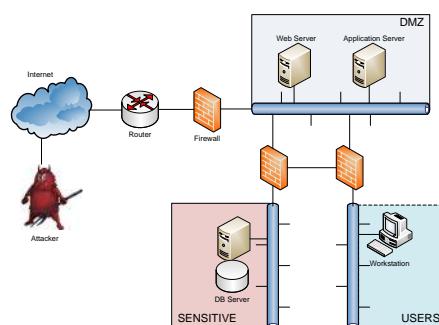


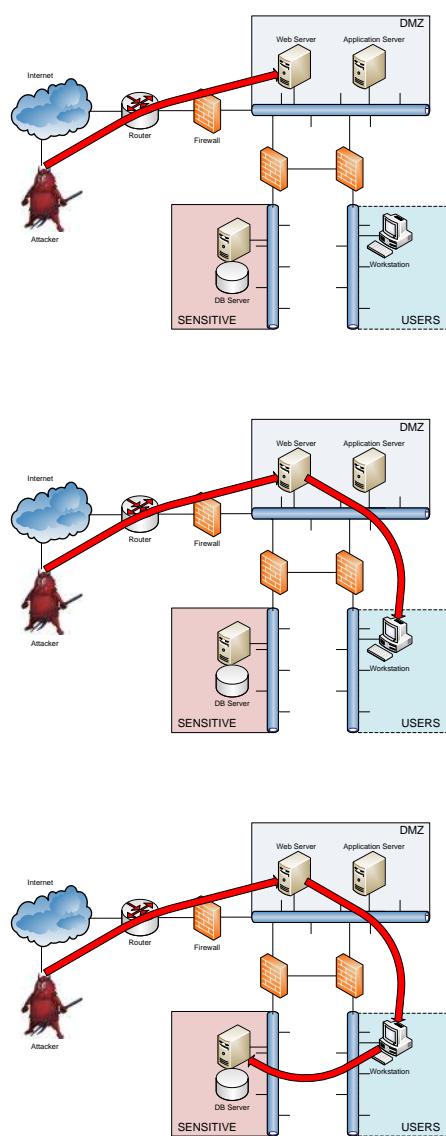
▷ **Input:** SAP-scale model of behavior of activities on Business Objects, process endpoint.

▷ **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.



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)

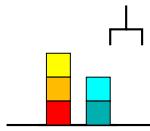
- ▷ **Intelligent:** Determines automatically how to solve a complex problem effectively! (**The ultimate goal, no?!**)
- ▷ **Efficiency loss:** Without any domain-specific knowledge about Chess, you don't beat Kasparov . . .
 - ▷ Trade-off between "automatic and general" vs. "manual work but effective".

Research Question: How to make fully automatic algorithms effective?



▷ ps. "Making Fully Automatic Algorithms Effective"

- ▷ **Example 16.1.1**



▷ 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 16.1.2** State spaces typically are huge even for simple problems.

In other words: Even solving "simple problems" automatically (without help from a human) requires a form of intelligence.

- ▷ With blind search, even the largest super-computer in the world won't scale beyond 20 blocks!



Algorithmic Problems in Planning

- ▷ **Definition 16.1.3** 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 sacrificing planning is *much* more effective in practice.
- ▷ **Definition 16.1.4** 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.



16.2 The History of Planning

[Planning History: In the Beginning . . .](#)

- ▷ **In the beginning: Man invented Robots:**
 - ▷ “Planning” as in “the making of plans by an autonomous robot”.
 - ▷ [Shakey the Robot](#) ([Full video here](#))

- ▷ **In a little more detail:**
 - ▷ [NS63] introduced [general problem solving](#).
 - ▷ *... not much happened (well not much we still speak of today) ...*
 - ▷ 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](#)”.



©: Michael Kohlhase

445



History of Planning Algorithms

- ▷ Compilation into Logics/Theorem Proving
 - ▷ **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
 - ▷ **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](#), ...



©: Michael Kohlhase

446



History of Planning Algorithms, ctd.

- ▷ GraphPlan
 - ▷ **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

- ▷ **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, ...



©: Michael Kohlhase

447



History of Planning Algorithms, ctd.

- ▷ Planning as Heuristic Search:
- ▷ **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, ...



©: Michael Kohlhase

448



The International Planning Competition (IPC)

- ▷ **Competition?** (<http://ipc.icaps-conference.org/>) Run competing planners on a set of benchmarks devised by the IPC organizers. Give awards to the most effective planners.
- ▷ 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014
- ▷ **PDDL** [McD+98; FL03; HE05; Ger+09]
- ▷ ≈ 50 domains, $\gg 1000$ instances, 74 (!!) planners in 2011
- ▷ Optimal track vs. satisfying track
- ▷ Various others: uncertainty, learning, ...



©: Michael Kohlhase

449



IPC 2000: Competitors

- ▷ **BlackBox**: Compilation to SAT [KS99].
- ▷ **HSP**: Heuristic search [BG01].
- ▷ **IPP**: GraphPlan variant [Koe+97].
- ▷ **STAN**: Heuristic search.

- ▷ **GRT**: Heuristic search.
- ▷ **Mips**: Heuristic search.
- ▷ **FF**: Heuristic search [HN01].
- ▷ ... (13 altogether)



©: Michael Kohlhase

450



IPC 2000: Benchmark Domains

- ▷ **Blocksworld**: Move around blocks on a table (yeah, I know).
- ▷ **Freecell**: The card game.
- ▷ **Logistics**: Transport packages using trucks and airplanes.
- ▷ **Miconic-ADL**: A complex elevator-control problem (see slide ??).
- ▷ **Schedule**: A simple scheduling problem where objects must be processed with various machines.

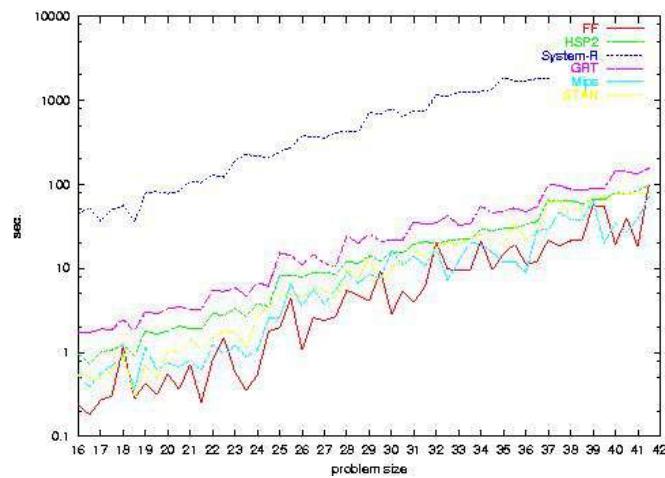


©: Michael Kohlhase

451



IPC'00 Results, Fully Automatic Track (Logistics)

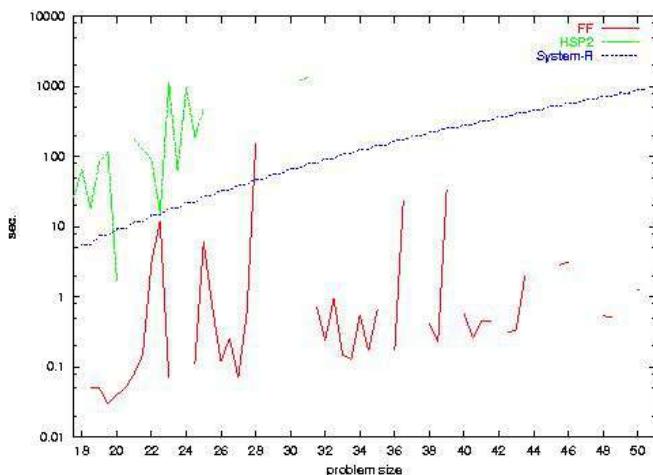


©: Michael Kohlhase

452



IPC'00 Results, Fully Automatic Track (Blocksworld)

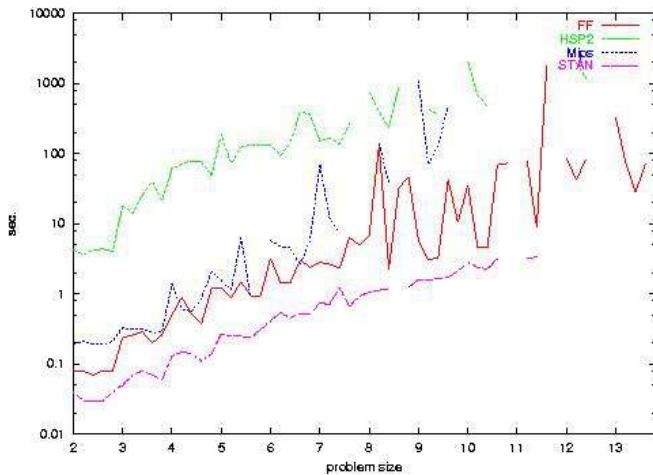


©: Michael Kohlhase

453



IPC'00 Results, Fully Automatic Track (FreeCell)

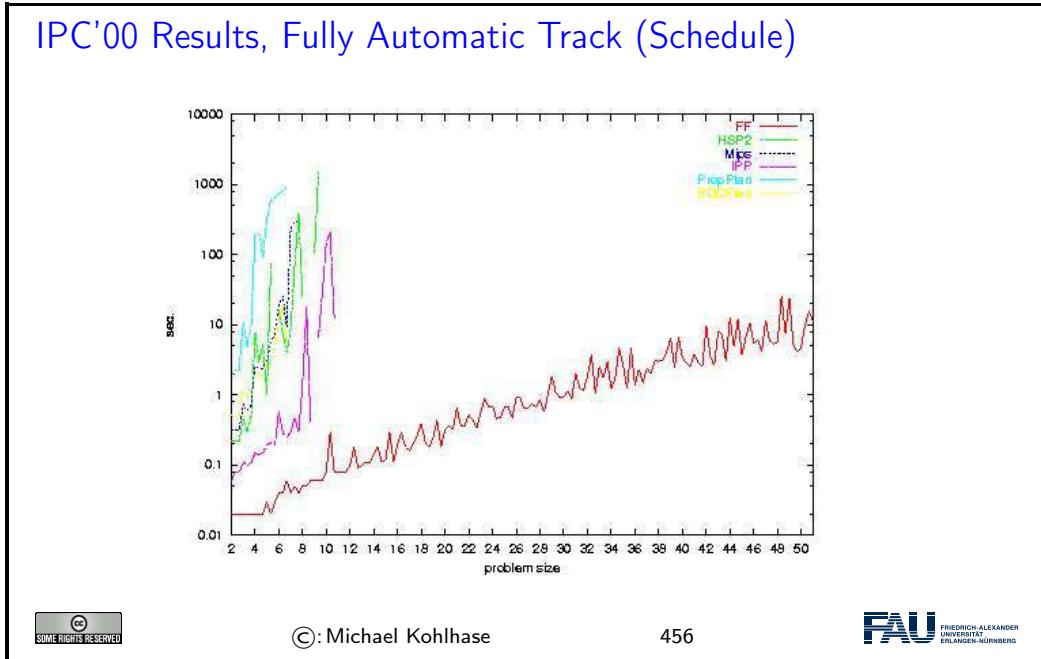
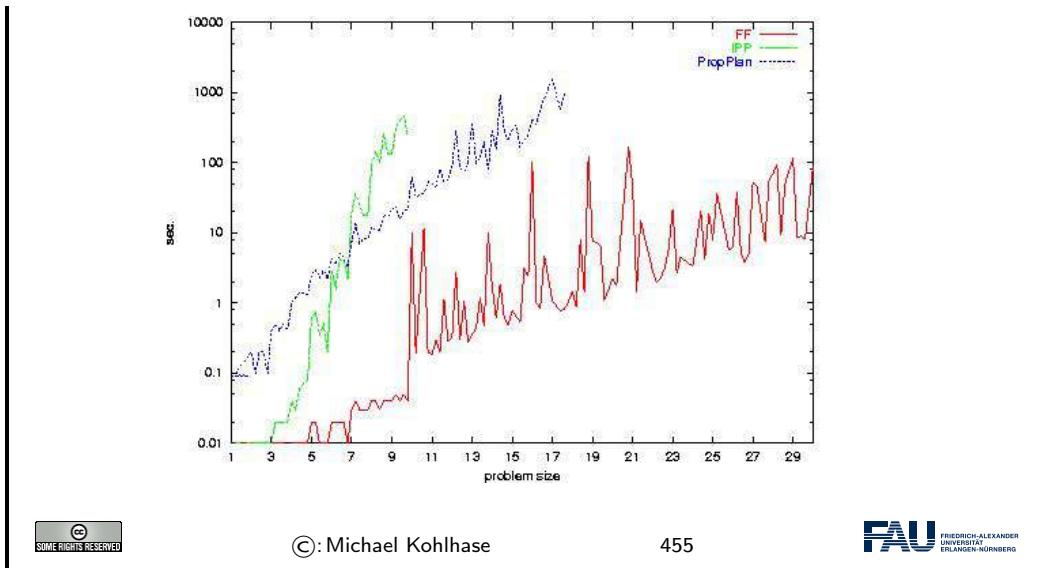


©: Michael Kohlhase

454



IPC'00 Results, Fully Automatic Track (Miconic)



And Since Then?

- ▷ Winners (what turned out to be successful)
 - ▷ **IPC 2000:** Winner FF, heuristic search.
 - ▷ **IPC 2002:** Winner LPG, heuristic search.
 - ▷ **IPC 2004:** Winner satisfying SGPlan, heuristic search; optimal SATPLAN, compilation to SAT.
 - ▷ **IPC 2006:** Winner satisfying SGPlan, heuristic search; optimal SATPLAN, compilation to SAT.

- ▷ **IPC 2008:** Winner satisfying LAMA, [heuristic search](#); optimal Gamer, symbolic search.
- ▷ **IPC 2011:** Winner satisfying LAMA, [heuristic search](#); optimal Fast-Downward, [heuristic search](#).
- ▷ **IPC 2014:** Winner satisfying Mercury, [heuristic search](#); optimal Symba, symbolic search.
- ▷ This is a VERY short summary of the history of the IPC! There are many different categories, and many different awards.
- ▷ For the rest of this chapter, we focus on planning as heuristic search.

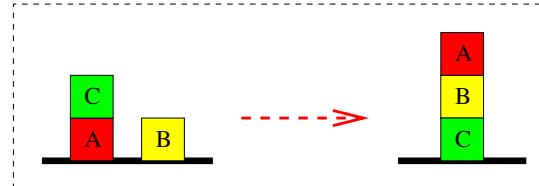


Questionnaire

- ▷ **Question:** If planners x and y compete in IPC'YY, and x wins, is x “better than” y ?
- ▷ **Answer:** Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a “winner”! On other domains and/or according to other criteria, you may well be better off with the “loser”.
- ▷ **Generally:** Assessing AI System suitability is complicated, over-simplification is dangerous. ([But, of course, nevertheless is being done all the time](#))



Planning History, p.s.



“The Sussman Anomaly”



16.3 The STRIPS Planning Formalism

“STRIPS” Planning

- ▷ **Definition 16.3.1** **STRIPS** = Stanford Research Institute Problem Solver.

STRIPS is the simplest possible (reasonably expressive) logics-based planning language.

- ▷ STRIPS has only **Boolean variables**: propositional logic atoms.
- ▷ Its preconditions/effects/goals are as canonical as imaginable:
 - ▷ Preconditions, goals: **conjunctions** of **positive atoms**.
 - ▷ Effects: **conjunctions of literals** (positive or negated atoms).
- ▷ We use the common special-case notation for this simple formalism.
- ▷ I'll outline some extensions beyond STRIPS later on, when we discuss PDDL.
- ▷ **Historical note:** STRIPS [FN71] was originally a planner (cf. Shakey), whose language actually wasn't quite that simple.



STRIPS Planning: Syntax

- ▷ **Definition 16.3.2** A **STRIPS planning task**, short **planning 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. Chapter 7)



“TSP” in Australia

- ▷ **Example 16.3.3 (Salesman Travelling in Australia)**



Strictly speaking, this is not actually a **TSP** problem instance; simplified/adapted for illustration.



STRIPS Encoding of “TSP”

▷ Example 16.3.4 (continuing)



- ▷ **Facts *P*:** $\{\text{at}(x), \text{vis}(x) \mid x \in \{\text{Sy}, \text{Ad}, \text{Br}, \text{Pe}, \text{Da}\}\}$.
- ▷ **Initial state *I*:** $\{\text{at}(\text{Sy}), \text{vis}(\text{Sy})\}$.
- ▷ **Goal *G*:** $\{\text{at}(\text{Sy})\} \cup \{\text{vis}(x) \mid x \in \{\text{Sy}, \text{Ad}, \text{Br}, \text{Pe}, \text{Da}\}\}$.
- ▷ **Actions $a \in A$:** $\text{drv}(x, y)$ where x and y have a road.
 Precondition pre_a : $\{\text{at}(x)\}$.
 Add list add_a : $\{\text{at}(y), \text{vis}(y)\}$.
 Delete list del_a : $\{\text{at}(x)\}$.
- ▷ **Plan:** $\langle \text{drv}(\text{Sy}, \text{Br}), \text{drv}(\text{Br}, \text{Sy}), \text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Ad}, \text{Pe}), \text{drv}(\text{Pe}, \text{Ad}), \text{drv}(\text{Ad}, \text{Da}), \text{drv}(\text{Da}, \text{Ad}), \text{drv}(\text{Ad}, \text{Sy}) \rangle$



STRIPS Planning: Semantics

▷ **Definition 16.3.5** Let $\Pi = \langle P, A, I, G \rangle$ be a STRIPS planning task. The **state space** of Π is $\Theta_\Pi = (S, A, T, I, S^G)$ where:

- ▷ The **states** (also **world states**) $S := \mathcal{P}(P)$ are the subsets of P .
- ▷ A is Π 's action set.
- ▷ The transitions are $T = \{s \xrightarrow{a} \text{apply}(s, a) \mid \text{pre}_a \subseteq s\}$.
 - 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 of Chapter 7, without a cost function. Solutions are defined as before (paths from I to a state in S^G).



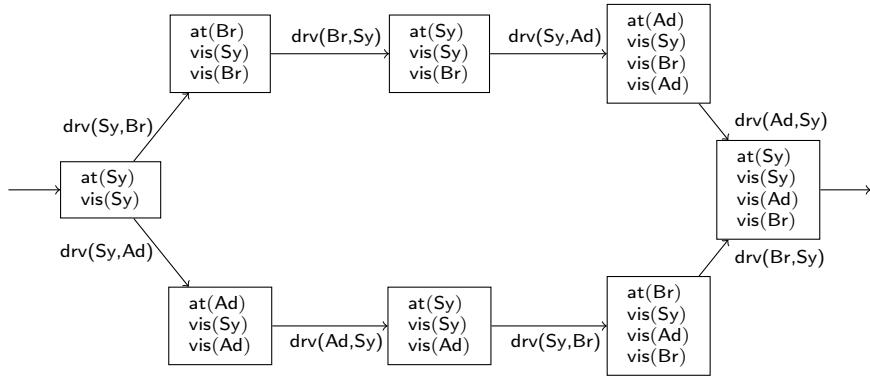
STRIPS Encoding of Simplified “TSP”



- ▷ **Facts P :** $\{\text{at}(x), \text{vis}(x) \mid x \in \{\text{Sy}, \text{Ad}, \text{Br}\}\}$.
- ▷ **Initial state I :** $\{\text{at}(\text{Sy}), \text{vis}(\text{Sy})\}$.
- ▷ **Goal G :** $\{\text{vis}(x) \mid x \in \{\text{Sy}, \text{Ad}, \text{Br}\}\}$. (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 :** $\{\text{at}(x)\}$.



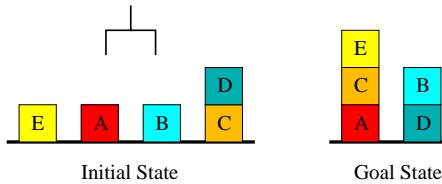
Encoding of Simplified “TSP”: State Space



- ▷ **Question:** Is this actually the state space?
- ▷ **Answer:** No, only the reachable part. E.g., Θ_{Π} also includes the states $\{\text{vis}(Sy)\}$ and $\{\text{at}(Sy), \text{at}(Br)\}$.



(Oh no it's) The Blocksworld



- ▷ **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:** $\{\text{on}(E, C), \text{on}(C, A), \text{on}(B, D)\}$.
- ▷ **Actions:** $\text{stack}(x, y)$, $\text{unstack}(x, y)$, $\text{putdown}(x)$, $\text{pickup}(x)$.

▷ **stack(x, y)?**

```

pre : {holding(x), clear(y)}
add : {on(x, y), armEmpty}
del : {holding(x), clear(y)}.
  
```



Questionnaire

- ▷ **Question:** Which are correct encodings (ones that are part of **some** correct overall model) of the STRIPS $\text{pickup}(x)$ action schema?

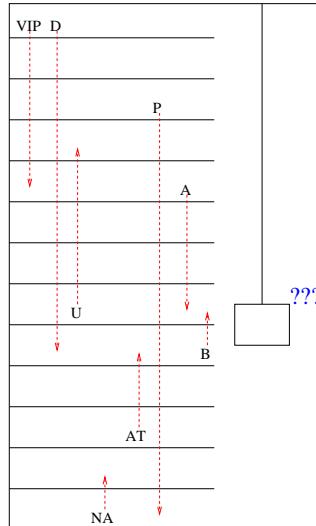
- | | |
|--|--|
| (A) {onTable(x), clear(x), armEmpty}
{holding(x)}
{onTable(x)} | (B) {onTable(x), clear(x), armEmpty}
{holding(x)}
{armEmpty} |
| (C) {onTable(x), clear(x), armEmpty}
{holding(x)}
{onTable(x), armEmpty, clear(x)} | (D) {onTable(x), clear(x), armEmpty}
{holding(x)}
{onTable(x), armEmpty} |

Recall: an actions a represented by a tuple $\langle \text{pre}_a, \text{add}_a, \text{del}_a \rangle$ of lists of facts.

- ▷ **Hint:** The only differences between them are the delete lists
- ▷ **Answer (A):** No, must delete armEmpty
- ▷ **Answer (B):** No, must delete onTable(x).
- ▷ **Answer (C) (D):** Both yes: We can, but don't have to, encode the *single-arm* Blocksworld so that the block currently in the hand is not clear.
For (C), stack(x, y) and putdown(x) need to add clear(x), so the encoding on the previous slide does not work.



Miconic-10: A Real-World Example



- ▷ 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 :-)



16.4 The PDDL Language

PDDL History

- ▷ PDDL $\hat{=}$ Planning Domain Description Language

- ▷ A description language for planning in the STRIPS formalism and various extensions.
- ▷ Used in the International Planning Competition (IPC).
- ▷ 1998: PDDL [McD+98].
- ▷ 2000: “PDDL subset for the 2000 competition” [Bac00].
- ▷ 2002: PDDL2.1, Levels 1-3 [FL03].
- ▷ 2004: PDDL2.2 [HE05].
- ▷ 2006: PDDL3 [Ger+09].

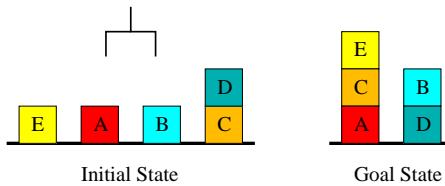


PDDL Quick Facts

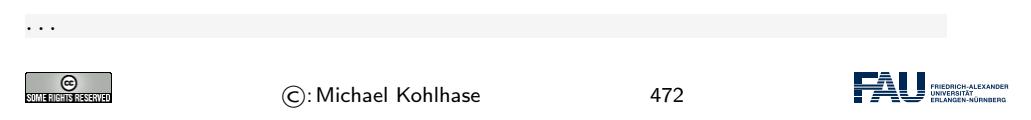
- ▷ PDDL is not a propositional language
 - ▷ Representation is lifted, using **object variables** to be instantiated from a finite set of **objects**. (Similar to predicate logic)
 - ▷ **Action schemas** parameterized by objects.
 - ▷ **Predicates** to be instantiated with objects.
- ▷ A PDDL planning task comes in two pieces
 - ▷ The **domain file** and the **problem file**.
 - ▷ The problem file gives the objects, the initial state, and the goal state.
 - ▷ The domain file gives the predicates and the operators; each benchmark domain has *one* domain file.



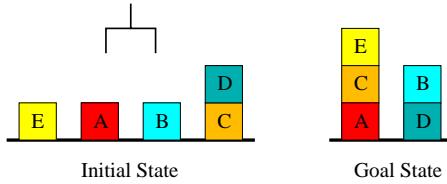
The Blocksworld in PDDL: Domain File



```
(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 blocksword)
  (: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))))))
)
```



Questionnaire

- ▷ **Question:** What is PDDL good for?
 - (A) Nothing.
 - (B) Free beer.
 - (C) Those AI planning guys.
 - (D) Being lazy at work.
- ▷ **Answer (A):** Nah, it's definitely good for *something* (see remaining answers)
- ▷ **Answer (B):** Generally, no. Sometimes, yes: PDDL is needed for the IPC, and if you win the IPC you get price money (= free beer).
- ▷ **Answer (C):** Yep. (Initially, every system had its own language, so running experiments felt a lot like "Lost in Translation".)
- ▷ **Answer (D):** Yep. You can be a busy bee, programming a solver yourself. Or you can be lazy and just write the PDDL. (I think I said that before ...)



©: Michael Kohlhase

475



16.5 Planning Complexity

Algorithmic Problems in Planning

- ▷ **Definition 16.5.1** 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 16.5.2** Programs solving these problems are called (optimal) **planner**, **planning system**, or **planning tool**.



©: Michael Kohlhase

476



Decision Problems in (STRIPS) Planning

- ▷ **Definition 16.5.3** By **PlanEx**, we denote the problem of deciding, given a STRIPS planning task Π , whether or not there exists a plan for Π .

- ▷ **PlanEx** corresponds to satisfying planning.
- ▷ **Definition 16.5.4** By **PlanLen**, we denote the problem of deciding, given a STRIPS planning 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 16.5.5** By **PolyPlanLen**, we denote the problem of deciding, given a STRIPS planning 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 16.5.6** *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 planning task. 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} .
 - P.1** The contents of the tape cells: $in(1, x_1), \dots, in(p(|w|), x_1), \dots, in(1, x_m), \dots, in(p(|w|), x_m)$.
 - P.2** The position of the R/W head: $at(1), \dots, at(p(|w|))$.
 - P.3** The internal state of the machine: $state(s_1), \dots, state(s_n)$.
 - P.4** Transitions rules \mapsto STRIPS actions; accepting state \mapsto STRIPS goal $\{state(s_{acc})\}$; initial state obvious.
 - P.5** This reduction to STRIPS runs in polynomial-time because we need only polynomially many facts. \square



Complexity of PlanEx, ctd. [Byl94]

- ▷ **Lemma 16.5.7** *PlanEx is in PSPACE* (*At most as hard as any other problem contained in PSPACE*)
- ▷ **Proof:** As $\text{PSPACE} = \text{NPSPACE}$ we show that **PlanEx** is in **NPSPACE**
 - P.1** 1. $s := I; l := 0$;
 - 2. Guess an applicable action a , compute the outcome state s' , set $l := l + 1$;

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 477). But, to decide **PlanEx**, we only need to remember its length.

▷ **Corollary 16.5.8 (Complexity of PlanEx)** **PlanEx** is **PSPACE**-complete.
(Immediate from previous two lemmta)



Complexity of **PlanLen** (after [Byl94])

- ▷ **PlanLen** isn't any easier than **PlanEx**
- ▷ **Corollary 16.5.9** **PlanLen** is **PSPACE**-complete.
- ▷ **Proof:**

P.1 (Membership) Same as before but failing at $l \geq B$.

P.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 16.5.10** **PolyPlanLen** is **NP**-complete.

- ▷ **Proof:**

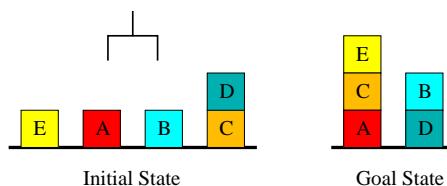
P.1 (Membership) Guess B actions and check whether they form a plan. This runs in polynomial time because B is polynomially bounded.

P.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!

The diagram illustrates the Blocksworld problem with two states: Initial State and Goal State.

Initial State:

			J				
			Z	X	H	L	
			X	N	I	D	
			H	F	C	P	
			N				
			F				
S	V	M	G	O	T	B	R
K	Y	A					
E	E	A					

Goal State:

			F	G	H	E	
			H			R	
			J	K	L	Y	
			Q	S	W	U	
Z	X	C	M	A	Q	I	
C	V	B	N	D	W	O	
B	S	D					
N							

Some Rights Reserved ©: Michael Kohlhase 482 FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Miconic-10: A Real-World Example

The diagram shows an elevator system with several passengers and specific constraints:

- Passenger **VIP** is served first.
- Passenger **D** can only go *down* when inside; similar for **U**.
- Passenger **NA** is never alone.
- Attendant **AT** is always present.
- Passengers **A** and **B** are never together in the same elevator (!).
- Passenger **P** is a normal passenger :-)

Some Rights Reserved ©: Michael Kohlhase 483 FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

16.6 Conclusion

Summary

- General problem solving attempts to develop solvers that perform well across a large class of problems.
- Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems. (Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.)

- ▷ Heuristic search planning has dominated the International Planning Competition (IPC). We focus on it here.
- ▷ STRIPS is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines actions in terms of precondition, add list, and delete list.
- ▷ PDDL is the de-facto standard language for describing planning problems.
- ▷ Plan existence (bounded or not) is **PSPACE**-complete to decide for STRIPS. If we bound plans polynomially, we get down to **NP**-completeness.



Reading:

- Chapters 10: *Classical Planning* and 11: *Planning and Acting in the Real World* in [RN09]

Content: Although the book is named “A Modern Approach”, the planning section was written long before the IPC was even dreamt of, before PDDL was conceived, and several years before heuristic search hit the scene. As such, what we have right now is the attempt of two outsiders trying in vain to catch up with the dramatic changes in planning since 1995.

Chapter 10 is Ok as a background read. Some issues are, imho, misrepresented, and it’s far from being an up-to-date account. But it’s Ok to get some additional intuitions in words different from my own.

Chapter 11 is useful in our context here because we don’t cover any of it. If you’re interested in extended/alternative planning paradigms, do read it.
- A good source for modern information (some of which we covered in the lecture) is Jörg *Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)* [Hof11] which is available online at <http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf>

Chapter 17

Planning II: Algorithms

17.1 Introduction

Reminder: Our Agenda for This Topic

- ▷ Chapter 16 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.



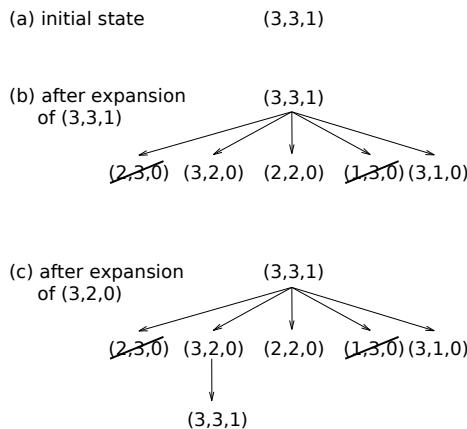
© Michael Kohlhase

485



Reminder: Search

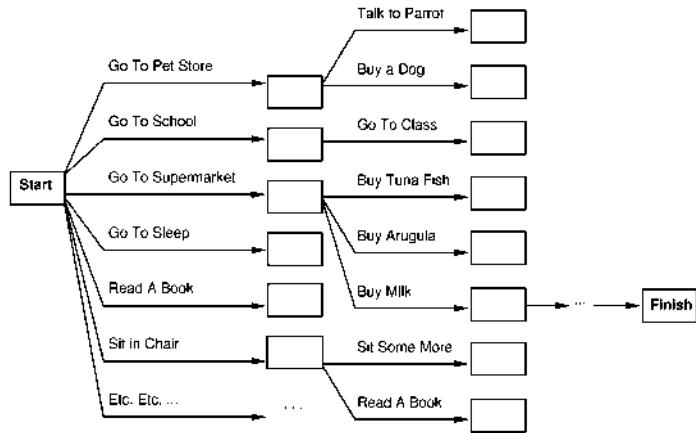
- Starting at initial state, produce all successor states step by step:



In planning, this is referred to as [forward search](#), or [forward state-space search](#).



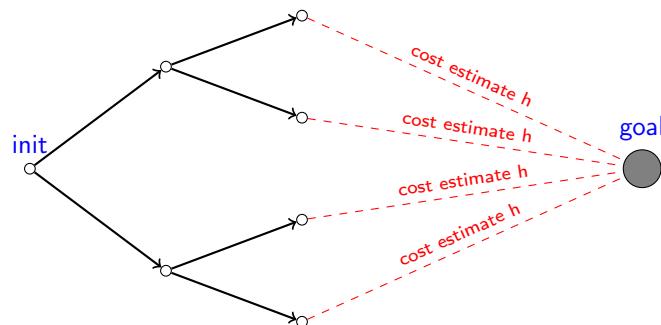
Search in the State Space?



▷ Use heuristic function to guide the search towards the goal!



Reminder: Informed Search



- ▷ Heuristic function h estimates the cost of an optimal path from a state s to the goal; search prefers to expand states s with small $h(s)$.

▷ Live Demo vs. Breadth-First Search:

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



Reminder: Heuristic Functions

- ▷ **Definition 17.1.1** Let Π be a planning 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 Chapter 7. Except, because we assume unit costs here, we use \mathbb{N} instead of \mathbb{R}^+ .
- ▷ **Definition 17.1.2** Let Π be a planning 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 Chapter 7, 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 planning 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^+ .



17.2 How to Relax in Planning

We will now instantiate our general knowledge about heuristic search to the planning domain. As always, the main problem is to find good heuristics. We will follow the intuitions of our discussion in Subsection 7.4.4 and consider full solutions to relaxed problems as a source for heuristics.

Reminder: Heuristic Functions from Relaxed Problems



▷ Problem II: Find a route from Saarbruecken To Edinburgh.



©: Michael Kohlhase

491



Reminder: Heuristic Functions from Relaxed Problems



▷ Relaxed Problem II': Throw away the map.

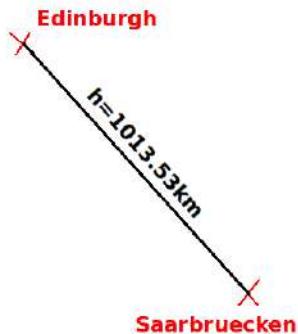


©: Michael Kohlhase

492



Reminder: Heuristic Functions from Relaxed Problems



▷ Heuristic function h : Straight line distance.



Relaxation in Route-Finding

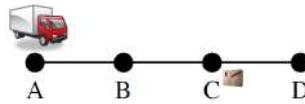


- ▷ Problem class \mathcal{P} : Route finding.
- ▷ Perfect heuristic h^*_P for \mathcal{P} : Length of a shortest route.
- ▷ Simpler problem class \mathcal{P}' : Route finding on an empty map.
- ▷ Perfect heuristic $h^*_{P'}$ for \mathcal{P}' : Straight-line distance.
- ▷ Transformation \mathcal{R} : Throw away the map.



How to Relax in Planning? (A Reminder!)

- ▷ Example 17.2.1 (Logistics)



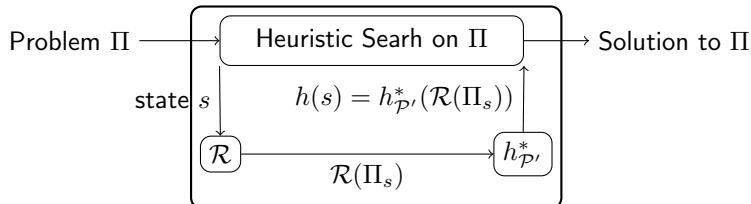
- ▷ **Facts P:** $\{\text{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\text{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.
- ▷ **Initial state I:** $\{\text{truck}(A), \text{pack}(C)\}$.
- ▷ **Goal G:** $\{\text{truck}(A), \text{pack}(D)\}$.
- ▷ **Actions A:** (Notated as “precondition \Rightarrow adds, \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 17.2.2 (“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 $\langle \text{unload}(D) \rangle$.



We will start with a very simple relaxation, which could be termed “positive thinking”: we do not consider preconditions of actions and leave out the delete lists as well.

How to Relax During Search: Overview

- ▷ **Attention:** Search uses the real (un-relaxed) Π . The relaxation is applied (e.g., in Only-Adds, the simplified actions are used) **only within the call to $h(s)$!!!**



- ▷ Here, Π_s is Π with initial state replaced by s , i.e., $\Pi = \langle P, A, I, G \rangle$ changed to $\langle P, A, s, G \rangle$: 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 $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $drXY, loX, ulX$.

Relaxed problem:

- ▷ State $s: AC$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 1: \langle ulD \rangle$.

Real problem:

- ▷ State $s: BC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $AC \xrightarrow{drAB} BC$.

Relaxed problem:

- ▷ State $s: BC$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 2: \langle drBA, ulD \rangle$.

Real problem:

- ▷ State $s: CC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $BC \xrightarrow{drBC} CC$.

Relaxed problem:

- ▷ State $s: CC$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 2: \langle drBA, ulD \rangle$.

Real problem:

- ▷ State $s: AC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $BC \xrightarrow{drBA} AC$.

Real problem:

- ▷ State $s: AC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ Duplicate state, prune.

Real problem:

- ▷ State $s: DC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $CC \xrightarrow{drCD} DC$.

Relaxed problem:

- ▷ State $s: DC$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 2: \langle drBA, ulD \rangle$.

Real problem:

- ▷ State $s: CT$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $CC \xrightarrow{loC} CT$.

Relaxed problem:

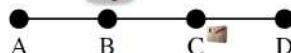
- ▷ State $s: CT$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 2: \langle drBA, ulD \rangle$.

Real problem:

- ▷ State $s: BC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $CC \xrightarrow{drCB} BC$.

**Relaxed problem:**

- ▷ State $s: CC$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 2: \langle drBA, ulD \rangle$.

**Real problem:**

- ▷ State $s: AC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $BC \xrightarrow{drBA} AC$.

Real problem:

- ▷ State $s: AC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ Duplicate state, prune.

Real problem:

- ▷ State $s: DC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $CC \xrightarrow{drCD} DC$.

Relaxed problem:

- ▷ State $s: DC$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 2: \langle drBA, ulD \rangle$.

Real problem:

- ▷ State $s: CT$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $CC \xrightarrow{loC} CT$.

Relaxed problem:

- ▷ State $s: CT$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 2: \langle drBA, ulD \rangle$.

Real problem:

- ▷ State $s: BC$; goal $G: AD$.
- ▷ Actions $A: \text{pre}, \text{add}, \text{del}$.
- ▷ $CC \xrightarrow{drCB} BC$.

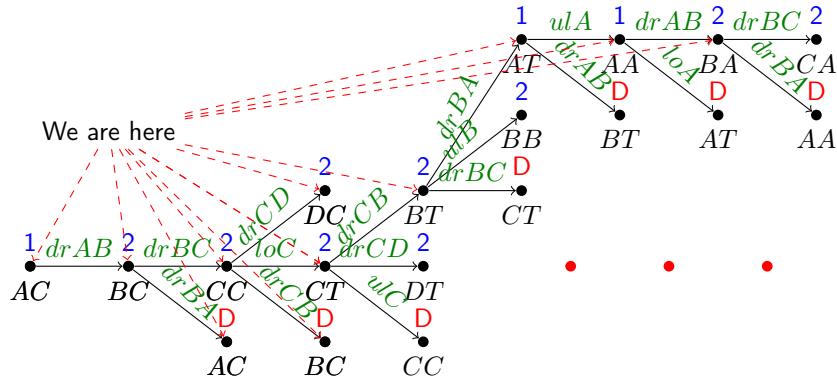
**Relaxed problem:**

- ▷ State $s: BC$; goal $G: AD$.
- ▷ Actions $A: \text{add}$.
- ▷ $h^R(s) = 2: \langle drBA, ulD \rangle$.

**Real problem:**

▷ Greedy best-first search:

(tie-breaking: alphabetic)



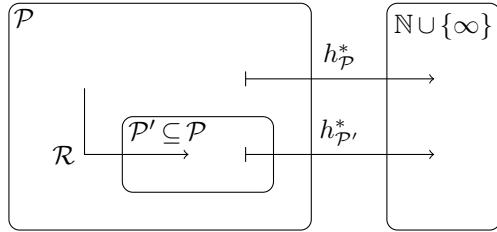
©: Michael Kohlhase

497



Only-Adds is a “Native” Relaxation

▷ Definition 17.2.3 (Native Relaxations) Confusing special case where $\mathcal{P}' \subseteq \mathcal{P}$.



- ▷ Problem class \mathcal{P} : STRIPS planning 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 planning tasks with empty preconditions and delete lists.
- ▷ Perfect heuristic for \mathcal{P}' : Shortest plan for only-adds STRIPS task.



©: Michael Kohlhase

498



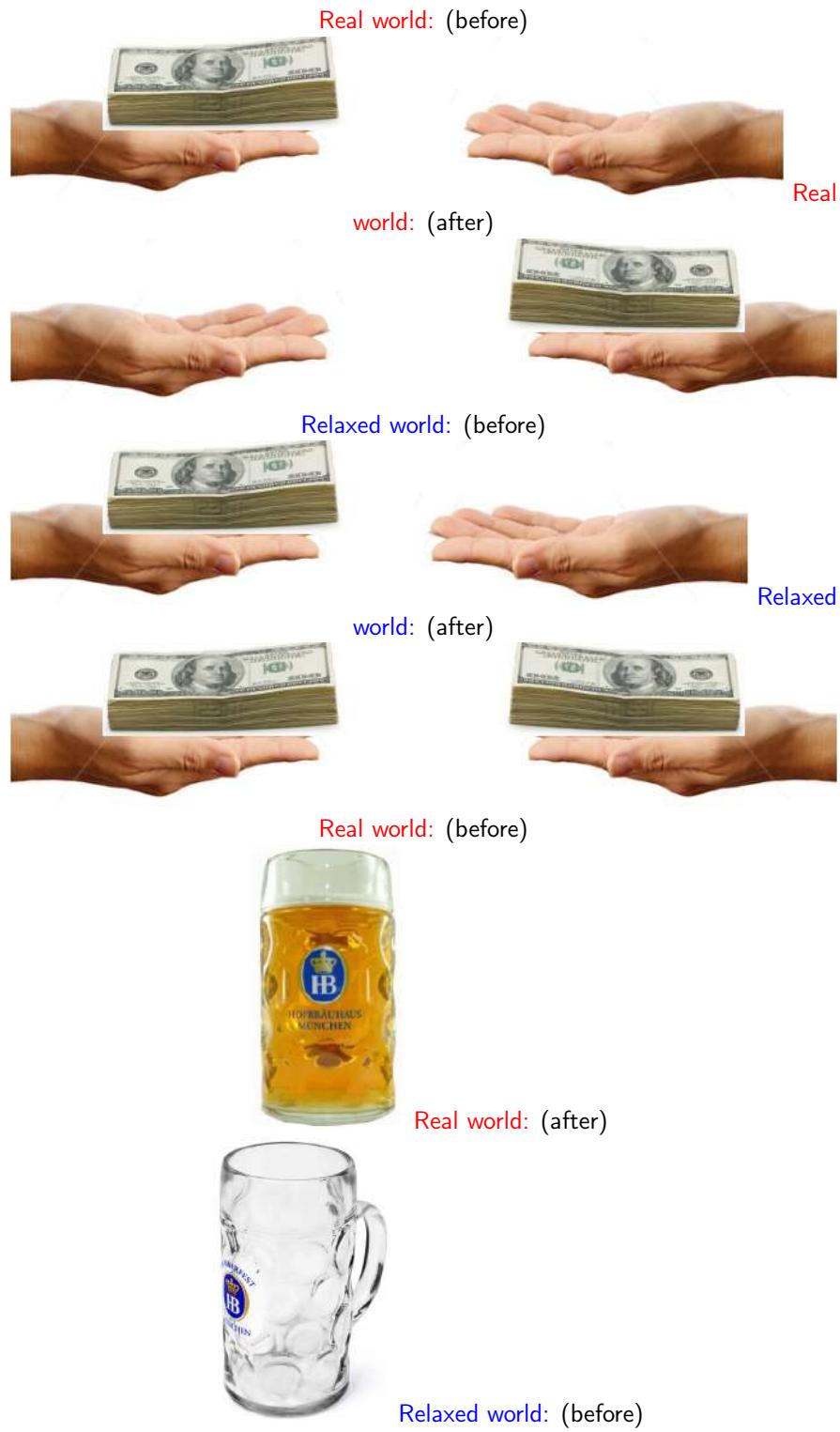
17.3 The Delete Relaxation

We turn to a more realistic relaxation, where we only disregard the delete list.

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)



Real world:



Relaxed world:



©: Michael Kohlhase

499

The Delete Relaxation

▷ **Definition 17.3.1 (Delete Relaxation)** Let $\Pi = \langle P, A, I, G \rangle$ be a planning task. The **delete-relaxation** of Π is the task $\Pi^+ = \langle P, A^+, I, G \rangle$ where $A^+ := \{a^+ \mid a \in A\}$ 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 planning tasks with empty delete lists, and the relaxation mapping \mathcal{R} drops the delete lists.
- ▷ **Definition 17.3.2 (Relaxed Plan)** Let $\Pi = \langle P, A, I, G \rangle$ be a planning 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.



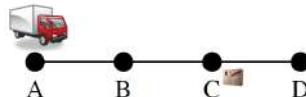
A Relaxed Plan for “TSP” in Australia



1. **Initial state:** $\{\text{at}(Sy), \text{vis}(Sy)\}$.
2. **Apply $\text{drv}(Sy, Br)^+$:** $\{\text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
3. **Apply $\text{drv}(Sy, Ad)^+$:** $\{\text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
4. **Apply $\text{drv}(Ad, Pe)^+$:** $\{\text{at}(Pe), \text{vis}(Pe), \text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.
5. **Apply $\text{drv}(Ad, Da)^+$:** $\{\text{at}(Da), \text{vis}(Da), \text{at}(Pe), \text{vis}(Pe), \text{at}(Ad), \text{vis}(Ad), \text{at}(Br), \text{vis}(Br), \text{at}(Sy), \text{vis}(Sy)\}$.



A Relaxed Plan for “Logistics”



- ▷ **Facts P :** $\{\text{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\text{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.
- ▷ **Initial state I :** $\{\text{truck}(A), \text{pack}(C)\}$.
- ▷ **Goal G :** $\{\text{truck}(A), \text{pack}(D)\}$.
- ▷ **Relaxed actions A^+** : (Notated as “precondition \Rightarrow adds”)
 - ▷ $\text{drive}(x, y)^+$: “ $\text{truck}(x) \Rightarrow \text{truck}(y)$ ”.
 - ▷ $\text{load}(x)^+$: “ $\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T)$ ”.

▷ $\text{unload}(x)^+$: “ $\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x)$ ”.

Relaxed plan:

$\langle \text{drive}(A, B)^+, \text{drive}(B, C)^+, \text{load}(C)^+, \text{drive}(C, D)^+, \text{unload}(D)^+ \rangle$

▷ We don't need to drive the truck back, because “it is still at A ”.



©: Michael Kohlhase

502



PlanEx⁺

▷ **Definition 17.3.3 (Relaxed Plan Existence Problem)** By **PlanEx⁺**, we denote the problem of deciding, given a planning 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!

▷ **Proposition 17.3.4 (PlanEx⁺ is Easy)** **PlanEx⁺** is a member of **P**.

▷ **Proof:** The following algorithm decides **PlanEx⁺**

P.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”

```

P.2 The algorithm terminates after at most $|P|$ iterations, and thus runs in polynomial time.

P.3 Correctness: See slide 506 □



©: Michael Kohlhase

503



Deciding PlanEx⁺ in “TSP” in Australia



Iterations on F :

1. {at(Sy), vis(Sy)}

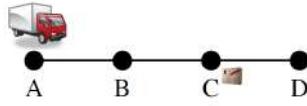
2. $\cup \{at(Ad), vis(Ad), at(Br), vis(Br)\}$
3. $\cup \{at(Da), vis(Da), at(Pe), vis(Pe)\}$



Deciding PlanEx⁺ in “Logistics”

▷ Example 17.3.5 (The solvable Case)

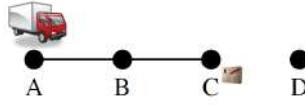
Iterations on F :



1. $\{truck(A), pack(C)\}$
2. $\cup \{truck(B)\}$
3. $\cup \{truck(C)\}$
4. $\cup \{truck(D), pack(T)\}$
5. $\cup \{pack(A), pack(B), pack(D)\}$

▷ Example 17.3.6 (The unsolvable Case)

Iterations on F :



1. $\{truck(A), pack(C)\}$
2. $\cup \{truck(B)\}$
3. $\cup \{truck(C)\}$
4. $\cup \{pack(T)\}$
5. $\cup \{pack(A), pack(B)\}$
6. $\cup \emptyset$



PlanEx⁺ Algorithm: Proof

Proof: To show: The algorithm returns “solvable” iff there is a relaxed plan for Π .

P.1 Denote by F_i the content of F after the i th iteration of the while-loop,

P.2 All $a \in A_0$ are applicable in I , all $a \in A_1$ are applicable in $apply(I, A_0^+)$, and so forth.

P.3 Thus $F_i = apply(I, \langle A_0^+, \dots, A_{i-1}^+ \rangle)$. (Within each A_j^+ , we can sequence the actions in any order.)

P.4.1 Direction “ \Rightarrow ”: If “solvable” is returned after iteration n then $G \subseteq F_n = apply(I, \langle A_0^+, \dots, A_{n-1}^+ \rangle)$ so $\langle A_0^+, \dots, A_{n-1}^+ \rangle$ can be sequenced to a relaxed plan which shows the claim. \square

P.4.2 Direction “ \Leftarrow ”:

P.4.2.1 Let $\langle a_0^+, \dots, a_{n-1}^+ \rangle$ be a relaxed plan, hence $G \subseteq \text{apply}(I, \langle a_0^+, \dots, a_{n-1}^+ \rangle)$.

P.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 .

P.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.

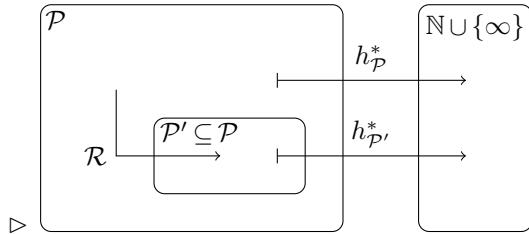
P.4.2.4 Assume to the contrary of the claim that, in an iteration $i < n$, (*) fires. Then $G \not\subseteq F$ and $F = F'$. But, with $F = F'$, $F = F_j$ for all $j > i$, and we get $G \not\subseteq F_n$ in contradiction. \square

\square



17.4 The h^+ Heuristic

Hold on a Sec – Where are we?



- ▷ \mathcal{P} : STRIPS planning tasks; $h_{\mathcal{P}}^*$: Length h^* of a shortest plan.
- ▷ $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS planning tasks with empty delete lists.
- ▷ \mathcal{R} : Drop the delete lists.
- ▷ Heuristic function: Length of a shortest *relaxed* plan ($h^* \circ \mathcal{R}$).
- ▷ **PlanEx**⁺ is not actually what we’re looking for. **PlanEx**⁺ $\hat{=}$ relaxed plan *existence*; we want relaxed plan *length* $h^* \circ \mathcal{R}$.



h^+ : The Ideal Delete Relaxation Heuristic

- ▷ **Definition 17.4.1 (Optimal Relaxed Plan)** Let $\Pi = \langle P, A, I, G \rangle$ be a planning 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 500, just adding the word “optimal”.
- ▷ **Here’s what we’re looking for:**

- ▷ **Definition 17.4.2** Let $\Pi = \langle P, A, I, G \rangle$ be a planning task with states S . The **ideal delete-relaxation heuristic** h^+ for Π is the function $h^+: S \rightarrow N \cup \{\infty\}$ where $h^+(s)$ is the length of an optimal relaxed plan for s if a relaxed plan for s exists, and $h^+(s) = \infty$ otherwise.
- ▷ In other words, $h^+ = h^* \circ \mathcal{R}$, cf. previous slide.



h^+ is Admissible

- ▷ **Lemma 17.4.3** Let $\Pi = \langle P, A, I, G \rangle$ be a planning 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 $\text{apply}(s, \langle a_1, \dots, a_i \rangle) \subseteq \text{apply}(s, \langle a_1^+, \dots, a_i^+ \rangle)$. \square
- ▷ If we ignore deletes, the states along the plan can only get bigger.
- ▷ **Theorem 17.4.4** h^+ is Admissible.
- ▷ **Proof:**
 - P.1** Let $\Pi = \langle P, A, I, G \rangle$ be a planning task with states S , and let $s \in S$.
 - P.2** $h^+(s)$ is defined as optimal plan length in $\langle P, A, s, G \rangle^+$.
 - P.3** With the lemma above, any plan for $\langle P, A, s, G \rangle$ also constitutes a plan for $\langle P, A, s, G \rangle^+$.
 - P.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. \square



How to Relax During Search: Ignoring Deletes

Real problem:

- ▷ Initial state I : AC ; goal G : AD .
- ▷ Actions A : pre, add, del.
- ▷ $drXY, loX, ulX$.

Relaxed problem:

- ▷ State s : AC ; goal G : AD .
- ▷ Actions A : pre, add.
- ▷ $h^+(s) = 5$: e.g. $\langle drAB, drBC, drCD, loC, ulD \rangle$.

Real problem:

- ▷ State s : BC ; goal G : AD .
- ▷ Actions A : pre, add, del.
- ▷ $AC \xrightarrow{drAB} BC$.

Relaxed problem:

- ▷ State s : BC ; goal G : AD .
- ▷ Actions A : pre, add.
- ▷ $h^+(s) = 5$: e.g. $\langle drBA, drBC, drCD, loC, ulD \rangle$.

Real problem:

- ▷ State s : CC ; goal G : AD .
- ▷ Actions A : pre, add, del.
- ▷ $BC \xrightarrow{drBC} CC$.

Relaxed problem:

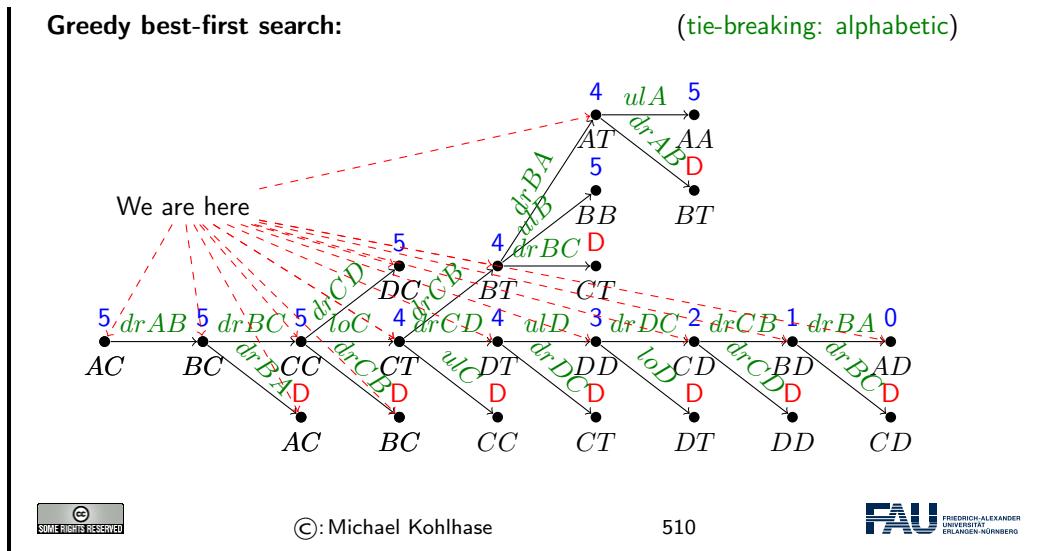
- ▷ State s : CC ; goal G : AD .
- ▷ Actions A : pre, add.
- ▷ $h^+(s) = 5$: e.g. $\langle drCB, drBA, drCD, loC, ulD \rangle$.

Real problem:

- ▷ State s : AC ; goal G : AD .
- ▷ Actions A : pre, add, del.
- ▷ $BC \xrightarrow{drBA} AC$.

Real problem:

- ▷ State s : AC ; goal G : AD .



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]



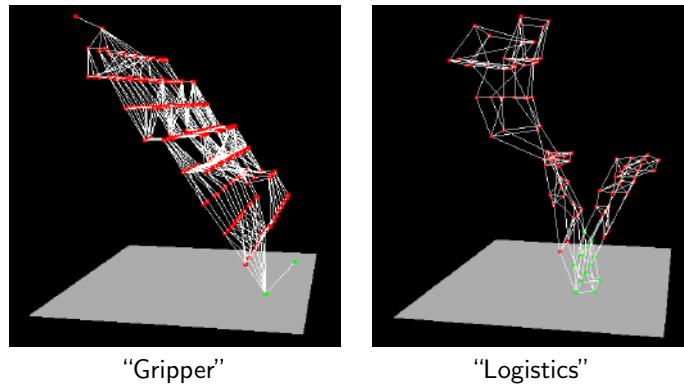
© Michael Kohlhase

511



A Brief Glimpse of h^+ Search Space Surfaces

- ▷ Graphs $\hat{=}$ state spaces, vertical height $\hat{=}$ h^+ :



On the side: In Russel/Norvig the text reads as if these illustrations referred to computing the heuristic, rather than to finding a plan.



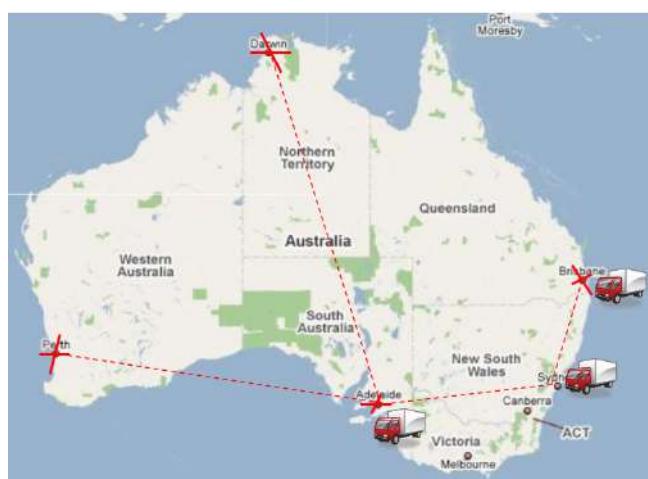
©: Michael Kohlhase

512



▷ h^+ in (the Real) TSP





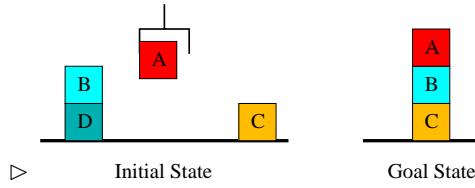
$h^+ \triangleq \text{Minimum Spanning Tree}$ SOME RIGHTS RESERVED

© Michael Kohlhase

513

Of course there are also bad cases. Here is one.

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 neighbors of these two states have a strictly higher h^+ value.



17.5 Planning Algorithms: Conclusion

Summary

- ▷ 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 planning tasks. A **relaxed plan** is a plan for such a relaxed task. $h^+(s)$ is the length of an optimal relaxed plan for state s . h^+ is **NP-hard** to compute.
- ▷ h^{FF} approximates h^+ by computing some, not necessarily optimal, relaxed plan. That is done by a forward pass (building a **relaxed planning graph**), followed by a backward pass (**extracting a relaxed plan**).



Topics We Didn't Cover Here

- ▷ **Abstractions, Landmarks, Critical-Path Heuristics, Cost Partitionings, 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 valuemaps 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:** ...



Reading (RN: Same As Previous Chapter):

- *Chapters 10: Classical Planning and 11: Planning and Acting in the Real World [RN09]*

Content: Although the book is named “A Modern Approach”, the planning section was written long before the IPC was even dreamt of, before PDDL was conceived, and several years before heuristic search hit the scene. As such, what we have right now is the attempt of two outsiders trying in vain to catch up with the dramatic changes in planning since 1995.

Chapter 10 is Ok as a background read. Some issues are, imho, misrepresented, and it’s far from being an up-to-date account. But it’s Ok to get some additional intuitions in words different from my own.

Chapter 11 is annoyingly named (I’ve seen lots of classical planning in the “real world”), but is useful in our context here because we don’t cover any of it. If you’re interested in extended/alternative planning paradigms, do read it.

Chapter 18

Planning and Acting in the Real World

Outline

- ▷ The real world (things go wrong)
- ▷ Agents and Belief States
- ▷ Conditional planning
- ▷ Monitoring and replanning



©: Michael Kohlhase

517



18.1 Introduction

The real world

- ▷ **Example 18.1.1** We have a flat tire – what to do?



START
~Flat(Spare) Intact(Spare) Off(Spare)
On(Tire1) Flat(Tire1)

On(x) ~Flat(x)

FINISH

Remove(x)
Off(x) ClearHub
On(x) ~ClearHub

Puton(x)
Off(x) ClearHub
On(x) ~ClearHub

Inflate(x)
Intact(x) Flat(x)
~Flat(x)



Things go wrong

- ▷ **Example 18.1.2 (Incomplete information)**
 - ▷ Unknown preconditions, e.g., *Intact(Spare)?*
 - ▷ Disjunctive effects, e.g., *Inflate(x)* causes *Inflated(x) ∨ SlowHiss(x) ∨ Burst(x) ∨ BrokenPump ∨ ...*
- ▷ **Example 18.1.3 (Incorrect information)**
 - ▷ Current state incorrect, e.g., spare NOT intact
 - ▷ Missing/incorrect postconditions in operators
- ▷ **Definition 18.1.4** With the **qualification problem** in planning is that we can never finish listing all the required preconditions and possible conditional outcomes of actions



What can we do if things (can) go wrong?

- ▷ **One Solution:** **Sensorless planning:** devise a plan that works regardless of state or outcome.
- ▷ **Problem:** Such plans may not exist!
- ▷ **Another Solution:** **Conditional planning:**
 - ▷ Plan to obtain information, (observation actions)
 - ▷ Subplan for each contingency.
- ▷ **Example 18.1.5 (A conditional Plan)** (AAA ≡ ADAC)

[*Check(T1)*, if *Intact(T1)* then *Inflate(T1)* else *CallAAA*]
- ▷ **Problem:** Expensive because it plans for many unlikely cases
- ▷ **Still another Solution:** **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 18.1.6** *We really need a combination; plan for likely/serious eventualities, deal with others when they arise, as they must eventually.*



18.2 Agent Architectures based on Belief States

We are now ready to proceed environments which can only partially observe and where our actions are non-deterministic. Both sources of uncertainty conspire to allow us only partial knowledge about the world, so that we can only optimize “expected utility” instead of “actual utility” of our actions.

World Models for Uncertainty

- ▷ **Problem:** We do not know with certainty what state the world is in!
- ▷ **Idea:** Just keep track of all the possible states it could be in.
- ▷ **Definition 18.2.1** A stateful reflex 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**.



©: Michael Kohlhase

521



That is exactly what we have been doing in the last semester: we have been studying methods that build on descriptions of the “actual” world, and have been concentrating on the progression from atomic to factored and ultimately structured representations. Tellingly, we spoke of “world states” instead of “belief states”; we have now justified this practice in the brave new belief-based world models by the (re-) definition of “world states” above. To fortify our intuitions, let us recap the methods from last semester from a belief-state-model perspective.

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.



Let us now see what happens when we lift the restrictions of [total observability](#) and determinism.

World Models for Complex Environments

- ▷ In a fully observable, but stochastic environment,
 - ▷ the [belief state](#) must deal with a set of possible states
 - ▷ generalize the [transition function](#) to a [transition relation](#)
- ▷ **Note:** this even applies for online problem solving 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, [belief model](#) $\hat{=}$ Bayesian networks, inference $\hat{=}$ probabilistic inference.
- ▷ **Decision-Theoretic Agents:** In a partially observable, stochastic, [belief model](#) + transition model $\hat{=}$ Decision networks, inference $\hat{=}$ MEU.



18.3 Conformant Planning

Conformant/Sensorless Planning

- ▷ **Definition 18.3.1** [Conformant](#) or [sensorless planning](#) tries to find plans

that work without any sensing (not even the initial state)

▷ Example 18.3.2 (Sensorless Vacuum Cleaner World)



States	integer dirt and robot locations
Actions	<i>left, right, suck, noOp</i>
Goal test	<i>notdirty?</i>

▷ Observation 18.3.3 In a sensorless world we do not know the initial state. (or any state after)

▷ Observation 18.3.4 sensorless planning must search in the space of *belief states* (sets of possible actual states).

▷ Example 18.3.5 (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' 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 are 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 18.3.6 In belief-state space the problem is always fully observable!



Let us see if we can understand the options for $\mathcal{T}^b(a, S)$ a bit better. The first question is when we

want an action a to be applicable to a belief state $S \subseteq \mathcal{S}$, i.e. when should $\mathcal{T}^b(a, S)$ be non-empty. In the first case, a^b would be applicable iff a is applicable to some $s \in S$, in the second case if a is applicable to all $s \in S$. So we only want to choose the first case if actions are harmless.

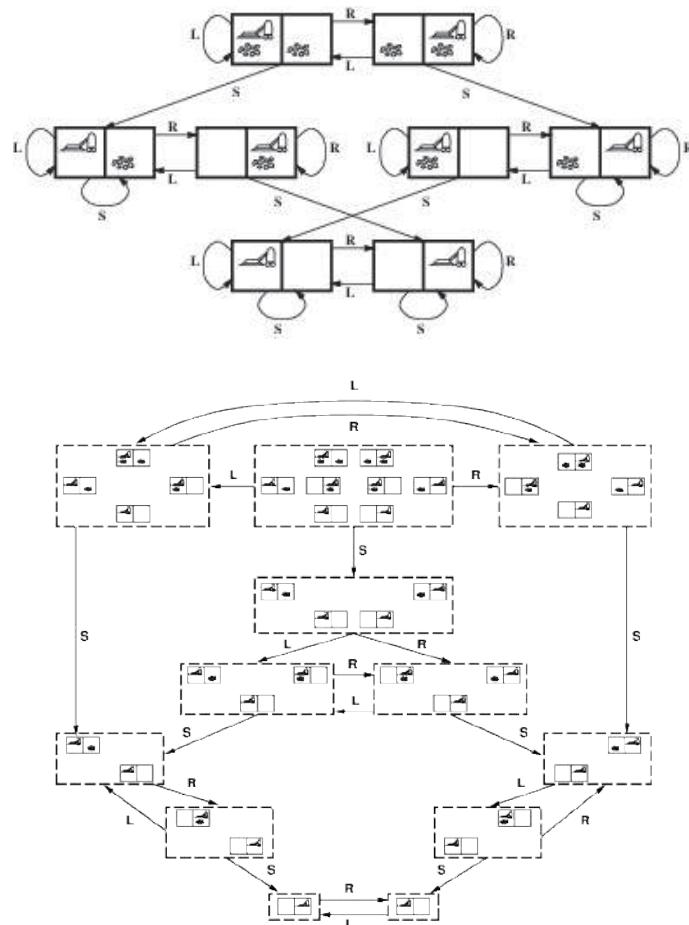
EdN:7

The second question we ask ourselves is what should be the results of applying a to $S \subseteq \mathcal{S}$? ⁷

State Space vs. Belief State Space

▷ Example 18.3.7 (State/Belief State Space in the Vacuum World)

In the vacuum world all actions are always applicable (1./2. equal)



©: Michael Kohlhase

527



Evaluating Conformant Planning

- ▷ 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.

⁷EDNOTE: MK: continue

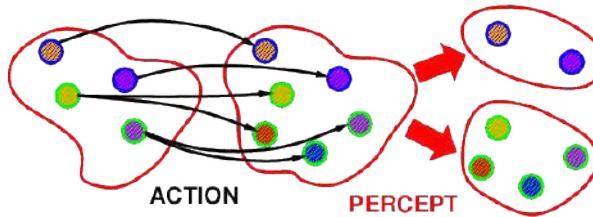
- ▷ belief state are **HUGE**; e.g. initial belief state for the 10×10 vacuum world contains $100 \cdot 2^{100} \approx 10^{32}$ physical states \leadsto use compact description
- ▷ **all** for initial state or **not leftmost column** after *Left*.



18.4 Conditional Planning

Conditional planning

- ▷ If the world is nondeterministic or partially observable then percepts usually provide information, i.e., split up the belief state



Conditional planning contd.

- ▷ Conditional plans check (any consequence of KB +) percept
- ▷ $[\dots, \text{if } C \text{ then } Plan_A \text{ else } Plan_B, \dots]$
- ▷ Execution: check *C* against current KB, execute “then” or “else”
- ▷ Need **some plan** for **every possible percept**
- ▷ **Observation 18.4.1** (cf. game playing) **some response for every opponent move**
- ▷ **Observation 18.4.2** (cf. backward chaining) **some rule such that every premise satisfied**
- ▷ **Idea:** Use an AND–OR tree search (very similar to backward chaining algorithm)



Conditional And/Or Search (Algorithm)

- ▷ **Definition 18.4.3** An algorithm for or searching AND–OR graphs generated by nondeterministic environments.

```

function AND-OR-GRAFH-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]

```

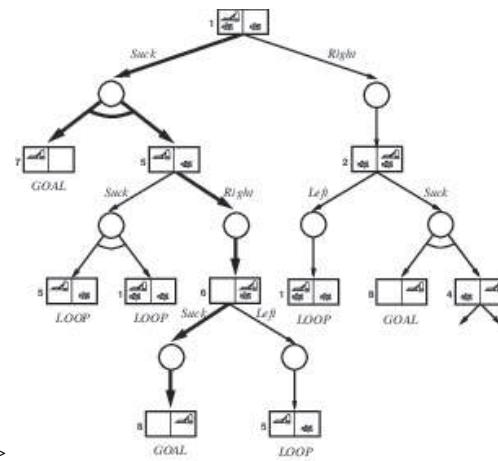
Cycle Handling: If a state has been seen before \rightsquigarrow **fail**

- ▷ ▷ does not mean *there is no solution*, but
- ▷ if there is a non-cyclic solution, then it is reachable by an earlier incarnation

The Erratic Vacuum Cleaner

- ▷ **Example 18.4.4 (The erratic vacuum world)** the Suck action works as follows:

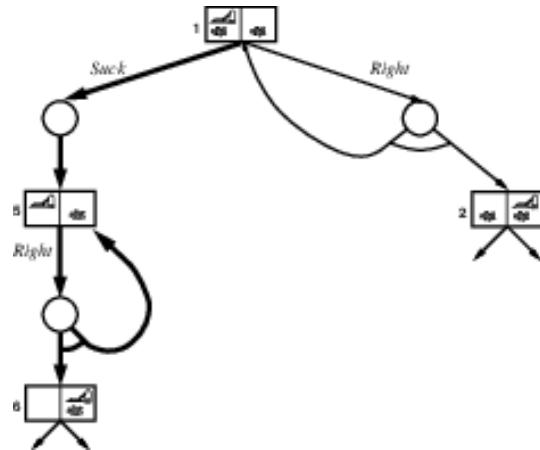
- ▷ in a dirty square clean the square and sometimes cleans up dirt in an adjacent square
- ▷ in a clean square the action sometimes deposits dirt on the carpet.



Solution: [Suck, if State = 5 then [Right, Suck] else []]

The Slippery Vacuum Cleaner (try, try, try, ... try again)

▷ Example 18.4.5 (The slippery vacuum world) Moving sometimes fails



Solution: $[L_1 : Left, \text{if } AtR \text{ then } L_1 \text{ else } [\text{if } CleanL \text{ then } \emptyset \text{ else } Suck]]$ or
 $[\text{while } AtR \text{ do } [Left], \text{if } CleanL \text{ then } \emptyset \text{ else } Suck]$

“Infinite loop” but will eventually work unless action always fails

▷



Chapter 19

Semester Change-Over

19.1 What did we learn in AI 1?

Topics of AI-1 (Winter Semester)

- ▷ Getting Started
 - ▷ What is Artificial Intelligence (situating ourselves)
 - ▷ Intelligent Agents (a unifying framework)
 - ▷ Logic Programming in Prolog (An influential paradigm)
- ▷ Problem Solving
 - ▷ Problem Solving and Search
 - ▷ Game playing (Adversarial Search)
 - ▷ Constraint Satisfaction Problems
- ▷ Knowledge and Reasoning
 - ▷ Formal Logic as the Mathematics of Meaning
 - ▷ Logic Programming
- ▷ Planning
 - ▷ Planning
 - ▷ Planning and Acting in the real world



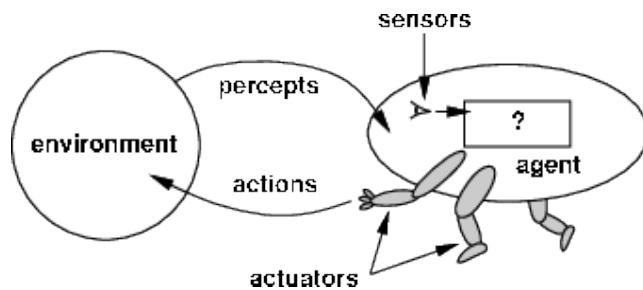
©: Michael Kohlhase

534

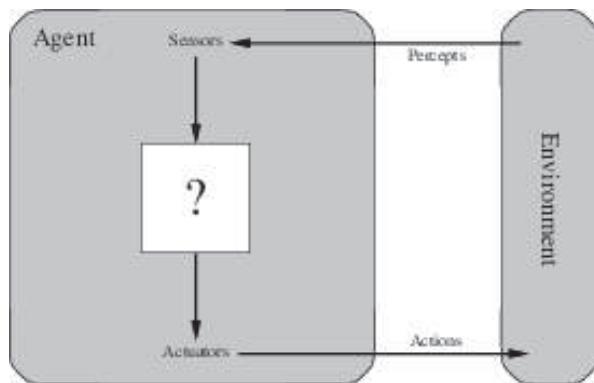


Rational Agents as an Evaluation Framework for AI

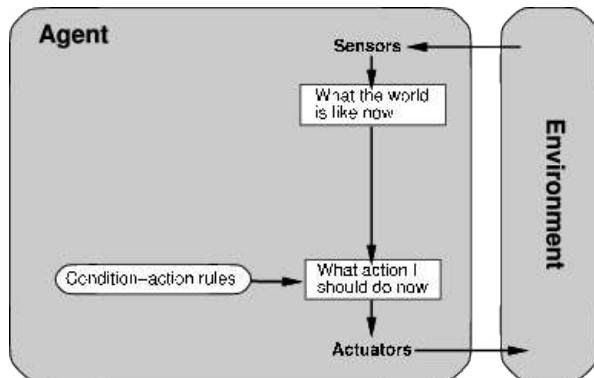
- ▷ Agents interact with the environment



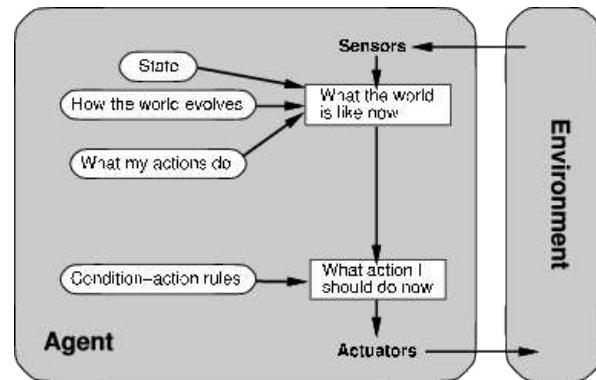
General agent schema



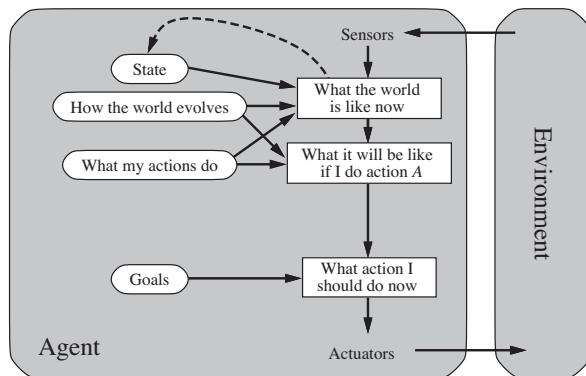
Simple Reflex Agents



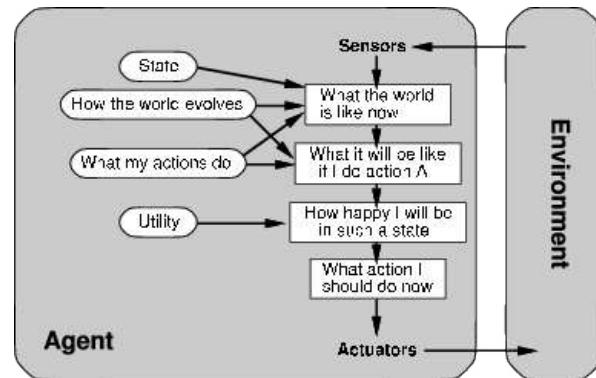
Reflex Agents with State



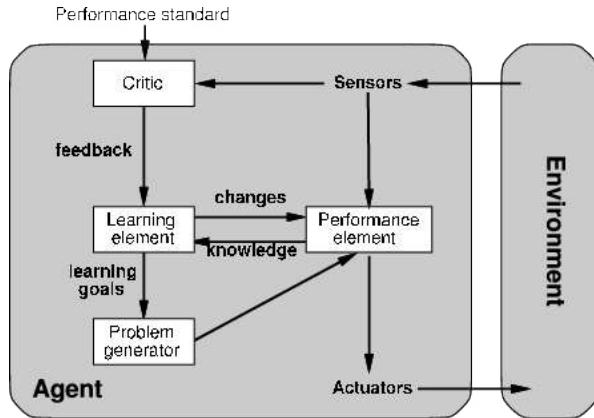
Goal-Based Agents



Utility-Based Agent



Learning Agents



Rational Agent

- ▷ **Idea:** Try to design agents that are successful (do the right thing)
- ▷ **Definition 19.1.1** 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 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 \sim exploration, learning, autonomy

Symbolic AI: Adding Knowledge to Algorithms

- ▷ Problem Solving (Black Box States, Transitions, Heuristics)
 - ▷ **Framework:** Problem Solving and Search (basic tree/graph walking)
 - ▷ **Variant:** Game playing (Adversarial Search) (Minimax+ $\alpha\beta$ Pruning)
- ▷ Constraint Satisfaction Problems (heuristic search over partial assignments)
 - ▷ States as partial variable assignments, transitions as assignment

- ▷ Heuristics informed by current restrictions, constraint graph
- ▷ Inference as constraint propagation (transferring possible values across arcs)
- ▷ Describing world states by formal language (and drawing inferences)
 - ▷ Propositional Logic and DPLL (deciding entailment efficiently)
 - ▷ First-Order Logic and ATP (reasoning about infinite domains)
 - ▷ **Digression:** Logic Programming (logic+search)
- ▷ 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



©: Michael Kohlhase

537



Topics of AI-2 (Summer Semester)

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



©: Michael Kohlhase

538



Artificial Intelligence (Künstliche Intelligenz) 2

Winter Semester 2019

– Lecture Notes –

Prof. Dr. Michael Kohlhase

Professur für Wissensrepräsentation und -verarbeitung
Informatik, FAU Erlangen-Nürnberg

Michael.Kohlhase@FAU.de

19.2 Administrativa

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

Prerequisites for AI-II

- ▷ the mandatory courses from Semester 1-4, in particular: **(or equivalent)**
- ▷ course “Mathematik C4” (InfMath4).
- ▷ (very) elementary complexity theory. **(big-Oh and friends)**
- ▷ AI-1 (“Künstliche Intelligenz I”) **(If you did not hear it, read the notes)**
- ▷ Motivation, Interest, Curiosity, hard work
- ▷ You can do this course if you want!



©: Michael Kohlhase

539



Now we come to a topic that is always interesting to the students: the grading scheme.

Assessment, Grades

- ▷ **Academic Assessment:** 90 minutes exam at the end of the semester **(~ Feb. 10.)**
- ▷ **Retake Exam:** 90 min exam at the end of the following semester **(~ July 15.)**
- ▷ **Mid-semester mini-exam:** online, optional, corrected but ungraded, **(so you can predict the exam style)**
- ▷ **Module Grade:**
 - ▷ Grade via the exam (Klausur) $\sim 100\%$ 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.



©: Michael Kohlhase

540



I basically do not have a choice in the grading scheme, as it is essentially the only one consistent with university policies. For instance, I would like to give you more incentives for the homework assignments – which would also mitigate the risk of having a bad day in the exam. Also, graded Wednesday quizzes would help you prepare for the lectures and thus let you get more out of them, but that is also impossible.

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](#)
 - ▷ please sign up for the AI-1 under <https://studon.fau.de/crs1865992.html>
 - ▷ Homeworks are handed in electronically (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.



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

Tutorials for Artificial Intelligence 1

- ▷ Weekly tutorials and homework assignments (first one in week two)
- ▷ **Instructor/Lead TA**: Dennis Müller (dennis.mueller@fau.de) Room: 11.138, Tel: 85-64053
- ▷ **Tutorials**: one each for Alpcan Dalga, Dennis Müller, Frederik Schaefer, and Max Rapp
- ▷ **Goal 1**: Reinforce what was taught in class (need)
- ▷ **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 it by having looked at the slides and the homework assignments
- ▷ **Inverted Classroom**: the latest craze in didactics (works well if done right)
in CS: Lecture + Homework assignments + Tutorials \cong Inverted Classroom



Do use the opportunity to discuss the AI-1 topics with others. After all, one of the non-trivial skills you want to learn in the course is how to talk about Artificial Intelligence topics. And that takes practice, practice, and practice.

Textbook, Handouts and Information, Forums



19.3 Overview over AI and Topics of AI-II

We restart the new semester by reminding ourselves of (the problems, methods, and issues of) Artificial Intelligence, and what has been achieved so far.

19.3.1 What is Artificial Intelligence?

The first question we have to ask ourselves is “what is Artificial Intelligence”, i.e. how can we define it. And already that poses a problem since the natural definition *like human intelligence, but artificially realized* presupposes a definition of Intelligence, which is equally problematic; even Psychologists and Philosophers – the subjects nominally “in charge” of human intelligence – have problems defining it, as witnessed by the plethora of theories e.g. found at [WHI].

What is Artificial Intelligence? Definition

▷ **Definition 19.3.1 (According to Wikipedia)**

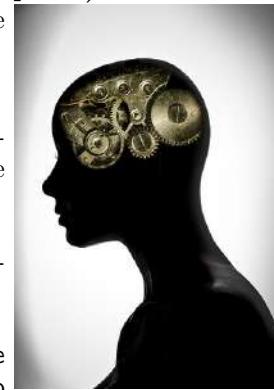
Artificial Intelligence (AI) is intelligence exhibited by machines

▷ **Definition 19.3.2 (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

▷ **Elaine Rich:** AI studies how we can make the computer do things that humans can still do better at the moment.

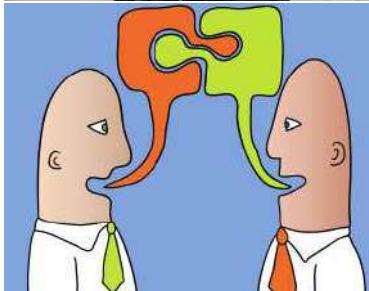


©: Michael Kohlhase

544

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

What is Artificial Intelligence? Components



19.3.2 Artificial Intelligence is here today!

The components of Artificial Intelligence are quite daunting, and none of them are fully understood,

much less achieved artificially. But for some tasks we can get by with much less. And indeed that is what the field of Artificial Intelligence does in practice – but keeps the lofty ideal around. This practice of “trying to achieve AI in selected and restricted domains” (cf. the discussion starting with slide 24) has borne rich fruits: systems that meet or exceed human capabilities in such areas. Such systems are in common use in many domains of application.

Artificial Intelligence is here today

- ▷ in outer space
 - ▷ in outer space systems need autonomous control:
 - ▷ remote control impossible due to time lag
- ▷ in artificial limbs
 - ▷ the user controls the prosthesis via existing nerves, can e.g. grip a sheet of paper.
- ▷ in household appliances
 - ▷ The iRobot Roomba vacuums, mops, and sweeps in corners,, parks, charges, and discharges.
 - ▷ general robotic household help is on the horizon.
- ▷ in hospitals
 - ▷ in the USA 90% of the prostate operations are carried out by RoboDoc
 - ▷ Paro is a cuddly robot that eases solitude in nursing homes.





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



- ▷ AlphaGo is a program developed 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.
- ▷ 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.



We will conclude this Subsection with a note of caution.

The AI Conundrum

- ▷ **Observation:** Reserving the term “Artificial Intelligence” has been quite a land-grab!
- ▷ **But:** researchers at the Dartmouth Conference (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 19.3.3** Functional/Logic Programming, Automated Theorem Proving, Planning, Machine Learning, Knowledge Representation, ...
- ▷ **Still Consequence:** AI research was alternately flooded with money and cut off brutally.



19.3.3 Two Ways to Attack the AI Problem

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

There are currently two main avenues of attack to the problem of building artificially intelligent systems. The (historically) first is based on the symbolic representation of knowledge about the world and uses inference-based methods to derive new knowledge on which to base action decisions. The second uses statistical methods to deal with uncertainty about the world state and learning methods to derive new (uncertain) world assumptions to act on.

Two ways of reaching Artificial Intelligence?

- ▷ Two avenues of attack for the problem: knowledge-based and statistical techniques
(they are complementary)

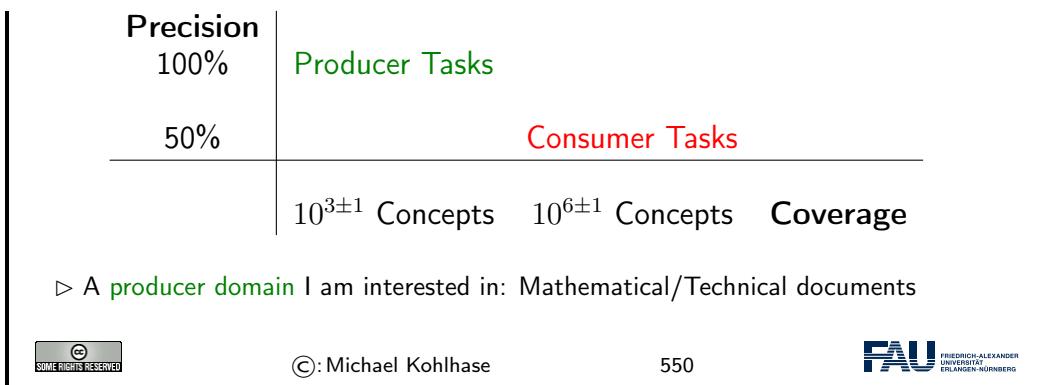
Deep	Knowledge-based AI-1	Not there yet <i>cooperation?</i>
Shallow	no-one wants this	Statistical Methods AI-2
Analysis ↑ vs. Coverage →	Narrow	Wide

- ▷ We will cover foundational methods of deep/narrow processing in this semester and shallow/wide-coverage ones in the next.



Environmental Niches for both Approaches to AI

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



To get this out of the way ...



- ▷ AlphaGo = search + neural networks
 - ▷ we do search this semester and cover neural networks in KI-2.
 - ▷ I will explain AlphaGo a bit in Chapter 8.



©: Michael Kohlhase

551



19.3.4 AI in the KWARC Group

The KWARC Research Group

- ▷ **Observation:** The ability to represent knowledge about the world and to draw logical inferences is one of the central components of intelligent behavior.
- ▷ **Thus:** reasoning components of some form are at the heart of many AI systems.
- ▷ **KWARC Angle:** Scaling up (web-coverage) without dumbing down (too much)
 - ▷ Content markup instead of full formalization (too tedious)
 - ▷ User support and quality control instead of "The Truth" (elusive anyway)
 - ▷ use Mathematics as a test tube (Δ Mathematics \cong Anything Formal Δ)
 - ▷ 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



©: Michael Kohlhase

552



Overview: KWARC Research and Projects

Applications: eMath 3.0, Active Documents, Semantic Spreadsheets, Semantic 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
- ▷ STEX: Semantic LATEX
- ▷ invasive editors
- ▷ Context-Aware IDEs
- ▷ Mathematical Corpora
- ▷ Linguistics of Math
- ▷ ML for Math Semantics Extraction

Foundations: Computational Logic, Web Technologies, *OMDoc/MMT*



©: Michael Kohlhase

553



Research Topics in the KWARC Group

- ▷ We are always looking for bright, motivated KWARCies
- ▷ We have topics in for all levels (Enthusiast, Bachelor, Master, Ph.D.)
- ▷ List of current topics: <https://gl.kwarc.info/kwarc/thesis-projects/>
 - ▷ Automated Reasoning: Maths Representation in the Large
 - ▷ Logics development, (Meta)ⁿ-Frameworks
 - ▷ Math Corpus Linguistics: Semantics Extraction
 - ▷ Serious Games, Cognitive Engineering, Math Information Retrieval
- ▷ 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)



©: Michael Kohlhase

554



19.3.5 AI-II: Advanced Rational Agents

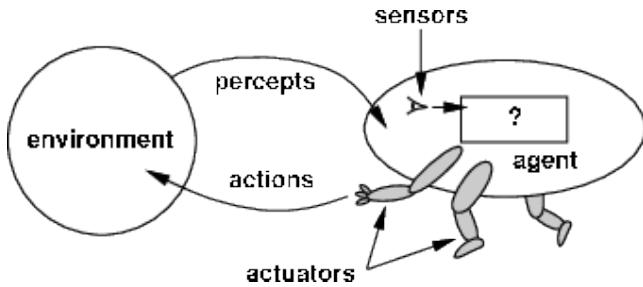
Remember the conceptual framework we gave ourselves in Chapter 6: we posited that all (artificial and natural) intelligence is situated in an agent that interacts with a given environment, and

postulated that what we experience as “intelligence” in a (natural or artificial) agent can be ascribed to the agent behaving rationally, i.e. optimizing the expected utility of its actions given the (current) environment.

Agents and Environments

▷ **Definition 19.3.4** 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 19.3.5** Agents include humans, robots, softbots, thermostats, etc.



In the last semester we restricted ourselves to fully observable, deterministic, episodic environments, where optimizing utility is easy in principle – but may still be computationally intractable, since we have full information about the world

Artificial Intelligence II Overview

- ▷ 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)



This semester we want to alleviate all these restrictions and study rationality in more realistic circumstances, i.e. environments which need only be partially observe and where our actions can be non-deterministic. Both of these extensions conspire to allow us only partial knowledge about the world, so that we can only optimize “expected utility” instead of “actual utility” of our actions. This directly leads to the first topic.

The second topic is motivated by the fact that environments can change and are initially

unknown, and therefore the agent must obtain and/or update parameters like utilities and world knowledge by observing the environment.

Topics of AI-2 (Summer Semester)

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



The last topic (which we will only attack if we have time) is motivated by [multi-agent](#) environments, where multiple agents have to collaborate for problem solving. Note that even though the adversarial search methods discussed in the game play chapter were essentially [single-agent](#) as both opponents optimized the utility of their actions alone.

In true [multi-agent](#) environments we have to also optimize collaboration between agents, and that is usually radically more efficient if agents can communicate.

Part V

Reasoning with Uncertain Knowledge

This part of the course notes addresses inference and agent decision making in partially observable environments, i.e. where we only know probabilities instead of certainties whether propositions are true/false. We cover basic probability theory and – based on that – Bayesian Networks and simple decision making in such environments. Finally we extend this to probabilistic temporal models and their decision theory.

Chapter 20

Quantifying Uncertainty

In this Chapter we develop a machinery for dealing with uncertainty: Instead of thinking about what we know to be true, we must think about what is likely to be true.

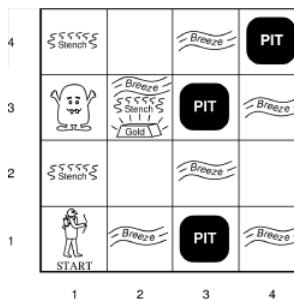
20.1 Dealing with Uncertainty: Probabilities

Before we go into the technical machinery in Subsection 20.1.5 ff., let us contemplate the sources of uncertainty our agents might have to deal with (Subsection 20.1.1) and how the agent models need to be extended to cope with that (Subsection 20.1.3).

20.1.1 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."

▷ **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?"



Unreliable Sensors

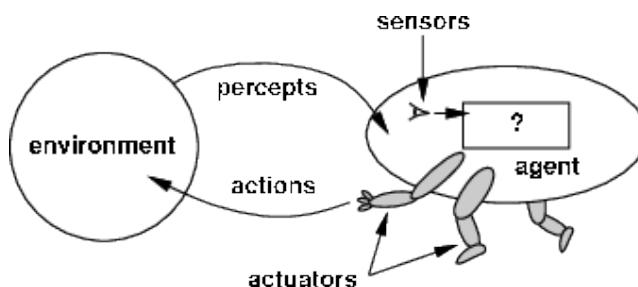
- ▷ **Robot Localization:** Suppose we want to support localization using landmarks to narrow down the area.
- ▷ **Example 20.1.1** *If you see the Eiffel tower, then you're in Paris.*
- ▷ **Difficulty:** Sensors can be imprecise.
 - ▷ Even if a landmark is perceived, we cannot conclude with certainty 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.



20.1.2 Recap: Rational Agents as a Conceptual Framework

Agents and Environments

- ▷ **Definition 20.1.2** 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).

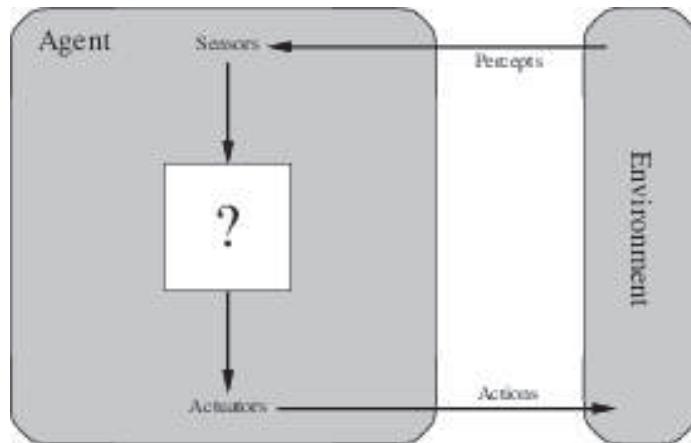


- ▷ **Example 20.1.3** Agents include humans, robots, softbots, thermostats, etc.



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.



Rationality

- ▷ **Idea:** Try to design agents that are successful (aka. "do the right thing")
- ▷ **Definition 20.1.4** A **performance measure** is a function that evaluates a sequence of environments.
- ▷ **Example 20.1.5** 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 20.1.6** An agent is called **rational**, if it chooses whichever action maximizes the expected value of the performance measure given the perceptual 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 \sim exploration, learning, autonomy

- ▷ **Definition 20.1.7** 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 20.1.8** 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 20.1.9 (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 20.1.10** *Agent design is largely determined by environment type.*

- ▷ **Problem:** There is a vast number of possible environments in AI.

- ▷ **Solution:** Classify along a handful of “dimensions” (**independent**

characteristics)

▷ **Definition 20.1.11** For an agent a we call an environment e

- ▷ **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**.
- ▷ **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**.
- ▷ **discrete**, iff the sets of e 's **states** and a 's actions are countable, else **continuous**.
- ▷ **single-agent**, iff only a acts on e (when must we count parts of e as agents?)



Some examples will help us understand this better.

Environment Types (Examples)

▷ **Example 20.1.12** Some environments classified:

	Solitaire	Backgammon	Internet shopping	Taxi
observable	Yes	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 20.1.13** The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, and **multi-agent** (worst case for AI)

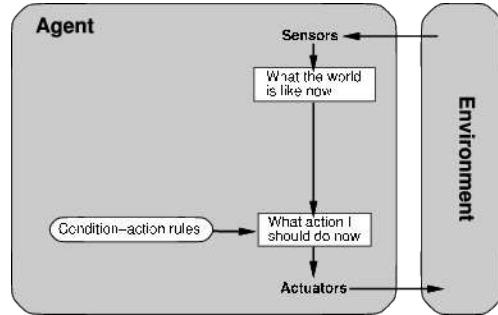


In the AI-1 course we will work our way from the simpler environment types to the more general ones. Each environment type will need its own agent types specialized to surviving and doing well in them.

Simple reflex agents

▷ **Definition 20.1.14** 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 20.1.15

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

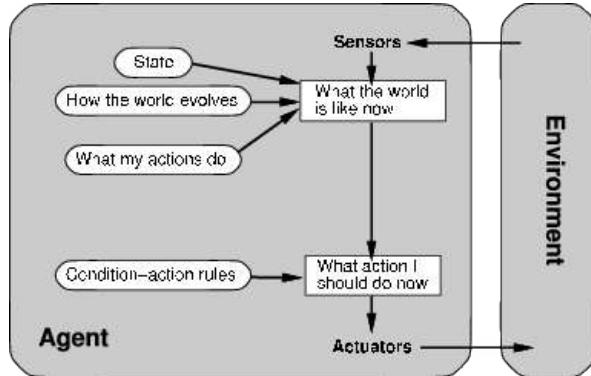


Reflex agents with state

▷ **Idea:** Keep track of the state of the world we cannot see now in an internal model

▷ **Definition 20.1.16** A **stateful reflex agent** (also called **reflex agent with state** or **model-based agent**) whose agent function depends on a model of the world (called the **world model**).

▷ Agent Schema:



20.1.3 Agent Architectures based on Belief States

We are now ready to proceed environments which can only partially observe and where our actions are non-deterministic. Both sources of uncertainty conspire to allow us only partial

knowledge about the world, so that we can only optimize “expected utility” instead of “actual utility” of our actions.

World Models for Uncertainty

- ▷ **Problem:** We do not know with certainty what state the world is in!
- ▷ **Idea:** Just keep track of all the possible states it could be in.
- ▷ **Definition 20.1.17** A stateful reflex 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**.



That is exactly what we have been doing in the last semester: we have been studying methods that build on descriptions of the “actual” world, and have been concentrating on the progression from atomic to factored and ultimately structured representations. Tellingly, we spoke of “world states” instead of “belief states”; we have now justified this practice in the brave new belief-based world models by the (re-) definition of “world states” above. To fortify our intuitions, let us recap the methods from last semester from a belief-state-model perspective.

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
 $\text{world state} \hat{=} \text{PL0}$, transition model $\hat{=} \text{Strips}$,
inference $\hat{=} \text{state/plan space search}$.



©: Michael Kohlhase

570



Let us now see what happens when we lift the restrictions of **total observability** and determinism.

World Models for Complex Environments

- ▷ In a fully observable, but stochastic environment,
 - ▷ the **belief state** must deal with a set of possible states
 - ▷ generalize the **transition function** to a **transition relation**
- ▷ **Note:** this even applies for online problem solving 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)



©: Michael Kohlhase

571



Preview: New World Models (Belief) \leadsto new Agent Types

- ▷ **Probabilistic Agents:** In a partially observable, **belief model** $\hat{=} \text{Bayesian networks}$, inference $\hat{=} \text{probabilistic inference}$.
- ▷ **Decision-Theoretic Agents:** In a partially observable, stochastic, **belief model** + transition model $\hat{=} \text{Decision networks}$, inference $\hat{=} \text{MEU}$.



©: Michael Kohlhase

572



20.1.4 Modeling Uncertainty

So we have extended the agent's world models to use sets of possible worlds instead of single (deterministic) world states. Let us evaluate whether this is enough for them to survive in the world.

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 20.1.18 (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)

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

▷ **Idea:** We need world models that estimate the pit-likelihood in cells!



Uncertainty and Logic

▷ **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" = "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?

$$\forall p. \text{Disease}(p, \text{cavity}) \Rightarrow \text{Symptom}(p, \text{toothache})$$

▷ **Is this rule correct?**

▷ No, not all cavities cause toothaches.

- ▷ Does this rule allow to deduce a cause from a symptom?
- ▷ 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.
- ▷ 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").



Beliefs and Probabilities

- ▷ What do we model with probabilities?
- ▷ Incomplete knowledge! We are not 100% sure, but we *believe to a certain degree* that something is true.
- ▷ Probability \approx Our degree of belief, given our current knowledge.
- ▷ Example 20.1.19 (Diagnosis)
 - ▷ $\text{Symptom}(p, \text{toothache}) \Rightarrow \text{Disease}(p, \text{cavity})$ 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 cavity, within the set of all p' that are indistinguishable from p based on our knowledge.
 - ▷ If we receive new knowledge (e.g., $\text{Disease}(p, \text{gingivitis})$), 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.
 - ▷ The process of estimating a probability P using statistics is called assessing P .
 - ▷ Assessing even a single P can require huge effort! (Eg. "The likelihood of making it to the university within 10 minutes")

- ▷ **What is probabilistic reasoning?** Deducing probabilities from knowledge about *other* probabilities.
- ▷ Probabilistic reasoning determines, based on probabilities that are (relatively) easy to assess, probabilities that are difficult to assess.



20.1.5 Acting under Uncertainty

Decision-Making Under Uncertainty

- ▷ **Example 20.1.20 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.
- ▷ **Better Example:** Which train to take to Frankfurt airport?



Uncertainty and Rational Decisions

- ▷ **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.

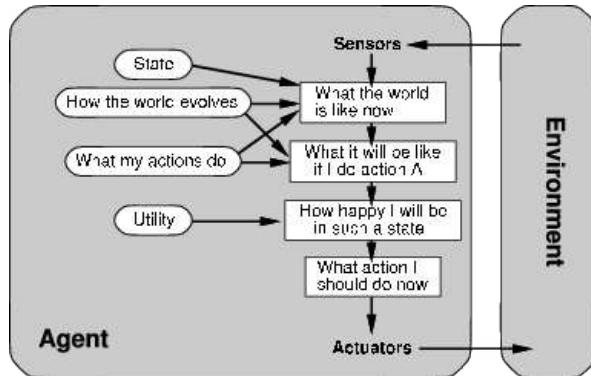


Utility-based agents

- ▷ **Definition 20.1.21** 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:**



©: Michael Kohlhase

580



Utility-based agents

- ▷ 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)



©: Michael Kohlhase

581



Decision-Theoretic Agent

- ▷ **A particular kind of utility-based agent:**

```

function DT-AGENT(percept) returns 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

```



©: Michael Kohlhase

582



20.1.6 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.
 - ▷ Chapter 21: 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.



©: Michael Kohlhase

583



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.



©: Michael Kohlhase

584



20.2 Unconditional Probabilities

Probabilistic Models

- ▷ **Definition 20.2.1** 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**: possible worlds cannot both be the case and **exhaustive**: one possible world must be the case.

- ▷ This determines the set of possible worlds
- ▷ **Example 20.2.2** If we roll two (distinguishable) dice with six sides, then we have 36 possible worlds: $(1, 1), (2, 1), \dots, (6, 6)$.
- ▷ We will restrict ourselves to a **discrete, countable sample space**. (others more complicated, less useful in AI)
- ▷ **Definition 20.2.3** A **probability model** $\langle \Omega, P \rangle$ consists of a 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 $\sum_{\omega \in \Omega} P(\omega) = 1$.



©: Michael Kohlhase

585



Unconditional Probabilities, Random Variables, and Events

- ▷ **Definition 20.2.4** 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 20.2.5** We will refer to the fact $X = x$ as an **outcome** and a set of outcomes as an **event**.
- ▷ The notation **uppercase "X"** for a variable, and **lowercase "x"** for one of its values will be used frequently. (Follows Russel/Norvig)
- ▷ **Definition 20.2.6** 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 20.2.7** $P(\text{Cavity} = \text{T}) = 0.2$, where Cavity is a random variable whose value is true iff some given person has a cavity.



©: Michael Kohlhase

586



Types of Random Variables

- ▷ **Note:** In general, random variables can have arbitrary domains. Here, we consider **finite-domain** random variables only, and **Boolean** random variables most of the time.
- ▷ **Example 20.2.8**

$$\begin{aligned} P(\text{Weather} = \text{sunny}) &= 0.7 \\ P(\text{Weather} = \text{rain}) &= 0.2 \\ P(\text{Weather} = \text{cloudy}) &= 0.08 \\ P(\text{Weather} = \text{snow}) &= 0.02 \\ P(\text{Headache} = \text{T}) &= 0.1 \end{aligned}$$

- ▷ 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 Boolean variable Name, we write name for Name = T and \neg name for Name = F . (Follows Russel/Norvig)



Probability Distributions

- ▷ **Definition 20.2.9** The probability distribution for a random variable X , written $\mathbf{P}(X)$, is the vector of probabilities for the (ordered) domain of X .
- ▷ **Example 20.2.10** Probability distributions for finite-domain and Boolean random variables

$$\mathbf{P}(\text{Headache}) = \langle 0.1, 0.9 \rangle$$

$$\mathbf{P}(\text{Weather}) = \langle 0.7, 0.2, 0.08, 0.02 \rangle$$

define the probability distribution for the random variables Headache and Weather.

- ▷ **Definition 20.2.11** Given a subset $\mathbf{Z} \subseteq \{X_1, \dots, X_n\}$ of random variables, an event is an assignment of values to the variables in \mathbf{Z} . The joint probability distribution, written $\mathbf{P}(\mathbf{Z})$, lists the probabilities of all events.

- ▷ **Example 20.2.12** $\mathbf{P}(\text{Headache}, \text{Weather})$ is

	$\text{Headache} = T$	$\text{Headache} = F$
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 20.2.13** Given random variables $\{X_1, \dots, X_n\}$, an atomic event is an assignment of values to all variables.
- ▷ **Example 20.2.14** 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 20.2.15** Given random variables $\{X_1, \dots, X_n\}$, the full joint probability distribution, denoted $\mathbf{P}(X_1, \dots, X_n)$, lists the probabilities of all atomic events.
- ▷ **Example 20.2.16** $\mathbf{P}(\text{Cavity}, \text{Toothache})$

	toothache	\neg toothache
cavity	0.12	0.08
\neg cavity	0.08	0.72

- ▷ All atomic events are disjoint (their pairwise conjunctions all are \perp); the sum of all fields is 1 (corresponds to their disjunction \top).



Probabilities of Propositional Formulas

- ▷ **Definition 20.2.17** Given random variables $\{X_1, \dots, X_n\}$, a **propositional formula**, short **proposition**, is a propositional formula 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

- (i) $P(\top) = 1$ and
- (ii) 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 20.2.18** $P(\text{cavity} \wedge \text{toothache}) = 0.12$ 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.



Questionnaire

- ▷ **Theorem 20.2.19 (Kolmogorow)** A function P that maps propositions into $[0, 1]$ is a probability measure if and only if

- i $P(\top) = 1$ and
- ii' for all propositions A, B : $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$.

- ▷ We can equivalently replace

- ii for all propositions A , $P(A) = \sum_{I \models A} P(I)$ (c.f. previous slide) with Kolmogorow's (ii').

1. **Question!:** Assume we have

- iii* $P(\perp) = 0$.

How to derive from (i), (ii'), and (iii) that, for all propositions A , $P(\neg a) = 1 - P(a)$?

- (a) By (i), $P(\top) = 1$; as $(a \vee \neg a) \Leftrightarrow \top$, we get $P(a \vee \neg a) = 1$.
- (b) By (iii), $P(\perp) = 0$; as $(a \wedge \neg a) \Leftrightarrow \perp$, we get $P(a \wedge \neg a) = 0$.
- (c) Inserting this into (ii'), we get $P(a \vee \neg a) = 1 = P(a) + P(\neg a) - 0$.



Questionnaire, ctd.

- ▷ **Reminder 1:** (i) $P(\top) = 1$; (ii') $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$.
 - ▷ **Reminder 2:** "Probabilities model our belief."
 - ▷ If P represents an objectively observable probability, the axioms clearly make sense. **But why should an agent respect these axioms, when modeling its subjective own belief?**
- Question:** Do you believe in Kolmogorow's axioms?
- ▷ You're free to believe whatever you want, but note this [DF31]: If an agent has a belief that violates Kolmogorov's axioms, then there exists a combination of "bets" on propositions so that the agent *always* loses money.
 - ▷ If your beliefs are contradictory, then you will not be successful in the long run (and even the next minute if your opponent is clever).



20.3 Conditional Probabilities

Conditional Probabilities: Intuition

- ▷ Do probabilities change as we gather new knowledge?
- ▷ Yes! Probabilities model our *belief*, thus they depend on our knowledge.
- ▷ **Example 20.3.1** Your "probability of missing the connection train" increases when you are informed that your current train has 30 minutes delay.
- ▷ **Example 20.3.2** 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 , $P(a \mid b)$ denotes the **conditional probability** of a (i.e., $A = \top$) given that **all we know** is b (i.e., $B = \top$).
- ▷ **Example 20.3.3** $P(\text{cavity}) = 0.2$ vs. $P(\text{cavity} \mid \text{toothache}) = 0.6$. And $P(\text{cavity} \mid \text{toothache} \wedge \neg \text{cavity}) = 0$



Conditional Probabilities: Definition

- ▷ **Definition 20.3.4** 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 20.3.5** $P(\text{cavity} \wedge \text{toothache}) = 0.12$ and $P(\text{toothache}) = 0.2$ yield $P(\text{cavity} | \text{toothache}) = 0.6$.



Conditional Probability Distributions

- ▷ **Definition 20.3.6** Given random variables X and Y , the **conditional probability distribution** of X given Y , written $\mathbf{P}(X | Y)$, is the table of all conditional probabilities of values of X given values of Y .
- ▷ For sets of variables: $\mathbf{P}(X_1, \dots, X_n | Y_1, \dots, Y_m)$.
- ▷ **Example 20.3.7** $\mathbf{P}(\text{Weather} | \text{Headache}) =$

	$\text{Headache} = \top$	$\text{Headache} = \perp$
$\text{Weather} = \text{sunny}$	$P(W = \text{sunny} \text{headache})$	$P(W = \text{sunny} \neg \text{headache})$
$\text{Weather} = \text{rain}$		
$\text{Weather} = \text{cloudy}$		
$\text{Weather} = \text{snow}$		

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)



20.4 Independence

Working with the Full Joint Probability Distribution

- ▷ **Example 20.4.1** Consider the following joint probability distribution:

	toothache	$\neg \text{toothache}$
cavity	0.12	0.08
$\neg \text{cavity}$	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 $P(\text{cavity} | \text{toothache})$?

$$\triangleright \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})}$$

▷ 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?

▷ **Answer:** No:

- ▷ Given n random variables with k values each, the 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.

▷ **Bayesian networks.** (First, we do the simple case.)



Independence of Events and Random Variables

▷ **Definition 20.4.2** Events a and b are independent if $P(a \wedge b) = P(a) \cdot P(b)$.

▷ **Proposition 20.4.3** Given independent events a and b where $P(b) \neq 0$, we have $P(a | b) = P(a)$.

▷ **Proof:**

P.1 By definition, $P(a | b) = \frac{P(a \wedge b)}{P(b)}$,

P.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 20.4.4** Random variables X and Y are independent if $\mathbf{P}(X, Y) = \mathbf{P}(X) \cdot \mathbf{P}(Y)$. (System of equations given by dot product!)

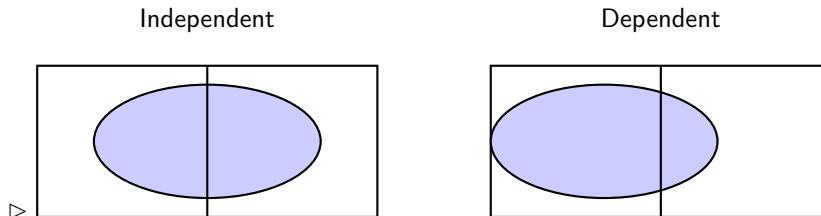


Independence (Examples)

▷ **Example 20.4.5**

- ▷ $P(\text{Dice1} = 6 \wedge \text{Dice2} = 6) = 1/36.$
- ▷ $P(W = \text{sunny} \mid \text{headache}) = P(W = \text{sunny})$ (unless you're weather-sensitive; cf. slide 600)
- ▷ 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:



Oval independent of rectangle, iff split equally



Illustration: Exploiting Independence

▷ **Example 20.4.6** Consider (again) the following 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 $P(c \wedge t \wedge w) = P(c \wedge t) \cdot P(w)$.
- ▷ $\mathbf{P}(\text{Cavity, Toothache, Weather})$ can be reconstructed from the separate tables $\mathbf{P}(\text{Cavity, Toothache})$ and $\mathbf{P}(\text{Weather})$. (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.



20.5 Basic Probabilistic Reasoning Methods

The Product Rule

- ▷ **Proposition 20.5.1 (Product Rule)** Given propositions a and b , $P(a \wedge b) = P(a | b) \cdot P(b)$
- ▷ **Example 20.5.2** $P(\text{cavity} \wedge \text{toothache}) = P(\text{toothache} | \text{cavity}) \cdot P(\text{cavity})$.
- ▷ 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 20.5.3** $\mathbf{P}(X, Y) = \mathbf{P}(X | Y) \cdot \mathbf{P}(Y)$ is a **system of equations**:

$$P(W = \text{sunny} \wedge \text{headache}) = P(W = \text{sunny} | \text{headache}) \cdot P(\text{headache})$$

$$P(W = \text{rain} \wedge \text{headache}) = P(W = \text{rain} | \text{headache}) \cdot P(\text{headache})$$

$$\vdots = \vdots$$

$$P(W = \text{snow} \wedge \neg \text{headache}) = P(W = \text{snow} | \neg \text{headache}) \cdot P(\neg \text{headache})$$

- ▷ Similar for unconditional distributions, $\mathbf{P}(X, Y) = \mathbf{P}(X) \cdot \mathbf{P}(Y)$.



The Chain Rule

- ▷ **Definition 20.5.4 (Chain Rule)** Given random variables X_1, \dots, X_n , we have

$$\mathbf{P}(X_1, \dots, X_n) = \mathbf{P}(X_n | X_{n-1}, \dots, X_1) \cdot \mathbf{P}(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot \mathbf{P}(X_2 | X_1) \cdot \mathbf{P}(X_1)$$

This identity is called the **chain rule**.

- ▷ **Example 20.5.5**

$$\begin{aligned} & P(\neg \text{brush} \wedge \text{cavity} \wedge \text{toothache}) \\ &= P(\text{toothache} | \text{cavity}, \neg \text{brush}) \cdot P(\text{cavity}, \neg \text{brush}) \\ &= P(\text{toothache} | \text{cavity}, \neg \text{brush}) \cdot P(\text{cavity} | \neg \text{brush}) \cdot P(\neg \text{brush}) \end{aligned}$$

- ▷ **Proof:** Iterated application of Product Rule

P.1 $\mathbf{P}(X_1, \dots, X_n) = \mathbf{P}(X_n | X_{n-1}, \dots, X_1) \cdot \mathbf{P}(X_{n-1}, \dots, X_1)$ by Product Rule.

P.2 In turn, $\mathbf{P}(X_{n-1}, \dots, X_1) = \mathbf{P}(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \mathbf{P}(X_{n-2}, \dots, X_1)$, etc. \square

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.



Marginalization

- ▷ Extracting a sub-distribution from a larger joint distribution:
 - ▷ **Proposition 20.5.6 (Marginalization)** *Given sets \mathbf{X} and \mathbf{Y} of random variables, we have:*
- $$\mathbf{P}(\mathbf{X}) = \sum_{y \in \mathbf{Y}} \mathbf{P}(\mathbf{X}, y)$$
- where $\sum_{y \in \mathbf{Y}}$ sums over all possible value combinations of \mathbf{Y} .
- ▷ **Example 20.5.7** (Note: Equation system!)

$$\mathbf{P}(\text{Cavity}) = \sum_{y \in \text{Toothache}} \mathbf{P}(\text{Cavity}, y)$$

$$P(\text{cavity}) = P(\text{cavity, toothache}) + P(\text{cavity, } \neg \text{toothache})$$

$$P(\neg \text{cavity}) = P(\neg \text{cavity, toothache}) + P(\neg \text{cavity, } \neg \text{toothache})$$



Questionnaire

- ▷ Say $P(\text{dog}) = 0.4$, $(\neg \text{dog}) \Leftrightarrow \text{cat}$, and $P(\text{likeslasagna} \mid \text{cat}) = 0.5$.
- ▷ **Question:** Is $P(\text{likeslasagna} \wedge \text{cat})$ is A: 0.2, B: 0.5, C: 0.475, D: 0.3
- ▷ **Answer:** We have $P(\text{cat}) = 0.6$ and $P(\text{likeslasagna} \mid \text{cat}) = 0.5$, hence (D) by the product rule.
- ▷ **Question:** Can we compute the value of $P(\text{likeslasagna})$, given the above information?
- ▷ **Answer:** No. We don't know the probability that *dogs* like lasagna, i.e. $P(\text{likeslasagna} \mid \text{dog})$.



We now come to a very important technique of computing unknown probabilities, which looks almost like magic. Before we formally define it on the next slide, we will get an intuition by considering it in the context of our dentistry example.

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)

$$\begin{aligned} P(\text{cavity} | \text{toothache}) &= \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.12}{P(\text{toothache})} \\ P(\neg \text{cavity} | \text{toothache}) &= \frac{P(\neg \text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.08}{P(\text{toothache})} \end{aligned}$$

▷ 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:

$$1 = \alpha (0.12 + 0.08) \rightsquigarrow \alpha = \frac{1}{0.12 + 0.08} = \frac{1}{0.2} = 5$$

▷ α is a **normalization constant** scaling the sum of relative weights to 1.

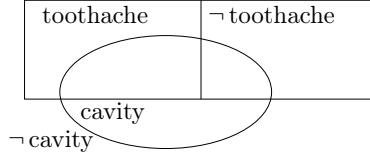


©: Michael Kohlhase

605



To understand what is going on, consider the situation in the following diagram:



Now consider the areas of $A_1 = \text{toothache} \wedge \text{cavity}$ and $A_2 = \text{toothache} \wedge \neg \text{cavity}$ then $A_1 \cup A_2 = \text{toothache}$; this is exactly what we will exploit (see next slide), but we notate it slightly differently in what will be a convenient manner in step 1.

In step 2 we only introduce a convenient placeholder α that makes subsequent argumentation easier.

In step 3, we view A_1 and A_2 as “relative weights”; say that we perceive the left half as “1” (because we already know toothache and don't need to worry about \neg toothache), and we re-normalize to get the desired sum $\alpha A_1 + \alpha A_2 = 1$.⁸

EdN:8

Normalization: Formal

▷ **Definition 20.5.8** 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 $\alpha \langle w_1, \dots, w_k \rangle := \frac{1}{\sum_{i=1}^k w_i}$.

▷ **Example 20.5.9** $\alpha \langle 0.12, 0.08 \rangle = 5 \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle$.

Proposition 20.5.10 (Normalization) *Given a random variable X and an event e , we have $P(X | e) = \alpha P(X, e)$.*

⁸EDNOTE: MK: rrval and sf interact negatively in notes

Proof:

▷ **P.1** For each value x of X , $P(X = x | e) = P(X = x \wedge e) / P(e)$.

So all we need to prove is that $\alpha = 1/P(e)$.

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. \square

P.2 P.3 Example 20.5.11 $\alpha \langle P(\text{cavity} \wedge \text{toothache}), P(\neg \text{cavity} \wedge \text{toothache}) \rangle = \alpha \langle 0.12, 0.08 \rangle$, 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.”

▷ **Normalization+Marginalization:** Given “query variable” X , “observed event” e , and “hidden variables” set Y : $\mathbf{P}(X | e) = \alpha \cdot \mathbf{P}(X, e) = \alpha \cdot \sum_{y \in Y} \mathbf{P}(X, e, y)$.

▷ Second of the four basic techniques in Bayesian networks.



20.6 Bayes’ Rule

Bayes’ Rule

▷ **Definition 20.6.1 (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:**

P.1 By definition, $P(a | b) = \frac{P(a \wedge b)}{P(b)}$

P.2 by the product rule $P(a \wedge b) = P(b | a) \cdot P(a)$ is equal to the claim. \square

Notation: note that this is a system of equations!

$$\mathbf{P}(X | Y) = \frac{\mathbf{P}(Y | X) \cdot \mathbf{P}(X)}{\mathbf{P}(Y)}$$



▷ Applying Bayes’ Rule

▷ **Example 20.6.2** 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?
- ▷ $P(\text{toothache} | \text{cavity})$ is **causal**, $P(\text{cavity} | \text{toothache})$ is **diagnostic**.
- ▷ **Causal dependencies are robust over frequency of the causes.**
- ▷ **Example 20.6.3** 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.
- ▷ 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)



Questionnaire

- ▷ Say $P(\text{dog}) = 0.4$, $P(\text{likeschappi} | \text{dog}) = 0.8$, and $P(\text{likeschappi}) = 0.5$.
- ▷ **Question:** What is $P(\text{dog} | \text{likeschappi})$?
 A: 0.8 B: 0.64 C: 0.9 D: 0.32?
- ▷ **Answer:** By Bayes' rule, $P(\text{dog} | \text{likeschappi}) = \frac{P(\text{likeschappi} | \text{dog}) \cdot P(\text{dog})}{P(\text{likeschappi})} = \frac{0.8 \cdot 0.4}{0.5} = 0.64$ so (B).

- ▷ **Question:** Is $P(\text{dog} \mid \text{likeschappi})$ causal or diagnostic?
- ▷ **Answer:** Diagnostic; liking Chappi does not cause anybody to be a dog.
- ▷ **Question:** Is $P(\text{likeschappi} \mid \text{dog})$ causal or diagnostic?
- ▷ **Answer:** Causal; liking or not liking dog food may be caused by being or not being a dog.



20.7 Conditional Independence

Bayes' Rule with Multiple Evidence

- ▷ **Example 20.7.1** Say we know from medicinical studies that $P(\text{cavity}) = 0.2$, $P(\text{toothache} \mid \text{cavity}) = 0.6$, $P(\text{toothache} \mid \neg\text{cavity}) = 0.1$, $P(\text{catch} \mid \text{cavity}) = 0.9$, and $P(\text{catch} \mid \neg\text{cavity}) = 0.2$.

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 $P(\text{cavity} \mid \text{toothache} \wedge \text{catch})$?

- ▷ **Trial 1:** Bayes' rule

$$P(\text{cavity} \mid \text{toothache} \wedge \text{catch}) = \frac{P(\text{toothache} \wedge \text{catch} \mid \text{cavity}) P(\text{cavity})}{P(\text{toothache} \wedge \text{catch})}$$

- ▷ **Trial 2:** Normalization $\mathbf{P}(X \mid e) = \alpha \mathbf{P}(X, e)$ then Product Rule $\mathbf{P}(X, e) = \mathbf{P}(e \mid X) \mathbf{P}(X)$, with $X = \text{Cavity}$, $e = \text{toothache} \wedge \text{catch}$:

$$\begin{aligned} \mathbf{P}(\text{Cavity} \mid \text{catch} \wedge \text{toothache}) &= \alpha \mathbf{P}(\text{toothache} \wedge \text{catch} \mid \text{Cavity}) \mathbf{P}(\text{Cavity}) \\ \mathbf{P}(\text{cavity} \mid \text{catch} \wedge \text{toothache}) &= \alpha P(\text{toothache} \wedge \text{catch} \mid \text{cavity}) P(\text{cavity}) \\ P(\neg\text{cavity} \mid \text{catch} \wedge \text{toothache}) &= \alpha P(\text{toothache} \wedge \text{catch} \mid \neg\text{cavity}) P(\neg\text{cavity}) \end{aligned}$$



Bayes' Rule with Multiple Evidence, ctd.

- ▷ $\mathbf{P}(\text{Cavity} \mid \text{toothache} \wedge \text{catch}) = \alpha \mathbf{P}(\text{toothache} \wedge \text{catch} \mid \text{Cavity}) \mathbf{P}(\text{Cavity})$
- ▷ **Question:** So, is everything fine?
- ▷ **Answer:** No! We need $\mathbf{P}(\text{toothache} \wedge \text{catch} \mid \text{Cavity})$, i.e. causal dependencies for all combinations of symptoms! (» 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 independent given the presence or absence of cavity!



Conditional Independence

- ▷ **Definition 20.7.2** 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 20.7.3**

$$P(\text{Toothache, Catch} | \text{cavity}) = P(\text{Toothache} | \text{cavity})P(\text{Catch} | \text{cavity})$$

$$P(\text{Toothache, Catch} | \neg \text{cavity}) = P(\text{Toothache} | \neg \text{cavity})P(\text{Catch} | \neg \text{cavity})$$

▷ For cavity: this may cause both, but they don't influence each other.

▷ For \neg cavity: catch and/or toothache would each be caused by something else.

- ▷ **Note:** The definition is symmetric regarding the roles of Z_1 and Z_2 : Toothache is conditionally independent of

- ▷ But there may be dependencies *within* Z_1 or Z_2 , e.g. $Z_2 = \{\text{Toothache, Sleeplessness}\}$.



Conditional Independence, ctd.

- ▷ **Proposition 20.7.4** If Z_1 and Z_2 are conditionally independent given Z , then $P(Z_1 | Z_2, Z) = P(Z_1 | Z)$.

- ▷ **Proof:**

P.1 By definition, $P(Z_1 | Z_2, Z) = \frac{P(Z_1, Z_2, Z)}{P(Z_2, Z)}$

P.2 which by product rule is equal to $\frac{P(Z_1, Z_2 | Z) \cdot P(Z)}{P(Z_2, Z)}$

P.3 which by conditional independence is equal to $\frac{P(Z_1 | Z) \cdot P(Z_2 | Z) \cdot P(Z)}{P(Z_2, Z)}$.

P.4 Since $\frac{P(Z_2 | Z) \cdot P(Z)}{P(Z_2, Z)} = 1$ this proves the claim. \square

- ▷ **Example 20.7.5** Using $\{\text{Toothache}\}$ as Z_1 , $\{\text{Catch}\}$ as Z_2 , and $\{\text{Cavity}\}$ as Z : $P(\text{Toothache} | \text{Catch, Cavity}) = P(\text{Toothache} | \text{Cavity})$.

- ▷ In the presence of conditional independence, we can drop variables from the right-hand side of conditional probabilities.

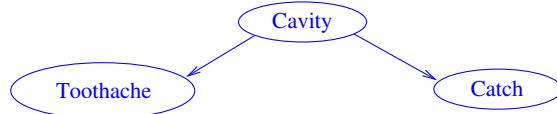
- ▷ Third of the four basic techniques in Bayesian networks.

- ▷ Last missing technique: “Capture variable dependencies in a graph”; illustration see next slide, details see next Chapter



Exploiting Conditional Independence: Overview

- ▷ **1. Graph captures variable dependencies:** (Variables X_1, \dots, X_n)



▷ Given evidence e , want to know $\mathbf{P}(X | e)$.

▷ Remaining vars: \mathbf{Y} .

- ▷ **2. Normalization+Marginalization:**

$$\mathbf{P}(X | e) = \alpha \cdot \mathbf{P}(X, e); \text{ if } \mathbf{Y} \neq \emptyset \text{ then } \mathbf{P}(X | e) = \alpha \cdot \sum_{\mathbf{y} \in \mathbf{Y}} \mathbf{P}(X, e, \mathbf{y})$$

▷ A sum over atomic events!

- ▷ **3. Chain rule:** Order X_1, \dots, X_n consistently with dependency graph.

$$\begin{aligned} \mathbf{P}(X_1, \dots, X_n) &= \\ \mathbf{P}(X_n | X_{n-1}, \dots, X_1) \cdot \mathbf{P}(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot \mathbf{P}(X_1) \end{aligned}$$

- ▷ **4. Exploit conditional independence:** Instead of $\mathbf{P}(X_i | X_{i-1}, \dots, X_1)$, with previous slide we can use $\mathbf{P}(X_i | \text{Parents}(X_i))$.

▷ Bayesian networks!



Exploiting Conditional Independence: Example

- ▷ **1. Graph captures variable dependencies:** (See previous slide.)

▷ Given toothache, catch, want $\mathbf{P}(\text{Cavity} | \text{toothache, catch})$. Remaining vars: \emptyset .

- ▷ **2. Normalization+Marginalization:**

$$\mathbf{P}(\text{Cavity} | \text{toothache, catch}) = \alpha \cdot \mathbf{P}(\text{Cavity, toothache, catch})$$

- ▷ **3. Chain rule:** Order $X_1 = \text{Cavity}$, $X_2 = \text{Toothache}$, $X_3 = \text{Catch}$.

$$\begin{aligned} \mathbf{P}(\text{Cavity, toothache, catch}) &= \\ \mathbf{P}(\text{catch} | \text{toothache, Cavity}) \cdot \mathbf{P}(\text{toothache} | \text{Cavity}) \cdot \mathbf{P}(\text{Cavity}) \end{aligned}$$

▷ **4. Exploit conditional independence:**

Instead of $\mathbf{P}(\text{catch} \mid \text{toothache}, \text{Cavity})$ use $\mathbf{P}(\text{catch} \mid \text{Cavity})$.

▷ **Thus:**

$$\begin{aligned}\mathbf{P}(\text{Cavity} \mid \text{toothache}, \text{catch}) &= \alpha \cdot \mathbf{P}(\text{catch} \mid \text{Cavity}) \cdot \mathbf{P}(\text{toothache} \mid \text{Cavity}) \cdot \mathbf{P}(\text{Cavity}) \\ &= \alpha \cdot \langle 0.9 \cdot 0.6 \cdot 0.2, 0.2 \cdot 0.1 \cdot 0.8 \rangle \\ &= \alpha \cdot \langle 0.108, 0.016 \rangle\end{aligned}$$

▷ So $\alpha \approx 8.06$ and $\mathbf{P}(\text{cavity} \mid \text{toothache} \wedge \text{catch}) \approx 0.87$.



Naive Bayes Models

▷ **Definition 20.7.6** 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 classifiers](#). (also called [idiot Bayes model](#) by Bayesian fundamentalists)

▷ **Observation 20.7.7** In a naive Bayes model, the full joint probability distribution can be written as

$$\mathbf{P}(\text{cause} \mid \text{effect}_1, \dots, \text{effect}_n) = \mathbf{P}(\text{cause}) \cdot \prod_i \mathbf{P}(\text{effect}_i \mid \text{cause})$$

▷ This kind of model is called “naive” or “idiot” since it is often used as a simplifying model if the effects are not conditionally independent after all.

▷ In practice, naive Bayes systems can work surprisingly well, even when the conditional independence assumption is not true.

▷ **Example 20.7.8** The dentistry example is a (true) naive Bayes model.



Questionnaire

▷ Consider the random variables $X_1 = \text{Animal}$, $X_2 = \text{LikesChappi}$, and $X_3 = \text{LoudNoise}$, and X_1 has values $\{\text{dog, cat, other}\}$, X_2 and X_3 are Boolean.

▷ **Question:** Which statements are correct?

- (A) Animal is independent of LikesChappi.
- (B) LoudNoise is independent of LikesChappi.
- (C) Animal is conditionally independent of LikesChappi given LoudNoise.
- (D) LikesChappi is conditionally independent of LoudNoise given Animal.

▷ **Answer:**

- (A) No: likeschappi indicates dog.
- (B) No: Not knowing what animal it is, loudnoise is an indication for dog which indicates likeschappi.
- (C) No: For example, even if we know loudnoise, knowing in addition that likeschappi gives us a stronger indication of Animal = dog.
- (D) Yes: If we know what animal it is, LoudNoise does not influence LikesChappi. (Well, at least that's a reasonable assumption.)



20.8 The Wumpus World Revisited

We will fortify our intuition about naive Bayes models with a variant of the Wumpus world we looked at in Example 20.1.18 to understand whether logic was up to the job of guiding an agent in the Wumpus cave.

Wumpus World Revisited

▷ **Example 20.8.1 (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!

1.1	2.1	3.1	4.1
1.2 B OK	2.2	3.2	4.2
1.3	2.3	3.3	4.3
1.4	2.4	3.4	4.4

▷ **Idea:** Let's evaluate our probabilistic reasoning machinery, if that can help!



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

- ▷ Full joint probability distribution

1. $\mathbf{P}(P_{1,1}, \dots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1}) = \mathbf{P}(B_{1,1}, B_{1,2}, B_{2,1} | P_{1,1}, \dots, P_{4,4}) \cdot \mathbf{P}(P_{1,1}, \dots, P_{4,4})$ (Product Rule)
2. $\mathbf{P}(P_{1,1}, \dots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} \mathbf{P}(P_{i,j})$ (pits are spread independently)
3. $\mathbf{P}(P_{1,1}, \dots, P_{4,4}) = 0.2^n \cdot 0.8^{16-n}$ (probability of a pit is 0.2 and there are n pits)

Wumpus: Query and Simple Reasoning

Assume that we have evidence:

▷ $b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$ 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	2,1 B OK	3,1	4,1
OK	OK	OK	OK

▷ The answer can be computed by enumeration of the full joint probability distribution.

▷ Let U be the variables $P_{i,j}$ except $P_{1,3}$ and κ , then

$$P(P_{1,3} | \kappa, b) = \sum_{u \in U} \mathbf{P}(P_{1,3}, u, \kappa, b)$$

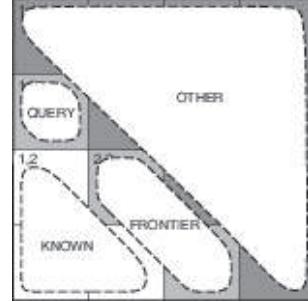
▷ Problem: We need to explore all possible values of variables in U ($2^{12} = 4096$ terms!)

▷ Can we do better (faster)?

Wumpus: Conditional Independence

▷ Observation 20.8.2

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.

From conditional independence we get: $P(b | P_{1,3}, \kappa, U) = P(b | P_{1,3}, \kappa, F)$

Now, let us exploit this formula.



©: Michael Kohlhase

622



Wumpus: Reasoning

▷ We calculate:

$$\begin{aligned}
 P(P_{1,3} | \kappa, b) &= \alpha \sum_{u \in U} \mathbf{P}(P_{1,3}, u, \kappa, b) \\
 &= \alpha \sum_{u \in U} \mathbf{P}(b | P_{1,3}, \kappa, u) \cdot \mathbf{P}(P_{1,3}, \kappa, u) \\
 &= \alpha \sum_{f \in F} \sum_{o \in O} \mathbf{P}(b | P_{1,3}, \kappa, f, o) \cdot \mathbf{P}(P_{1,3}, \kappa, f, o) \\
 &= \alpha \sum_{f \in F} \mathbf{P}(b | P_{1,3}, \kappa, f) \cdot \sum_{o \in O} \mathbf{P}(P_{1,3}, \kappa, f, o) \\
 &= \alpha \sum_{f \in F} \mathbf{P}(b | P_{1,3}, \kappa, f) \cdot \sum_{o \in O} \mathbf{P}(P_{1,3}) \cdot P(\kappa) \cdot P(f) \cdot P(o) \\
 &= \alpha \mathbf{P}(P_{1,3}) P(\kappa) \sum_{f \in F} \mathbf{P}(b | P_{1,3}, \kappa, f) \cdot P(f) \cdot \sum_{o \in O} P(o) \\
 &= \alpha' P(P_{1,3}) \sum_{f \in F} \mathbf{P}(b | P_{1,3}, \kappa, f) \cdot P(f)
 \end{aligned}$$

for $\alpha' := \alpha P(\kappa)$ as $\sum_{o \in O} P(o) = 1$.



©: Michael Kohlhase

623

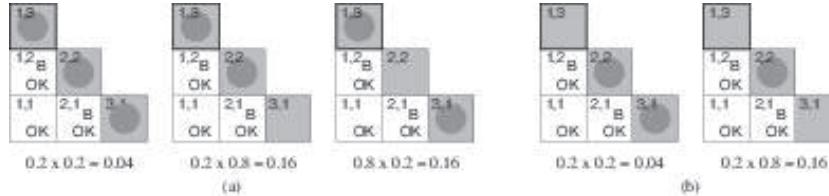


Wumpus: Solution

- ▷ We calculate using the product rule and conditional independence (see above)

$$P(P_{1,3} | \kappa, b) = \alpha' P(P_{1,3}) \sum_{f \in F} \mathbf{P}(b | P_{1,3}, \kappa, f) \cdot P(f)$$

- ▷ Let us explore possible models (values) of Fringe that are F compatible with observation b .



$$\triangleright P(P_{1,3} | \kappa, b) = \alpha' \langle 0.2 (0.04 + 0.16 + 0.16), 0.8 (0.04 + 0.16) \rangle = \langle 0.31, 0.69 \rangle$$

$$\triangleright P(P_{3,1} | \kappa, b) = \langle 0.31, 0.69 \rangle \text{ by symmetry}$$

$$\triangleright P(P_{2,2} | \kappa, b) = \langle 0.86, 0.14 \rangle \quad (\text{definitely avoid})$$



20.9 Conclusion

Summary

- ▷ **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 means to 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 manner.



Reading: Chapter 13: Quantifying Uncertainty [RN03].

Content: Sections 13.1 and 13.2 roughly correspond to my “Introduction” and “Probability Theory Concepts”. Section 13.3 and 13.4 roughly correspond to my “Basic Probabilistic Inference”. Section 13.5 roughly corresponds to my “Bayes’ Rule” and “Multiple Evidence”.

In Section 13.6, RN go back to the Wumpus world and discuss some inferences in a probabilistic version thereof.

Overall, the content is quite similar. I have added some examples, have tried to make a few

subtle points more explicit, and I indicate already how these techniques will be used in Bayesian networks. RN gives many complementary explanations, nice as additional background reading.

Chapter 21

Probabilistic Reasoning, Part II: Bayesian Networks

21.1 Introduction

Reminder: Our Agenda for This Topic

- ▷ Our treatment of the topic “Probabilistic Reasoning” consists of this and last Section.
 - ▷ Chapter 20: All the basic machinery at use in Bayesian networks.
 - ▷ **This Chapter:** Bayesian networks: What they are, how to build them, how to use them.
 - ▷ The most wide-spread and successful practical framework for probabilistic reasoning.



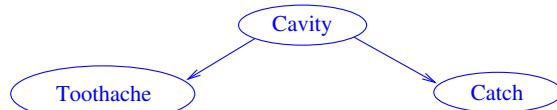
©: Michael Kohlhase

626



Reminder: Our Machinery

- ▷ **1. Graph captures variable dependencies:** (Variables X_1, \dots, X_n)



- ▷ Given evidence e , want to know $\mathbf{P}(X | e)$. Remaining vars: \mathbf{Y} .

- ▷ **2. Normalization+Marginalization:**

$$\mathbf{P}(X | e) = \alpha \mathbf{P}(X, e) = \alpha \sum_{\mathbf{y} \in \mathbf{Y}} \mathbf{P}(X, e, \mathbf{y})$$

- ▷ A sum over atomic events!

- ▷ **3. Chain rule:** X_1, \dots, X_n consistently with dependency graph.

$$\mathbf{P}(X_1, \dots, X_n) = \mathbf{P}(X_n | X_{n-1}, \dots, X_1) \cdot \mathbf{P}(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot \mathbf{P}(X_1)$$

- ▷ **4. Exploit conditional independence:** Instead of $\mathbf{P}(X_i | X_{i-1}, \dots, X_1)$, we can use $\mathbf{P}(X_i | \text{Parents}(X_i))$.

- ▷ Bayesian networks!



©: Michael Kohlhase

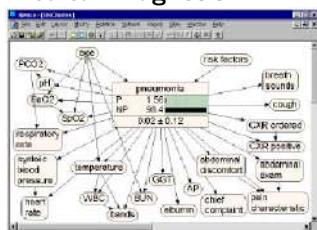
627



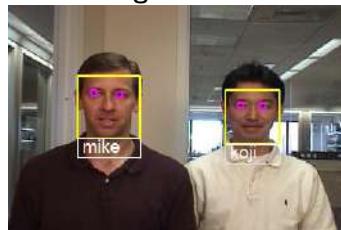
Some Applications

- ▷ A ubiquitous problem: Observe “symptoms”, need to infer “causes”.

Medical Diagnosis



Face Recognition



Self-Localization



Nuclear Test Ban



©: Michael Kohlhase

628



Our Agenda for This Chapter

- ▷ **What is a Bayesian Network?** 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.



21.2 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 $P(X_i | Parents(X_i))$.*”
 - ▷ “*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 : $P(X | e)$.



John, Mary, and My Brand-New Alarm

▷ Example 21.2.1 (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 BN

▷ Cooking Recipe:

- (1) Design the random variables X_1, \dots, X_n ;
- (2) Identify their dependencies;
- (3) Insert the conditional probability tables $\mathbf{P}(X_i | \text{Parents}(X_i))$.

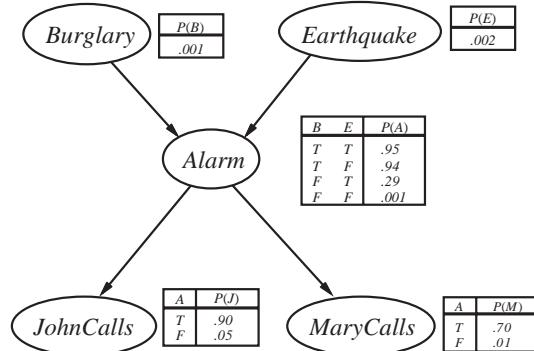
▷ Example 21.2.2 (Let's cook!)

- (1) Random variables: Burglary, Earthquake, Alarm, JohnCalls, MaryCalls.
- (2) Dependencies: Burglaries and earthquakes are independent (this is actually debatable \rightsquigarrow 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 BN

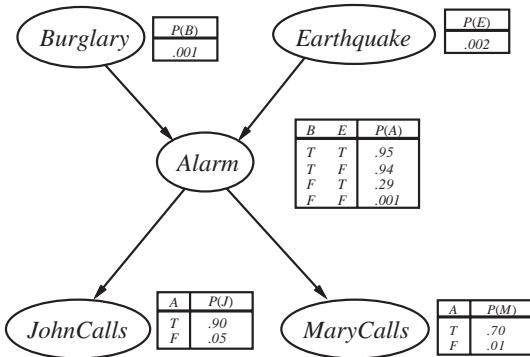
▷ Example 21.2.3



Note: In each $\mathbf{P}(X_i | \text{Parents}(X_i))$, we show only $\mathbf{P}(X_i = T | \text{Parents}(X_i))$. We don't show $\mathbf{P}(X_i = F | \text{Parents}(X_i))$ which is $1 - \mathbf{P}(X_i = T | \text{Parents}(X_i))$.



▷ The Syntax of Bayesian Networks



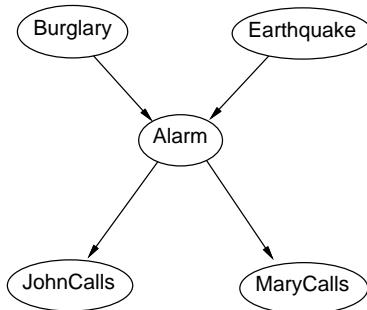
▷ **Definition 21.2.4 (Bayesian Network)** Given random variables X_1, \dots, X_n with finite domains D_1, \dots, D_n , a **Bayesian network** (also **belief network** or **probabilistic network**) is an acyclic directed graph $BN = \langle \{X_1, \dots, X_n\}, E \rangle$. We denote $\text{Parents}(X_i) := \{X_j \mid (X_j, X_i) \in E\}$. 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**.

▷ Related formalisms summed up under the term **graphical models**.



21.3 What is the Meaning of a Bayesian Network?

The Semantics of BNs: Illustration



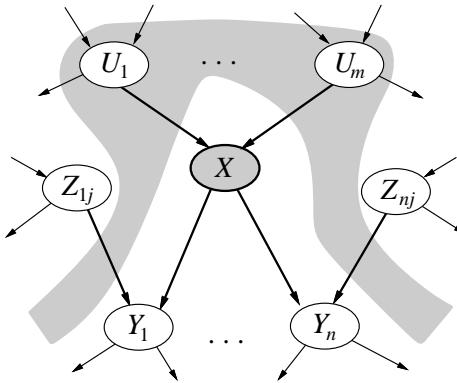
- ▷ Alarm depends on Burglary and Earthquake.
- ▷ MaryCalls only depends on Alarm. $P(\text{MaryCalls} \mid \text{Alarm}, \text{Burglary}) = P(\text{MaryCalls} \mid \text{Alarm})$
- ▷ Bayesian networks represent sets of independence assumptions.



The Semantics of BNs: Illustration, ctd.

- ▷ **Observation 21.3.1** Each node X in a BN is conditionally independent of its

non-descendants given its parents $\text{Parents}(X)$.

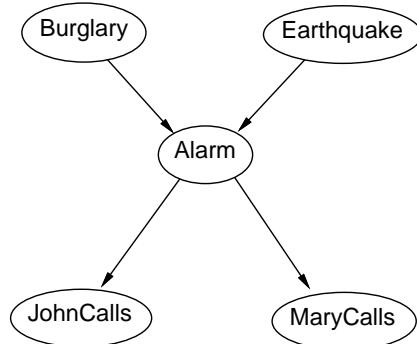


Question: Why “non-descendants” of X ?

- ▷ **Intuition:** Given that BNs are acyclic, these are exactly those nodes that **could** have an edge into X .



The Semantics of BNs: Questionnaire.

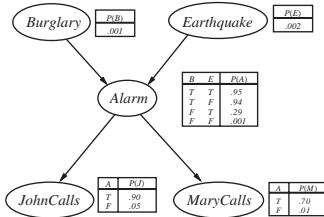


- ▷ **Question:** Given the value of Alarm , MaryCalls is independent of?

- ▷ **Answer:** Its non-descendants $\{\text{Burglary}, \text{Earthquake}, \text{JohnCalls}\}$.



The Semantics of BNs: Formal



▷ **Definition 21.3.2** Given a Bayesian network $BN = \langle \{X_1, \dots, X_n\}, E \rangle$, we identify BN with the following two assumptions:

- For $1 \leq i \leq n$, X_i is conditionally independent of $\text{NonDesc}(X_i)$ given $\text{Parents}(X_i)$, where $\text{NonDesc}(X_i) := \{X_j \mid (X_i, X_j) \notin E^*\} \setminus \text{Parents}(X_i)$ where E^* is the transitive-reflexive closure of E .
- For $1 \leq i \leq n$, all values x_i of X_i , and all value combinations of $\text{Parents}(X_i)$, we have $P(x_i \mid \text{Parents}(X_i)) = \text{CPT}(x_i, \text{Parents}(X_i))$.



Recovering the Full Joint Probability Distribution

▷ “A Bayesian network is a methodology for representing the full joint probability distribution.”

▷ **Problem:** How to recover the full joint probability distribution $\mathbf{P}(X_1, \dots, X_n)$ from $BN = \langle \{X_1, \dots, X_n\}, E \rangle$?

▷ **Chain rule:** For any ordering X_1, \dots, X_n , we have:

$$\mathbf{P}(X_1, \dots, X_n) = \mathbf{P}(X_n \mid X_{n-1}, \dots, X_1) \cdot \mathbf{P}(X_{n-1} \mid X_{n-2}, \dots, X_1) \cdot \dots \cdot \mathbf{P}(X_1)$$

Choose X_1, \dots, X_n consistent with BN : $X_j \in \text{Parents}(X_i) \rightsquigarrow j < i$.

▷ **Observation 21.3.3 (Exploiting Conditional Independence)**

With BN assumption (A), we can use $\mathbf{P}(X_i \mid \text{Parents}(X_i))$ instead of $\mathbf{P}(X_i \mid X_{i-1}, \dots, X_1)$:

$$\mathbf{P}(X_1, \dots, X_n) = \prod_{i=1}^n \mathbf{P}(X_i \mid \text{Parents}(X_i))$$

The distributions $\mathbf{P}(X_i \mid \text{Parents}(X_i))$ are given by BN assumption (B).

▷ Same for atomic events $P(x_1, \dots, x_n)$.

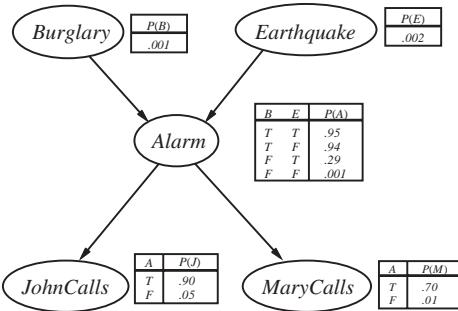
▷ **Observation 21.3.4 (Why “acyclic”?)** for cyclic BN , this does NOT hold, indeed cyclic BNs may be self-contradictory. (need a consistent ordering)



Recovering a Probability for John, Mary, and the Alarm

- ▷ **Example 21.3.5** John and Mary called because there was an alarm, but no earthquake or burglary

$$\begin{aligned} P(j, m, a, \neg b, \neg e) &= P(j | a) \cdot P(m | a) \cdot P(a | \neg b, \neg e) \cdot P(\neg b) \cdot P(\neg e) \\ &= 0.9 * 0.7 * 0.001 * 0.999 * 0.998 \\ &= 0.00062 \end{aligned}$$



Questionnaire



- ▷ Say BN is the Bayesian network above. Which statements are correct?

- (A) Animal is independent of LikesChappi.
- (B) LoudNoise is independent of LikesChappi.
- (C) Animal is conditionally independent of LikesChappi given LoudNoise.
- (D) LikesChappi is conditionally independent of LoudNoise given Animal.

- ▷ Answers:

- (A) No: likeschappi indicates dog.
- (B) No: Not knowing what animal it is, likeschappi is an indication for dog which indicates loudnoise.
- (C) No: For example, even if we know loudnoise, knowing in addition that likeschappi gives us a stronger indication of Animal = dog.
- (D) Yes: $X_i = \text{LikesChappi}$ is conditionally independent of $\text{NonDesc}(X_i) = \{\text{LoudNoise}\}$ given $\text{Parents}(X_i) = \{\text{Animal}\}$.

21.4 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 $\mathbf{P}(X_i | X_{i-1}, \dots, X_1) = \mathbf{P}(X_i | \text{Parents}(X_i))$.
 - 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 $\mathbf{P}(X_i | \text{Parents}(X_i))$.

Attention: Which variables we need to include into $\text{Parents}(X_i)$ depends on what " $\{X_1, \dots, X_{i-1}\}$ " is ... !

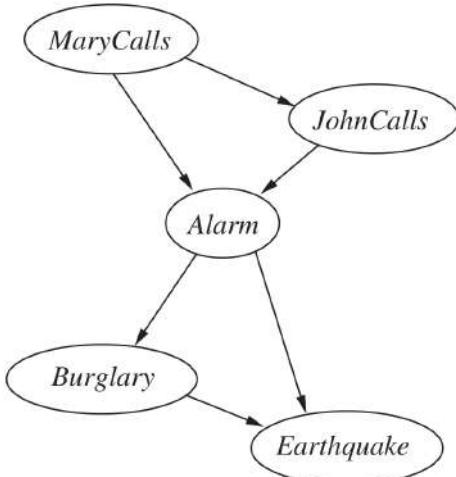
▷ The size of the resulting BN depends on the chosen order X_1, \dots, X_n .

▷ The size of a Bayesian network is *not* a fixed property of the domain. It depends on the skill of the designer.



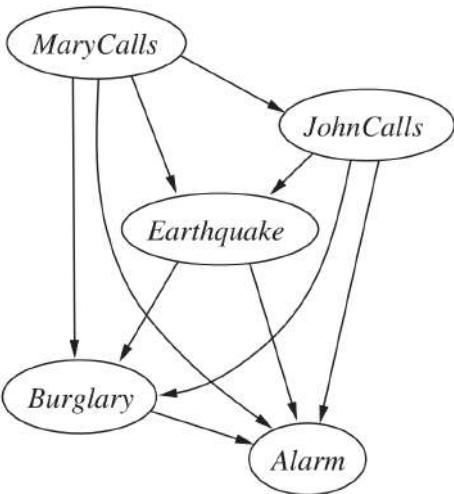
John and Mary Depend on the Variable Order!

▷ Example 21.4.1 MaryCalls, JohnCalls, Alarm, Burglary, Earthquake.

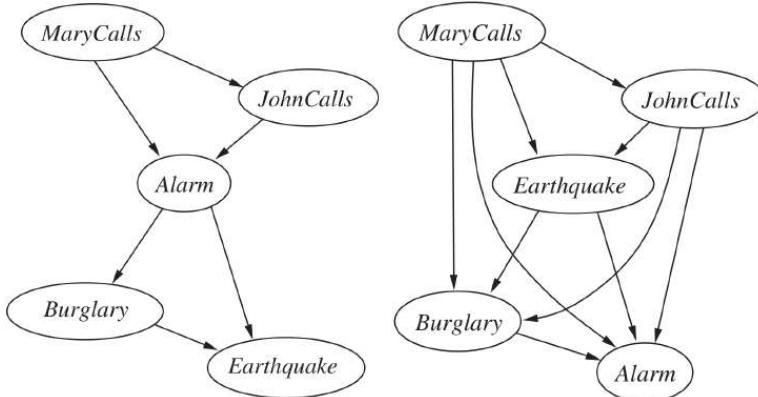


John and Mary Depend on the Variable Order! Ctd.

▷ Example 21.4.2 MaryCalls, JohnCalls, Earthquake, Burglary, Alarm.



John and Mary, What Went Wrong?



- ▷ These BNs link from symptoms to causes! (P(Cavity | Toothache))
- ▷ We fail to identify many conditional independence relations (e.g., get dependencies between conditionally independent symptoms).
- ▷ 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 21.4.3** Given random variables X_1, \dots, X_n with finite domains D_1, \dots, D_n , the size of $BN = \langle \{X_1, \dots, X_n\}, E \rangle$ is defined as $\text{size}(BN) := \sum_{i=1}^n \#(D_i) \cdot \prod_{X_j \in \text{Parents}(X_i)} \#(D_j)$.

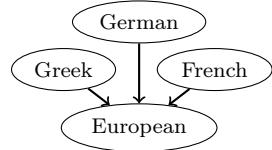
- ▷ = The total number of entries in the CPTs.
- ▷ Smaller BN \sim assess less probabilities, more efficient inference.
- ▷ Explicit full joint probability distribution has size $\prod_{i=1}^n \#(D_i)$.
- ▷ If $\#\text{Parents}(X_i) \leq k$ for every X_i , and D_{\max} is the largest variable domain, then $\text{size}(BN) \leq n \#(D_{\max})^{k+1}$.
- ▷ 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, $\text{size}(BN) = n \cdot \prod_{i=1}^n \#(D_i)$, 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

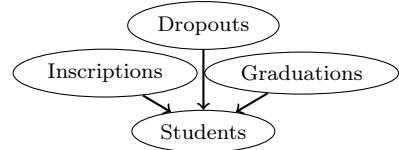
- ▷ Problem: Even if $\max(\text{Parents})$ is small, the CPT has 2^k entries. (worst-case)
- ▷ Idea: Usually CPTs follow standard patterns called canonical distributions.
- ▷ only need to determine pattern and some values.
- ▷ Definition 21.4.4 A node X in a Bayesian network is called deterministic, if its value is completely determined by the values of $\text{Parents}(X)$.
- ▷ Example 21.4.5 (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 21.4.6 (Numerical Dependencies)

In the network on the right, the node *Students* is deterministic, the CPT corresponds to a sum, i.e. $P(S = i - d - g) = P(I = i) \wedge P(D = d) \wedge P(G = g)$.



- ▷ Intuition: Deterministic nodes model direct, causal relationships



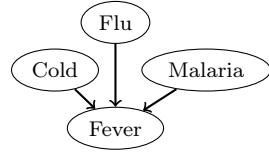
Representing Conditional Distributions: Noisy Nodes

- ▷ Problem: Sometimes, values of nodes are only “almost deterministic”. (uncertain, but mostly logical)

▷ **Idea:** Use “noisy” logical relationships. (generalize logical ones softly to [0, 1])

▷ **Example 21.4.7 (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 Example 21.4.7:

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_d .

▷ **Definition 21.4.8** The CPT of a **noisy disjunction node** X in a Bayesian network is given by $P(x_i | \text{Parents}(X_i)) = \prod_{\{j | X_j=\top\}} q_j$, where the q_i are the inhibition factors of $X_i \in \text{Parents}(X)$.



Representing Conditional Distributions: Noisy Nodes

▷ **Example 21.4.9** We have the following inhibition factors for Example 21.4.7:

$$\begin{aligned} q_{\text{cold}} &= P(\neg \text{fever} | \text{cold}, \neg \text{flu}, \neg \text{malaria}) = 0.6 \\ q_{\text{flu}} &= P(\neg \text{fever} | \neg \text{cold}, \text{flu}, \neg \text{malaria}) = 0.2 \\ q_{\text{malaria}} &= P(\neg \text{fever} | \neg \text{cold}, \neg \text{flu}, \text{malaria}) = 0.1 \end{aligned}$$

If we model *Fever* as a noisy disjunction node, then the general rule $P(x_i | \text{Parents}(X_i)) = \prod_{\{j | X_j=\top\}} q_j$ 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 · 0.1
T	F	F	0.4	0.6
T	F	T	0.94	0.06 = 0.6 · 0.1
T	T	F	0.88	0.12 = 0.6 · 0.2
T	T	T	0.988	0.012 = 0.6 · 0.2 · 0.1



Representing Conditional Distributions: Noisy Nodes

▷ **Observation 21.4.10** 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 21.4.11** 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.



Questionnaire

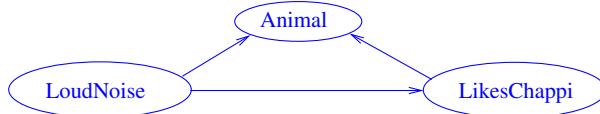
- ▷ **Question:** What is the Bayesian network we get by constructing according to the ordering $X_1 = \text{LoudNoise}$, $X_2 = \text{Animal}$, $X_3 = \text{LikesChappi}$?

- ▷ **Answer:**



- ▷ **Question:** What is the Bayesian network we get by constructing according to the ordering $X_1 = \text{LoudNoise}$, $X_2 = \text{LikesChappi}$, $X_3 = \text{Animal}$?

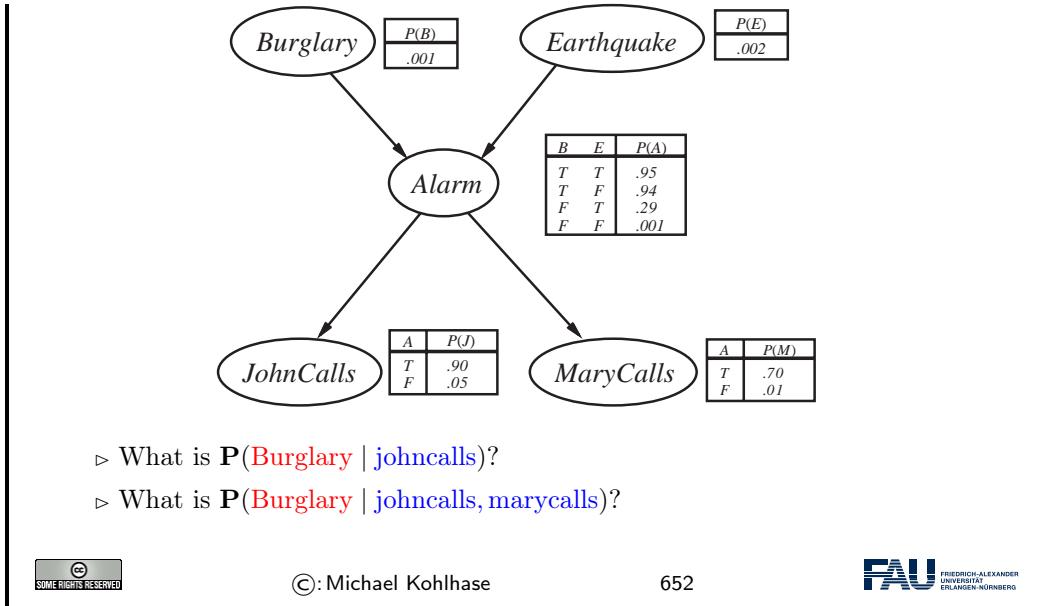
- ▷ **Answer:**



21.5 Inference in Bayesian Networks

Inference for Mary and John

- ▷ **Intuition:** Observe **evidence variables** and draw conclusions on **query variables**.
- ▷ **Example 21.5.1**



Probabilistic Inference Tasks in Bayesian Networks

▷ **Definition 21.5.2 (Probabilistic Inference Task)** Given random variables X_1, \dots, X_n , a **probabilistic inference task** consists of a set $\mathbf{X} \subseteq \{X_1, \dots, X_n\}$ of **query variables**, a set $\mathbf{E} \subseteq \{X_1, \dots, X_n\}$ of **evidence variables**, and an **event** \mathbf{e} that assigns values to \mathbf{E} . We wish to compute the **posterior probability distribution** $\mathbf{P}(\mathbf{X} \mid \mathbf{e})$.

$\mathbf{Y} := \{X_1, \dots, X_n\} \setminus (\mathbf{X} \cup \mathbf{E})$ are the **hidden variables**.

▷ **Notes:**

- ▷ We assume that a *BN* for X_1, \dots, X_n is given.
- ▷ In the remainder, for simplicity, $\mathbf{X} = \{X\}$ is a singleton.

▷ **Example 21.5.3** In $\mathbf{P}(\text{Burglary} \mid \text{johncalls}, \text{marycalls})$, $\mathbf{X} = \text{Burglary}$, $\mathbf{e} = \text{johncalls}, \text{marycalls}$, and $\mathbf{Y} = \{\text{Alarm}, \text{EarthQuake}\}$.

Inference by Enumeration: The Principle (A Reminder!)

- ▷ **Problem:** Given evidence \mathbf{e} , want to know $\mathbf{P}(X \mid \mathbf{e})$. Hidden variables: \mathbf{Y} .
- ▷ **1. Bayesian network BN captures variable dependencies.**
- ▷ **2. Normalization+Marginalization.**

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}); \text{ if } \mathbf{Y} \neq \emptyset \text{ then } \mathbf{P}(X \mid \mathbf{e}) = \alpha \sum_{\mathbf{y} \in \mathbf{Y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y})$$

- ▷ Recover the summed-up probabilities $\mathbf{P}(X, \mathbf{e}, \mathbf{y})$ from *BN*!

- ▷ 3. **Chain rule.** Order X_1, \dots, X_n consistent with BN .

$$\mathbf{P}(X_1, \dots, X_n) = \mathbf{P}(X_n | X_{n-1}, \dots, X_1) \cdot \mathbf{P}(X_{n-1} | X_{n-2}, \dots, X_1) \cdot \dots \cdot \mathbf{P}(X_1)$$

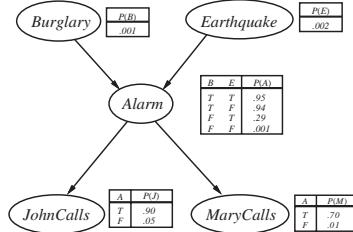
- ▷ 4. **Exploit conditional independence.** Instead of $\mathbf{P}(X_i | X_{i-1}, \dots, X_1)$, use $\mathbf{P}(X_i | \text{Parents}(X_i))$.

- ▷ Given a Bayesian network BN , probabilistic inference tasks can be solved as sums of products of conditional probabilities from BN .

- ▷ Sum over all value combinations of hidden variables.



Inference by Enumeration: John and Mary



- ▷ Want: $\mathbf{P}(\text{Burglary} | \text{johncalls}, \text{marycalls})$.
 Hidden variables: $\mathbf{Y} = \{\text{Earthquake}, \text{Alarm}\}$.

- ▷ **Normalization+Marginalization:**

$$\mathbf{P}(B | j, m) = \alpha \cdot \mathbf{P}(B, j, m) = \alpha \sum_{v_E} \sum_{v_A} \mathbf{P}(B, j, m, v_E, v_A)$$

- ▷ **Order** $X_1 = B, X_2 = E, X_3 = A, X_4 = J, X_5 = M$.

- ▷ **Chain rule and conditional independence:**

$$\mathbf{P}(B | j, m) = \alpha \sum_{v_E} \sum_{v_A} \mathbf{P}(B) \cdot P(v_E) \cdot \mathbf{P}(v_A | B, v_E) \cdot P(j | v_A) \cdot P(m | v_A)$$

- ▷ Continuation on next slide ...



Inference by Enumeration: John and Mary, ctd.

- ▷ **Move variables outwards** (until we hit the first parent):

$$\mathbf{P}(B | j, m) = \alpha \cdot \mathbf{P}(B) \cdot \sum_{v_E} P(v_E) \cdot \sum_{v_A} \mathbf{P}(v_A | B, v_E) \cdot P(j | v_A) \cdot P(m | v_A)$$

- ▷ The probabilities of the outside-variables multiply the entire “rest of the sum”

▷ Chain rule and conditional independence, ctd.:

$$\begin{aligned}
 & \mathbf{P}(B \mid j, m) \\
 &= \alpha \mathbf{P}(B) \sum_{v_E} P(v_E) \sum_{v_A} \mathbf{P}(v_A \mid B, v_E) P(j \mid v_A) P(m \mid v_A) \\
 &= \alpha \cdot P(b) \cdot \left(P(e) \cdot \left(+ \underbrace{\overbrace{P(a \mid b, e) P(j \mid a) P(m \mid a)}^a}_{P(\neg a \mid b, e) P(j \mid \neg a) P(m \mid \neg a)} \right)_e \right. \\
 &\quad \left. + P(\neg e) \cdot \left(+ \underbrace{\overbrace{P(a \mid b, \neg e) P(j \mid a) P(m \mid a)}^{\neg a}}_{P(\neg a \mid b, \neg e) P(j \mid \neg a) P(m \mid \neg a)} \right)_{\neg e} \right) \\
 &= \alpha \langle 0.00059224, 0.0014919 \rangle \approx \langle 0.284, 0.716 \rangle
 \end{aligned}$$



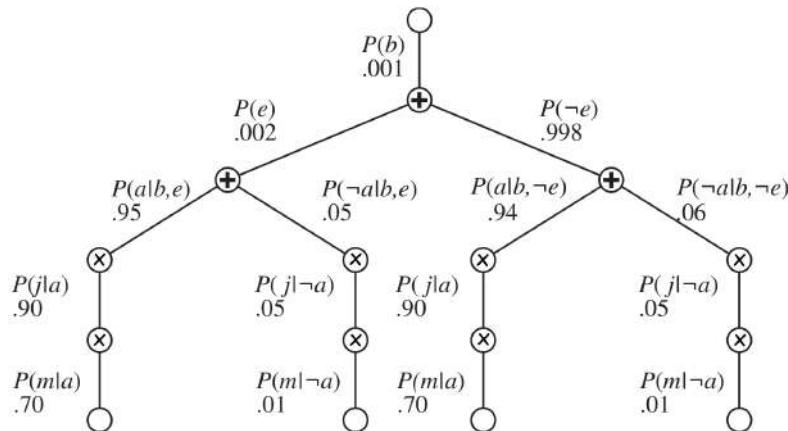
©: Michael Kohlhase

656



This computation can be viewed as a “search tree”, see next slide.

The Evaluation of $P(b \mid j, m)$, as a “Search Tree”



▷ Inference by enumeration = a tree with “sum nodes” branching over values of hidden variables, and with non-branching “multiplication nodes”.



©: Michael Kohlhase

657



Inference by Enumeration: Variable Elimination

▷ 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 21.5.4** Variable elimination is a BNI algorithm that avoids
 - ▷ (A) repeated computation, and (see below)
 - ▷ (B) irrelevant computation. (see below)
- ▷ In some special cases, variable elimination runs in polynomial time.



Variable Elimination: Sketch of Ideas

- ▷ **(A) 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):

$$\mathbf{P}(B | j, m) = \alpha \underbrace{\mathbf{P}(B)}_{\mathbf{f}_1(B)} \sum_{v_E} \underbrace{\mathbf{P}(v_E)}_{\mathbf{f}_2(E)} \sum_{v_A} \underbrace{\mathbf{P}(v_A | B, v_E)}_{\mathbf{f}_3(A, B, E)} \underbrace{\mathbf{P}(j | v_A)}_{\mathbf{f}_4(A)} \underbrace{\mathbf{P}(m | v_A)}_{\mathbf{f}_5(A)}$$

2. Then the computation is performed in terms of **factor product** and **summing out variables** from factors:

$$\mathbf{P}(B | j, m) = \alpha \cdot \mathbf{f}_1(B) \cdot \sum_{v_E} \mathbf{f}_2(E) \cdot \sum_{v_A} \mathbf{f}_3(A, B, E) \cdot \mathbf{f}_4(A) \cdot \mathbf{f}_5(A)$$

- ▷ **(B) Avoiding irrelevant computation:** Repeatedly remove hidden variables that are leaf nodes.

- ▷ For query $P(\text{JohnCalls} | \text{burglary})$:

$$\mathbf{P}(J | b) = \alpha P(b) \sum_{v_E} P(v_E), \sum_{v_A} P(v_A | b, v_E) \mathbf{P}(J | v_A) \sum_{v_M} P(v_M | v_A)$$

▷ The rightmost sum equals 1 and can be dropped.



The Complexity of Exact Inference

- ▷ **Good News:**
 - ▷ **Definition 21.5.5** A graph is called **singly connected**, or a **polytree**, if there is at most one undirected path between any two nodes in the graph.
 - ▷ **Theorem 21.5.6** On polytree Bayesian networks, variable elimination runs in polynomial time.
- ▷ Is our BN for Mary & John a polytree?
- ▷ **Bad News:**

- ▷ For multiply connected Bayesian networks, in general probabilistic inference is $\#\text{P}$ -hard.
- ▷ $\#\text{P}$ is harder than NP (i.e. $\text{NP} \subseteq \#\text{P}$).
- ▷ **So?**
- ▷ Life goes on ... In the hard cases, if need be we can throw exactitude to the winds and approximate.
- ▷ **Example 21.5.7** Sampling techniques



21.6 Conclusion

Summary

- ▷ **Bayesian networks (BN)** are a wide-spread tool to model uncertainty, and to reason about it. A BN represents **conditional independence relations** between random variables. It consists of a graph encoding the variable dependencies, and of **conditional probability tables (CPTs)**.
- ▷ Given a variable 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 $\#\text{P}$ -hard. Approximate probabilistic inference methods exist.



Topics We Didn't Cover Here

- ▷ **Inference by sampling:** A whole zoo of methods for doing this exists.
- ▷ **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.



©: Michael Kohlhase

662



Reading:

- *Chapter 14: Probabilistic Reasoning* of [RN03].
 - Section 14.1 roughly corresponds to my “What is a Bayesian Network?”.
 - Section 14.2 roughly corresponds to my “What is the Meaning of a Bayesian Network?” and “Constructing Bayesian Networks”. The main change I made here is to *define* the semantics of the BN in terms of the conditional independence relations, which I find clearer than RN’s definition that uses the reconstructed full joint probability distribution instead.
 - Section 14.4 roughly corresponds to my “Inference in Bayesian Networks”. RN give full details on variable elimination, which makes for nice ongoing reading.
 - Section 14.3 discusses how CPTs are specified in practice.
 - Section 14.5 covers approximate sampling-based inference.
 - Section 14.6 briefly discusses relational and first-order BNs.
 - Section 14.7 briefly discusses other approaches to reasoning about uncertainty.

All of this is nice as additional background reading.

Chapter 22

Making Simple Decisions Rationally

22.1 Introduction

Decision Theory

- ▷ **Definition 22.1.1** Decision theory investigates how an agent a deals with choosing among actions based on the desirability of their outcomes.
- ▷ **Wait:** Isn't that what we did in Section 7.1 *Problem Solving*?
- ▷ **Yes, but:** now we do it for stochastic (i.e. non-deterministic), partially observable environments.
- ▷ **Recall:** We call an agent environment
 - ▷ 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.
- ▷ We restrict ourselves to episodic decision theory, which deals with choosing among actions based on the desirability of their immediate outcomes.



©: Michael Kohlhase

663



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 variables are the possible outcome 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^+ .
- ▷ **Definition 22.1.2** The **expected utility** $EU(a)$ of an action a (given evidence e) is then

$$EU(a|e) = \sum_{s'} P(R(a) = s' | a, e) \cdot U(s')$$

- ▷ **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**)



©: Michael Kohlhase

664



Outline of this Chapter

- ▷ Rational preferences
- ▷ Utilities and Money
- ▷ Multiattribute utilities
- ▷ Decision networks
- ▷ Value of information



©: Michael Kohlhase

665

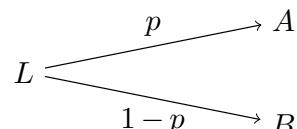


22.2 Rational Preferences

Preferences

- ▷ **Problem:** We cannot directly measure utility of (or satisfaction/happiness in) a state.
- ▷ **Idea:** We can let people choose between two states! (subjective preference)
- ▷ **Problem:** We do not always have full information about the states we choose between
- ▷ **Example 22.2.1 (Airline Food)** *Do you want chicken or pasta* (but we cannot see through the tin foil)
- ▷ **Definition 22.2.2 (Preferences)**

An agent chooses among **prizes** (A , B , etc.) and **lotteries** $L = [p, A; (1 - p), B]$ – i.e., situations with uncertain prizes – by expressing **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*



©: Michael Kohlhase

666



Rational Preferences

- ▷ **Idea:** Preferences of a rational agent must obey constraints:
Rational preferences \leadsto behavior describable as maximization of expected utility

- ▷ **Definition 22.2.3** We call a set of preferences \succ *rational*, iff the following constraints hold:

<i>Orderability</i>	$A \succ B \vee B \succ A \vee A \sim B$
<i>Transitivity</i>	$A \succ B \wedge B \succ C \Rightarrow A \succ C$
<i>Continuity</i>	$A \succ B \succ C \Rightarrow (\exists p.[p, A; (1-p), C] \sim B)$
<i>Substitutability</i>	$A \sim B \Rightarrow [p, A; (1-p), C] \sim [p, B; (1-p), C]$
<i>Monotonicity</i>	$A \succ B \Rightarrow (p > q) \Leftrightarrow [p, A; (1-p), B] \succeq [q, A; (1-q), B]$
<i>Decomposability</i>	$[p, A; (1-p), [q, B; (1-q), C]] \sim [p, A; (1-p), q, B; (1-p)(1-q), C]$



©: Michael Kohlhase

667



The rationality constraints can be understood as follows:

Orderability: $A \succ B \vee B \succ A \vee A \sim B$ Given any two *prizes* or *lotteries*, a rational agent must either prefer one to the other or else rate the two as equally preferable. That is, the agent cannot avoid deciding. Refusing to bet is like refusing to allow time to pass.

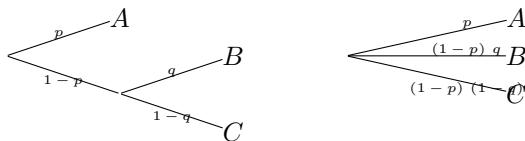
Transitivity: $A \succ B \wedge B \succ C \Rightarrow A \succ C$

Continuity: $A \succ B \succ C \Rightarrow (\exists p.[p, A; (1-p), C] \sim B)$ If some lottery B is between A and C in preference, then there is some probability p for which the rational agent will be indifferent between getting B for sure and the lottery that yields A with probability p and C with probability $1-p$.

Substitutability: $A \sim B \Rightarrow [p, A; (1-p), C] \sim [p, B; (1-p), C]$ If an agent is indifferent between two lotteries A and B , then the agent is indifferent between two more complex lotteries that are the same except that B is substituted for A in one of them. This holds regardless of the probabilities and the other outcome(s) in the lotteries.

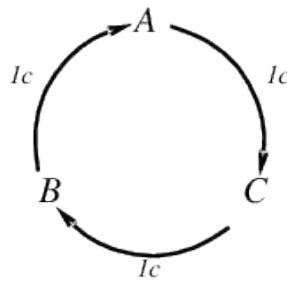
Monotonicity: $A \succ B \Rightarrow (p > q) \Leftrightarrow [p, A; (1-p), B] \succeq [q, A; (1-q), B]$ Suppose two lotteries have the same two possible outcomes, A and B . If an agent prefers A to B , then the agent must prefer the lottery that has a higher probability for A (and vice versa).

Decomposability: $[p, A; (1-p), [q, B; (1-q), C]] \sim [p, A; (1-p), q, B; (1-p)(1-q), C]$ Compound lotteries can be reduced to simpler ones using the laws of probability. This has been called the “no fun in gambling” rule because it says that two consecutive lotteries can be compressed into a single equivalent lottery: the following two are equivalent:



Rational preferences contd.

- ▷ Violating the constraints leads to self-evident irrationality
- ▷ **Example 22.2.4** 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



22.3 Utilities and Money

Ramseys Theorem and Value Functions

- ▷ **Theorem 22.3.1** (*Ramsey, 1931; von Neumann and Morgenstern, 1944*)

Given a rational set of preferences there exists a real-valued function U such that

$$(U(A) \geq U(B)) \Leftrightarrow A \succeq B \quad \text{and} \quad U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$

- ▷ 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 .

- ▷ With deterministic prizes only (no lottery choices), only a total order on prizes can be determined

- ▷ **Definition 22.3.2** We call a total ordering on states a **value function** or **ordinal utility function**.



Maximizing Expected Utility

- ▷ **Definition 22.3.3 (MEU principle)** Choose the action that **maximizes** expected utility (**MEU**)
- ▷ **Note:** an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
- ▷ **Example 22.3.4** 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**)



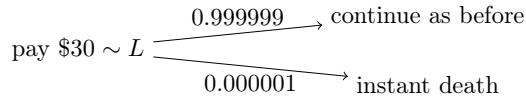
©: Michael Kohlhase

670



Utilities

- ▷ Utilities map states to real numbers. Which numbers?
- ▷ **Definition 22.3.5 (Standard approach to assessment of human utilities)**
Compare a given state A to a **standard lottery** L_p that has
 - ▷ “best possible prize” u_{\top} with probability p
 - ▷ “worst possible catastrophe” u_{\perp} with probability $1 - p$
 adjust lottery probability p until $A \sim L_p$. Then $U(A) = p$.
- ▷ **Example 22.3.6** Choose $u_{\top} \hat{=} \text{current state}$, $u_{\perp} \hat{=} \text{instant death}$



©: Michael Kohlhase

671



Utility scales

- ▷ **Definition 22.3.7 Normalized utilities:** $u_{\top} = 1$, $u_{\perp} = 0$
- ▷ **Definition 22.3.8 Micromorts:** one-millionth chance of 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 370 km incurs a risk of one micromort;

- ▷ over the life of your car – say, 150,000 km that's 400 micromorts.
- ▷ People appear to be willing to pay about €10,000 more for a safer car that halves the risk of death (\leadsto €25 per micromort)
- ▷ This figure has been confirmed across many individuals and risk types.
- ▷ Of course, this argument holds only for small risks. Most people won't agree to kill themselves for €25 million.
- ▷ **Definition 22.3.9 QALYs:** quality-adjusted life years
- ▷ they useful for medical decisions involving substantial risk



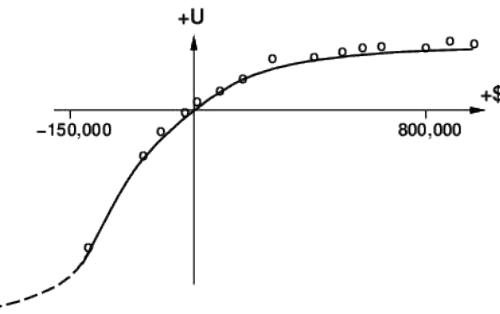
©: Michael Kohlhase

672



Money vs. Utility

- ▷ Money does *not* behave as a utility function
- ▷ Given a lottery L with expected monetary value $EMV(L)$, usually $U(L) < U(EMV(L))$, 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.



©: Michael Kohlhase

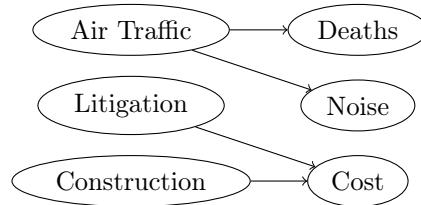
673



22.4 Multi-Attribute Utility

Multi-Attribute Utility

- ▷ How can we handle utility functions of many variables $X_1 \dots X_n$?
- ▷ **Example 22.4.1 (Assessing an Airport Site)**



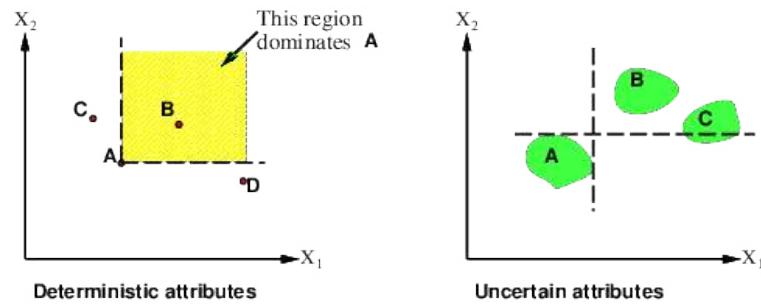
what is
 $U(Deaths, Noise, Cost)$
 for a projected airport?

- ▷ 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 22.4.2** Choice B **strictly dominates** choice A iff $X_i(B) \geq X_i(A)$ for all i (and hence $U(B) \geq U(A)$)



- ▷ 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

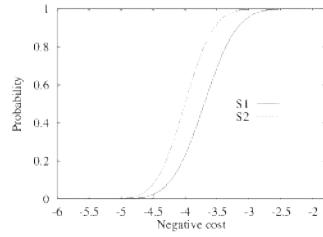
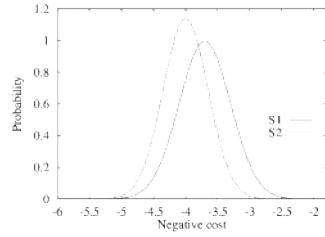


Stochastic Dominance

- ▷ **Definition 22.4.3** Distribution p_2 **stochastically dominates** distribution p_1 iff the **cumulative distribution** of p_2 dominates that for p_1 for all t , i.e.

$$\int_{-\infty}^t p_1(x)dx \leq \int_{-\infty}^t p_2(x)dx$$

▷ Example 22.4.4



Stochastic dominance contd.

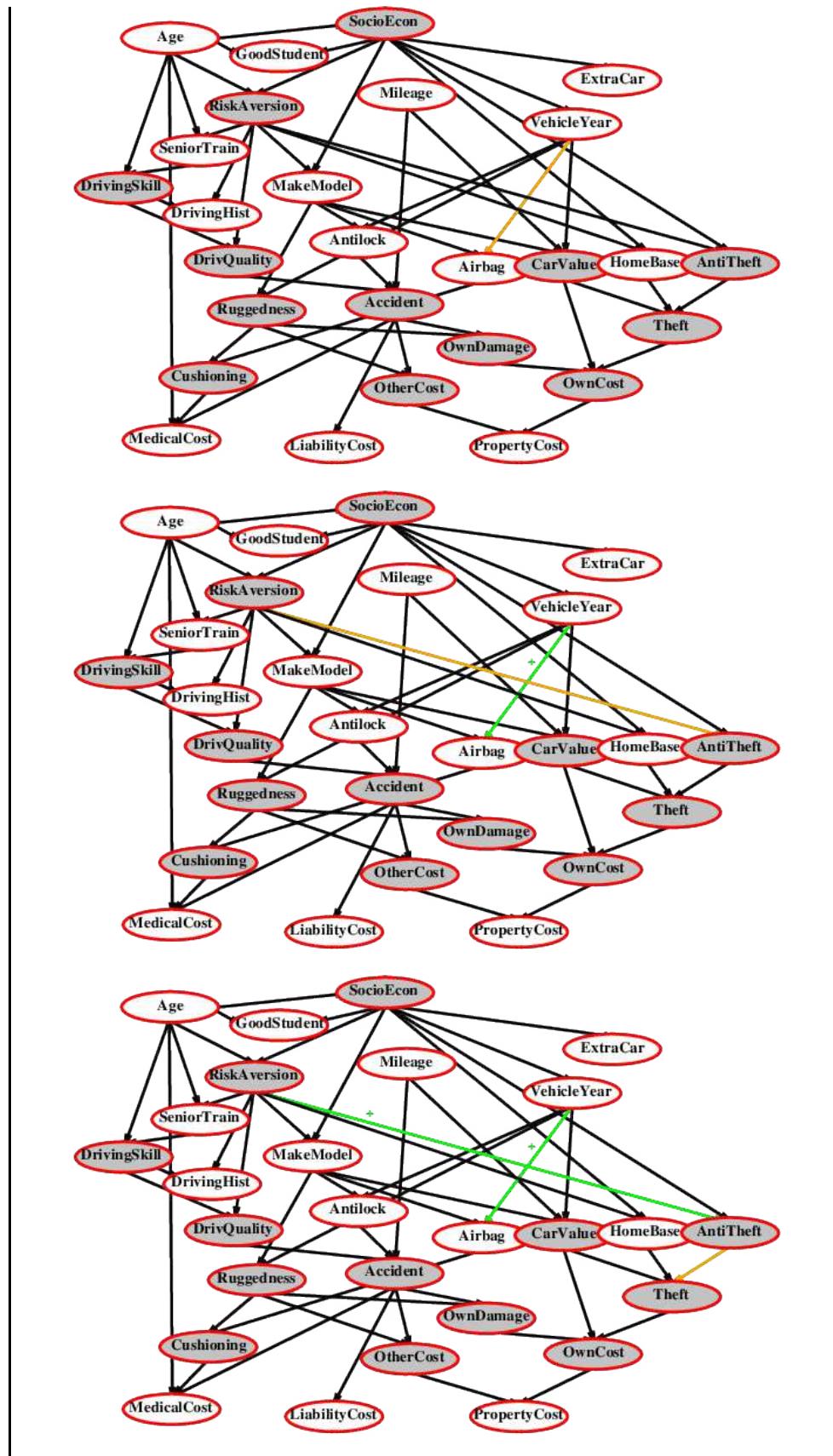
▷ **Observation 22.4.5** If U is monotonic in x , then A_1 with outcome distribution p_1 stochastically dominates A_2 with outcome distribution p_2 :

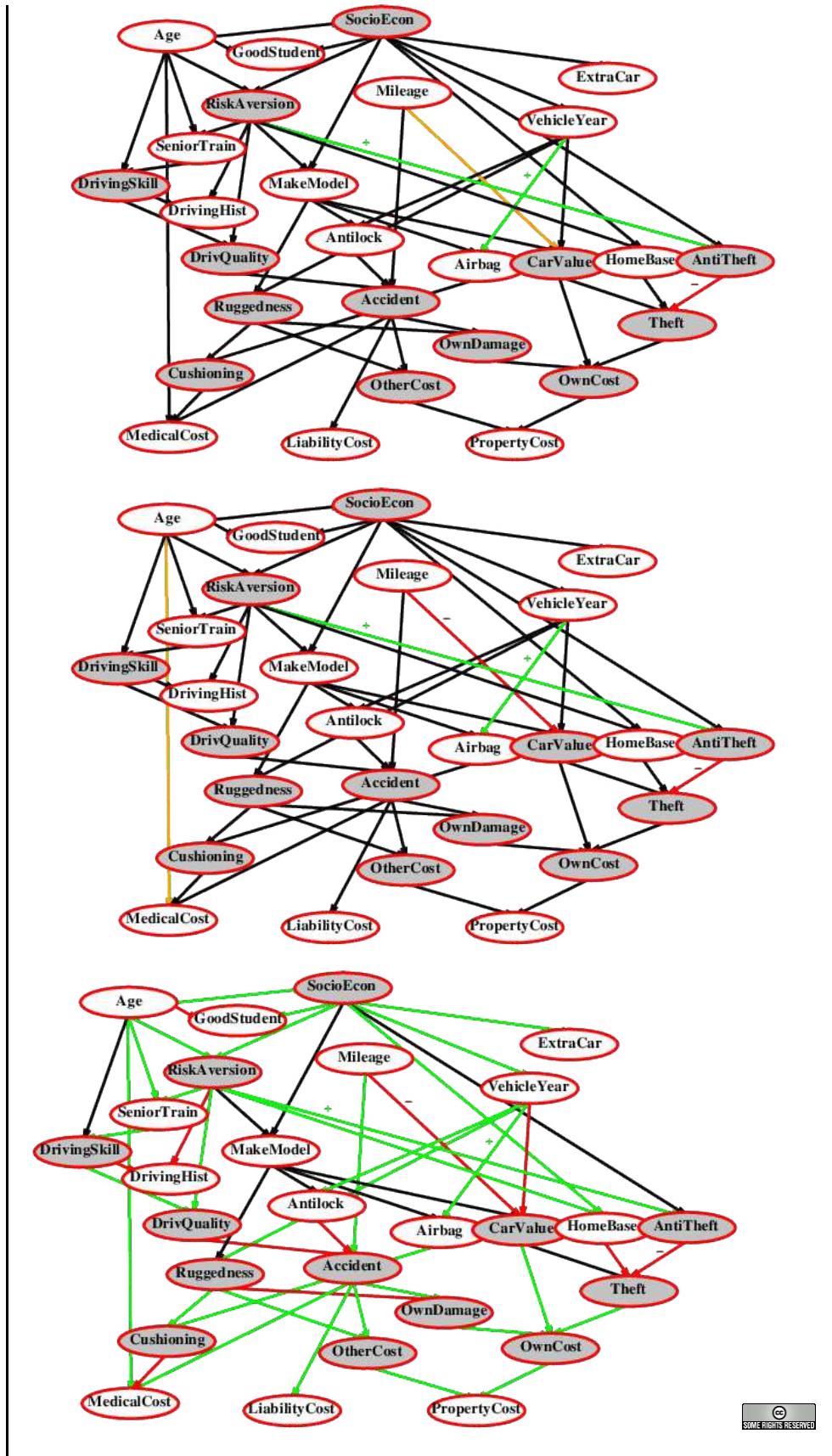
$$\int_{-\infty}^{\infty} p_1(x)U(x)dx \geq \int_{-\infty}^{\infty} p_2(x)U(x)dx$$

- ▷ Multiattribute case: stochastic dominance on all attributes \rightsquigarrow optimal
- ▷ Stochastic dominance can often be determined without exact distributions using qualitative reasoning
- ▷ **Example 22.4.6** Construction cost increases with distance from city S_1 is closer to the city than $S_2 \rightsquigarrow S_1$ stochastically dominates S_2 on cost
- ▷ **Example 22.4.7** Injury increases with collision speed
- ▷ **Idea:** Annotate belief networks with stochastic dominance information.
- ▷ **Definition 22.4.8** $X \xrightarrow{+} Y$ (X positively influences Y) means that $\mathbf{P}(Y | x_1, \mathbf{z})$ stochastically dominates $\mathbf{P}(Y | x_2, \mathbf{z})$ for every value \mathbf{z} of Y 's other parents \mathbf{Z} and all x_1 and x_2 with $x_1 \geq x_2$.



Label the arcs + or - for influence in a Bayesian Network





Preference Structure and Multi-Attribute Utility

▷ **Observation 22.4.9** n attributes with d values each \sim 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.



©: Michael Kohlhase

679



Preference structure: Deterministic

▷ **Recall:** In deterministic environments an agent has a **value function**.

▷ **Definition 22.4.10** 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 22.4.11** E.g., \langle Noise, Cost, 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 22.4.12 (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 22.4.13 (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 22.4.14** The value function for the airport decision might be

$$V(\text{noise}, \text{cost}, \text{deaths}) = -\text{noise} \cdot 10^4 - \text{cost} - \text{deaths} \cdot 10^{12}$$



©: Michael Kohlhase

680



Preference structure: Stochastic

▷ Need to consider preferences over lotteries and real utility functions (not just value functions)

▷ **Definition 22.4.15** \mathbf{X} is **utility-independent** of \mathbf{Y} iff preferences over lotteries in \mathbf{X} do not depend on particular values in \mathbf{Y} .

▷ **Definition 22.4.16** A set X is **mutually utility-independent**, iff each subset is utility-independent of its complement.

▷ **Theorem 22.4.17** For **mutually utility-independent** sets there is a multiplicative utility function: [Kee74]

$$U = k_1 U_1 + k_2 U_2 + k_3 U_3 + k_1 k_2 U_1 U_2 + k_2 k_3 U_2 U_3 + k_3 k_1 U_3 U_1 + k_1 k_2 k_3 U_1 U_2 U_3$$

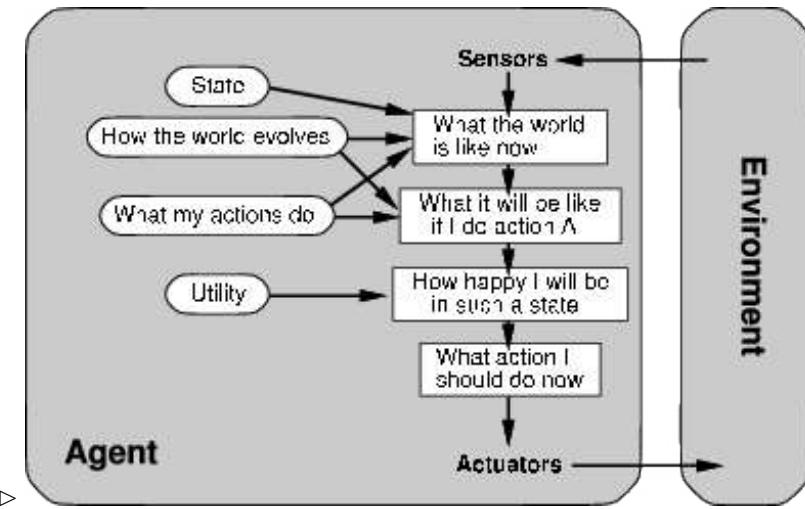
▷ Routine procedures and software packages for generating preference tests to identify various canonical families of utility functions



22.5 Decision Networks

Now that we understand the utilities, we can complete our design of a utility-based agent, which we now recapitulate as a refresher.

Utility-Based Agents (Recap)

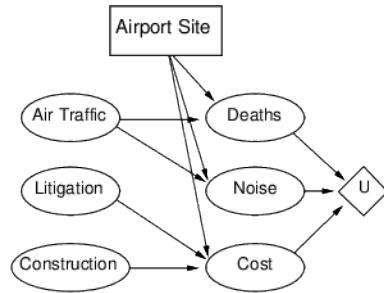


As we already use Bayesian Networks for the world/belief model, integrating utilities and possible actions into the network suggests itself naturally. This leads to the notion of a decision network.

Decision networks

▷ **Definition 22.5.1** Add **action nodes** and **utility nodes** (also called **value nodes**) to belief networks to enable rational decision making.

▷ **Example 22.5.2 (Choosing an Airport Site)**

**Algorithm:**

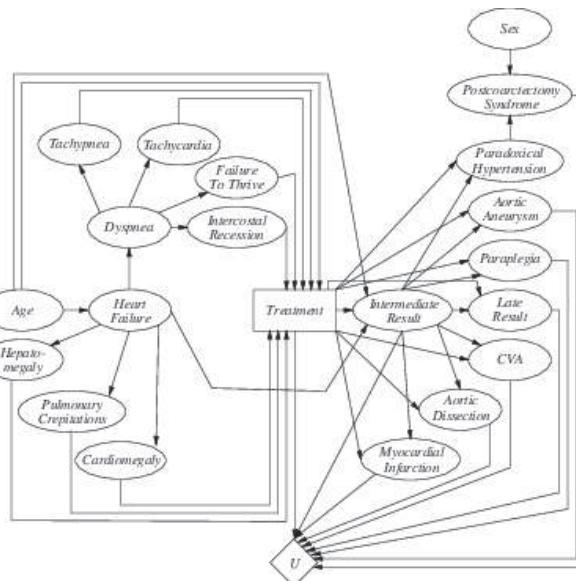
For each value of action node
 compute expected value of utility node given action, evidence
 Return MEU action (via argmax)



©: Michael Kohlhase

683

▷ A Decision-Theoretic Expert System or Aortic Coarctation



©: Michael Kohlhase

684

Knowledge Eng. for Decision-Theoretic Expert Systems

- ▷ Create a causal model
 - ▷ symptoms, disorders, treatments, outcomes, and their influences
- ▷ Simplify to a qualitative decision model
 - ▷ remove vars not involved in treatment decisions

- ▷ Assign probabilities (\rightsquigarrow Bayesian network)
 - ▷ e.g. from patient databases, literature studies, or the expert's subjective assessments
- ▷ Assign utilities (e.g. in QUALYs or micromorts)
- ▷ Verify and refine the model wrt. a gold standard given by experts
 - ▷ refine by "running the model backwards" and compare with the literature
- ▷ Perform sensitivity analysis (important step in practice)
 - ▷ is the optimal treatment decision robust against small changes in the parameters?
(if yes \rightsquigarrow great! if not, collect better data)



©: Michael Kohlhase

685



So far we have tacitly been concentrating on actions that directly affect the environment. We will now come to a type of action we have hypothesized in the beginning of the course, but have completely ignored up to now: information acquisition actions.

22.6 The Value of Information

What if we do not have all information we need?

- ▷ It is Well-Known: that one of the most important parts of decision making is knowing what questions to ask.
- ▷ Example 22.6.1 (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)
 - ▷ 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.

Information value theory enables the agent to make such decisions rationally.

- ▷ 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.



©: Michael Kohlhase

686



Value of Information by Example

- ▷ Idea: compute value of acquiring each possible piece of evidence

Can be done *directly from decision network*

- ▷ **Example 22.6.2 (Buying Oil Drilling Rights)** n blocks of rights, exactly one has oil, worth k
 - ▷ Prior probabilities $1/n$ each, mutually exclusive
 - ▷ Current price of each block is k/n
 - ▷ “Consultant” offers accurate survey of block 3. Fair price?

Solution: compute expected value of information = expected value of best action given the information minus expected value of best action without information

▷ **Example 22.6.3 (Oil Drilling Rights contd.)**

- ▷ Survey may say “oil in block 3”, prob. $1/n \rightsquigarrow$ buy block 3 for k/n make profit of $k - k/n$.
- ▷ Survey may say “no oil in block 3” prob. $(n-1)/n \rightsquigarrow$ buy another block make profit of $k/(n-1) - k/n$.
- ▷ Expected profit is $\frac{1}{n} \cdot \frac{(n-1)k}{n} + \frac{n-1}{n} \cdot \frac{k}{n(n-1)} = \frac{k}{n}$
- ▷ we should pay up to k/n for the information (as much as block 3 is worth)



General formula (VPI)

- ▷ Current evidence E , current best action α

- ▷ Possible action outcomes S_i :

$$EU(\alpha | E) = \max_a \sum_i U(S_i) \cdot P(S_i | E, a)$$

- ▷ Suppose we knew $E_j = e_{jk}$ (new evidence), then we would choose $\alpha_{e_{jk}}$ s.t.

$$EU(\alpha_{e_{jk}} | E, E_j = e_{jk}) = \max_a \sum_i U(S_i) \cdot P(S_i | E, a, E_j = e_{jk})$$

E_j is a random variable whose value is *currently unknown*

- ▷ So we must compute expected gain over all possible values:

$$VPI_E(E_j) = \sum_k P(E_j = e_{jk} | E) \cdot EU(\alpha_{e_{jk}} | E, E_j = e_{jk}) - EU(\alpha | E)$$

- ▷ **Definition 22.6.4** VPI $\hat{=}$ value of perfect information



Properties of VPI

- ▷ **Nonnegative:** in *expectation*, not *post hoc*: $VPI_E(E_j) \geq 0$ for all j and E
- ▷ **Nonadditive:** consider, e.g., obtaining E_j twice

$$VPI_E(E_j, E_k) \neq VPI_E(E_j) + VPI_E(E_k)$$

- ▷ **Order-independent:**

$$VPI_E(E_j, E_k) = VPI_E(E_j) + VPI_{E, E_j}(E_k) = VPI_E(E_k) + VPI_{E, E_k}(E_j)$$

- ▷ **Note:** when more than one piece of evidence can be gathered, maximizing VPI for each to select one is not always optimal
 \leadsto evidence-gathering becomes a *sequential* decision problem



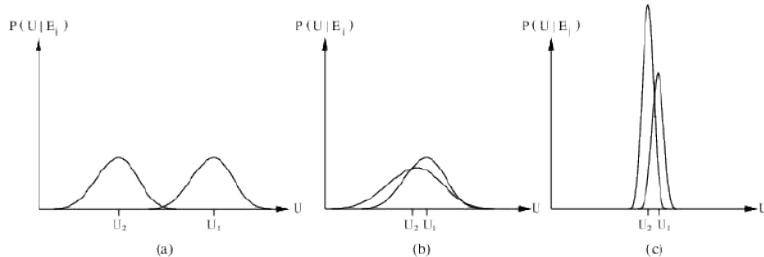
©: Michael Kohlhase

689



Questionnaire: Qualitative behaviors

- ▷ **Question:** Say we have three distributions for $P(U | E_j)$



What is the value of information in these three cases?

- ▷ **Answers:** qualitatively

- a) Choice is obvious (U_1 almost certainly better) \leadsto information worth little
- b) Choice is non-obvious (unclear) \leadsto information worth a lot
- c) Choice is non-obvious (unclear) **but** makes little difference \leadsto information worth little

The fact that U_2 has a high peak in (c) means that its expected value is known with higher certainty than U_1 .
(irrelevant to the argument)



©: Michael Kohlhase

690



We will now use information value theory to specialize our utility-based agent from above.

A simple Information-Gathering Agent

▷ **Definition 22.6.5** A simple information gathering agent (gathers info before acting)

```

function Information–Gathering–Agent (percept) returns an action
  persistent:  $D$ , a decision network
  integrate percept into  $D$ 
   $j := \underset{k}{\operatorname{argmax}}(\text{VPI}_E(E_k)/\text{Cost}(E_k))$ 
  if  $\text{VPI}_E(E_j) > \text{Cost}(E_j)$  return  $\text{Request}(E_j)$ 
  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

Outline

- ▷ Time and uncertainty
- ▷ Inference: filtering, prediction, smoothing
- ▷ Hidden Markov models
- ▷ Dynamic Bayesian networks
- ▷ Particle filtering
- ▷ Further Algorithms and Topics



©: Michael Kohlhase

692



23.1 Modeling Time and Uncertainty

Time and uncertainty

- ▷ **Observation 23.1.1** *The world changes; we need to track and predict it*
- ▷ **Example 23.1.2** Diabetes management vs. vehicle diagnosis
- ▷ **Definition 23.1.3** 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)
- ▷ **Basic idea:** index random variables by \mathbb{N} .
 - ▷ \mathbf{X}_t = set of unobservable state variables at time t
e.g., BloodSugar_t , StomachContents_t , etc.
 - ▷ \mathbf{E}_t = set of observable evidence variables at time t
e.g., $\text{MeasuredBloodSugar}_t$, PulseRate_t , FoodEaten_t

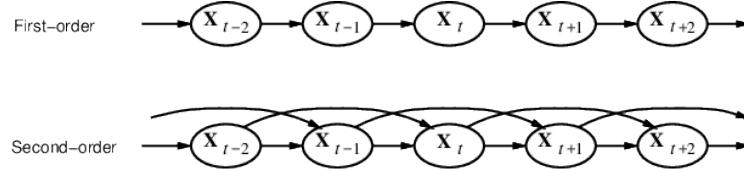
▷ **Example 23.1.4 (Umbrellas)** You are a security guard in a secret underground facility, want to know if it is raining outside. Your only source of information is whether the director comes in with an umbrella.
State variables R_1, R_2, R_3, \dots , Observations U_1, U_2, U_3, \dots

▷ **Notation:** $\mathbf{X}_{a:b} = \mathbf{X}_a, \mathbf{X}_{a+1}, \dots, \mathbf{X}_{b-1}, \mathbf{X}_b$



Markov Processes

- ▷ Construct a Bayesian network from these variables: parents?
- ▷ **Definition 23.1.5 Markov property:** \mathbf{X}_t only depends on a *bounded* subset of $\mathbf{X}_{0:t-1}$.
- ▷ **Definition 23.1.6** A (discrete-time) **Markov process** (also called **Markov chain**) is a sequence of random variables with the Markov property.
- ▷ **Definition 23.1.7 First-order Markov process:** $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1})$



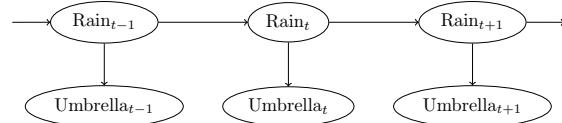
Second-order Markov process: $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-2}, \mathbf{X}_{t-1})$

- ▷ We will use Markov processes to model sequential environments.



Markov Process Example: The Umbrella

- ▷ **Example 23.1.8 (Umbrellas continued)** We model the situation in a Bayesian network:



Problem: First-order Markov assumption not exactly true in real world!

▷▷ **Possible fixes:**

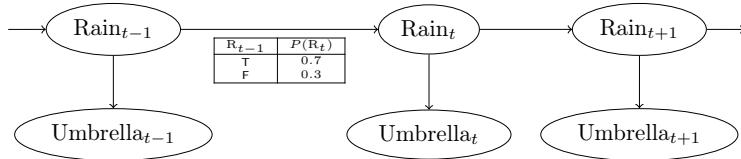
1. *Increase order* of Markov process
2. *Augment state*, e.g., add Temp_t , Pressure_t

- ▷ **Example 23.1.9 (Robot Motion)** Augment Position_{*t*} and Velocity_{*t*} with Battery_{*t*}



Stationary Markov Processes as Transition Models

- ▷ **Definition 23.1.10** We divide the random variables in a Markov process M into a set of (hidden) **state variables** \mathbf{X}_t and a set of (observable) **evidence variables** \mathbf{E}_t . We call $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1})$ the **transition model** and $\mathbf{P}(\mathbf{E}_t | \mathbf{E}_{t-1})$ the **sensor model** of M .
 - ▷ **Problem:** Even with **Markov assumption** the transition model is infinite ($t \in \mathbb{N}$)
 - ▷ **Definition 23.1.11** A **Markov process** is called **stationary** if the transition model is independent of time, i.e. $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1})$ is the same for all t .
 - ▷ **Example 23.1.12 (Umbrellas are stationary)** $\mathbf{P}(\mathbf{R}_t | \mathbf{R}_{t-1})$ does not depend on t (need only one table)



- ▷ Don't confuse "stationary" (processes) with "static" (environments).
 - ▷ We restrict ourselves to stationary [Markov processes](#) in this course.



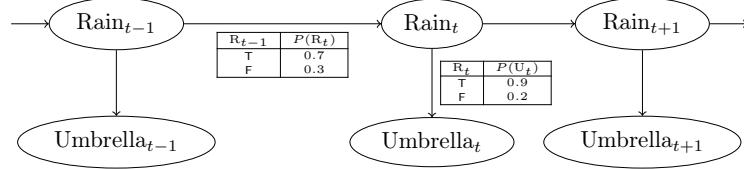
Markov Sensor Models

- ▷ **Recap:** The **sensor model** predicts the influence of percepts (and the world state) on the belief state (used during update)
 - ▷ **Problem:** The evidence variables \mathbf{E}_t could depend on previous variables as well as the current state.
 - ▷ we restrict dependency to current state (otherwise state repr. deficient)
 - ▷ **Definition 23.1.13** We say that a sensor model has the **sensor Markov property**, iff $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$
 - ▷ **Assumptions on Sensor Models:** We usually assume the sensor Markov property and make it stationary as well: $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$ is fixed for all t .



Umbrellas, the full Story

- ▷ Example 23.1.14 (Umbrellas, Transition & Sensor Models)



Note that influence goes from Rain_t to Umbrella_t (causal)

- ▷ Observation 23.1.15 If we additionally know the initial prior probabilities $\mathbf{P}(\mathbf{X}_0)$ ($\hat{=}$ time $t = 0$), then we can compute the full joint probability distribution as

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{0:t}) = \mathbf{P}(\mathbf{X}_0) \cdot \prod_{i=1}^t \mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1}) \cdot \mathbf{P}(\mathbf{E}_i | \mathbf{X}_i)$$



23.2 Inference: Filtering, Prediction, and Smoothing

Inference tasks

- ▷ Definition 23.2.1 Filtering (or monitoring): $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$
computing the belief state – input to the decision process of a rational agent.
- ▷ Definition 23.2.2 Prediction (or state estimation): $\mathbf{P}(\mathbf{X}_{t+k} | \mathbf{e}_{1:t})$ for $k > 0$
evaluation of possible action sequences. ($\hat{=}$ filtering without the evidence)
- ▷ Definition 23.2.3 Smoothing (or hindsight): $\mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:t})$ for $0 \leq k < t$
better estimate of past states (essential for learning)
- ▷ Definition 23.2.4 Most likely explanation $\underset{\mathbf{x}_{1:t}}{\operatorname{argmax}}(P(\mathbf{x}_{1:t} | \mathbf{e}_{1:t}))$
speech recognition, decoding with a noisy channel.



Filtering

- ▷ Aim: recursive state estimation: $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t}))$

▷ Project the current distribution forward from t to $t + 1$:

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) && (\text{dividing up evidence}) \\ &= \alpha \cdot \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \cdot \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) && (\text{using Bayes' rule}) \\ &= \alpha \cdot \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \cdot \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) && (\text{sensor Markov assumption})\end{aligned}$$

▷ Note that $\mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1})$ can be obtained directly from the sensor model

▷ Continue by conditioning on the current state \mathbf{X}_t :

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= \alpha \cdot \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \cdot \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t, \mathbf{e}_{1:t}) \cdot P(\mathbf{x}_t | \mathbf{e}_{1:t}) \\ &= \alpha \cdot \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \cdot \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) \cdot P(\mathbf{x}_t | \mathbf{e}_{1:t})\end{aligned}$$

▷ $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{X}_t)$ is simply the transition model, $P(\mathbf{x}_t | \mathbf{e}_{1:t})$ the “recursive call”.

▷ So $f_{1:t+1} = \alpha \cdot \text{FORWARD}(f_{1:t}, e_{t+1})$ where $f_{1:t} = \mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$ and FORWARD is the update shown above. (Time and space constant (independent of t))



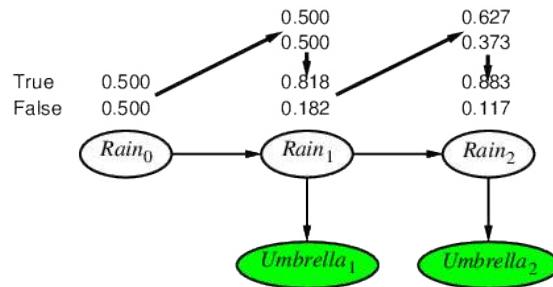
Filtering the Umbrellas

▷ **Example 23.2.5** Say the guard believes $\mathbf{P}(R_0) = \langle 0.5, 0.5 \rangle$. On day 1 and 2 the umbrella appears.

$$\mathbf{P}(R_1) = \sum_{r_0} \mathbf{P}(R_1 | r_0) \cdot P(r_0) = \langle 0.7, 0.3 \rangle \cdot 0.5 + \langle 0.3, 0.7 \rangle \cdot 0.5 = \langle 0.5, 0.5 \rangle$$

update with evidence for $t = 1$ gives

$$\mathbf{P}(R_1 | u_1) = \alpha \cdot \mathbf{P}(u_1 | R_1) \cdot \mathbf{P}(R_1) = \alpha \cdot \langle 0.9, 0.2 \rangle \langle 0.5, 0.5 \rangle = \alpha \cdot \langle 0.45, 0.1 \rangle \approx \langle 0.818, 0.182 \rangle$$



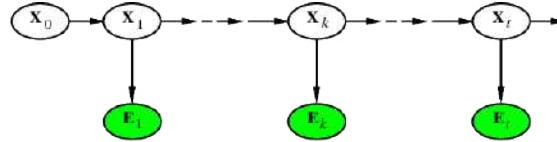
Prediction

- ▷ Prediction computes future $k > 0$ state distributions: $\mathbf{P}(\mathbf{X}_{t+k} | \mathbf{e}_{1:t})$
- ▷ **Intuition:** Prediction is filtering without new evidence.
- ▷ **Lemma 23.2.6** $\mathbf{P}(\mathbf{X}_{t+k+1} | \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \mathbf{P}(\mathbf{X}_{t+k+1} | \mathbf{x}_{t+k}) \cdot P(\mathbf{x}_{t+k} | \mathbf{e}_{1:t})$
- ▷ **Proof Sketch:** Using the same reasoning as for the FORWARD algorithm for filtering. \square
- ▷ **Observation 23.2.7** As $k \rightarrow \infty$, $P(\mathbf{x}_{t+k} | \mathbf{e}_{1:t})$ 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 $\mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:t})$ for $0 \leq k < t$



- ▷ Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$ (before k) and $\mathbf{e}_{k+1:t}$ (after k):

$$\begin{aligned}
 \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
 &= \alpha \cdot \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}) \cdot \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{e}_{1:k}) \quad (\text{Bayes Rule}) \\
 &= \alpha \cdot \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}) \cdot \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k) \quad (\text{cond. independence}) \\
 &= \alpha \cdot \mathbf{f}_{1:k} \cdot \mathbf{b}_{k+1:t}
 \end{aligned}$$



Smoothing (continued)

- ▷ Backward message $\mathbf{b}_{k+1:t} = \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k)$ computed by a backwards recur-

sion:

$$\begin{aligned}
 \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{x}_{k+1}) \cdot \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k) \\
 &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{x}_{k+1}) \cdot \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k) \\
 &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) \cdot \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k) \\
 &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) \cdot P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) \cdot \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k)
 \end{aligned}$$

$P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1})$ and $\mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k)$ can be directly obtained from the model, $P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1})$ is the “recursive call” ($\mathbf{b}_{k+2:t}$).

▷ In message notation: $\mathbf{b}_{k+1:t} = \text{BACKWARD}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1:t})$ where BACKWARD is the update shown above. (Time and space **constant** (independent of t))



Smoothing example

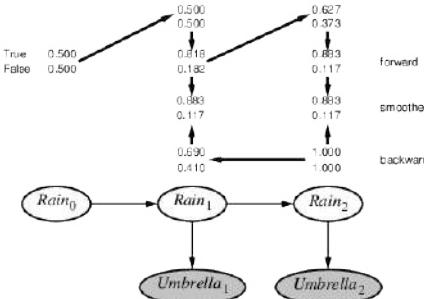
▷ **Example 23.2.8 (Smoothing Umbrellas)** Umbrella appears on days 1/2.

$$\triangleright \mathbf{P}(R_1 | u_1, u_2) = \alpha \cdot \mathbf{P}(R_1 | u_1) \cdot \mathbf{P}(u_2 | R_1) = \alpha \cdot \langle 0.818, 0.182 \rangle \cdot \mathbf{P}(u_2 | R_1)$$

▷ compute $\mathbf{P}(u_2 | R_1)$ by backwards recursion:

$$\begin{aligned}
 \mathbf{P}(u_2 | R_1) &= \sum_{r_2} P(u_2 | r_2) \cdot P(r_2 | R_1) \cdot \mathbf{P}(r_2 | R_1) \\
 &= 0.9 \cdot 1 \cdot \langle 0.7, 0.3 \rangle + 0.2 \cdot 1 \cdot \langle 0.3, 0.7 \rangle = \langle 0.69, 0.41 \rangle
 \end{aligned}$$

$$\triangleright \text{So } \mathbf{P}(R_1 | u_1, u_2) = \alpha \cdot \langle 0.818, 0.182 \rangle \cdot \langle 0.69, 0.41 \rangle \approx 0.883, 0.117$$



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 23.2.9 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 linear in *t* (polytree inference), space $\mathcal{O}(t \cdot \#(\mathbf{f}))$



Most Likely Explanation

- ▷ **Observation 23.2.10** *Most likely sequence* \neq *sequence of most likely states!*
- ▷ **Example 23.2.11** 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 \mathbf{x}_{t+1} = most likely path to *some* \mathbf{x}_t plus one more step
$$\begin{aligned} & \max_{\mathbf{x}_1, \dots, \mathbf{x}_t} \mathbf{P}(\mathbf{x}_1, \dots, \mathbf{x}_t, \mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) \\ &= \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) \cdot \max_{\mathbf{x}_t} (\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t) \cdot \max_{\mathbf{x}_1, \dots, \mathbf{x}_{t-1}} P(\mathbf{x}_1, \dots, \mathbf{x}_{t-1}, \mathbf{x}_t \mid \mathbf{e}_{1:t})) \end{aligned}$$
- ▷ Identical to filtering, except $\mathbf{f}_{1:t}$ replaced by

$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_1, \dots, \mathbf{x}_{t-1}} \mathbf{P}(\mathbf{x}_1, \dots, \mathbf{x}_{t-1}, \mathbf{X}_t \mid \mathbf{e}_{1:t})$$

I.e., $\mathbf{m}_{1:t}(i)$ gives the probability of the most likely path to state i .
Update has sum replaced by max, giving the **Viterbi algorithm**:

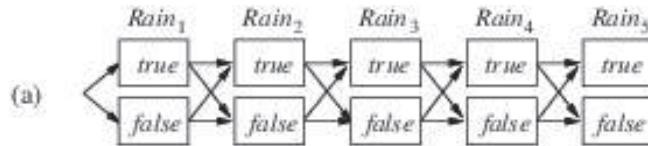
$$\mathbf{m}_{1:t+1} = \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \cdot \max_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t \mathbf{m}_{1:t})$$

- ▷ **Observation 23.2.12** Viterbi has linear time complexity (like filtering), but linear space complexity (needs to keep a pointer to most likely sequence leading to each state).

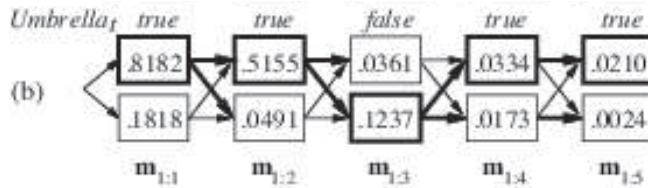


Viterbi example

- ▷ **Example 23.2.13 (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 $\mathbf{m}_{1:t}$ (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 $\mathbf{m}_{1:5}$.



23.3 Hidden Markov Models

The preceding section developed algorithms for temporal probabilistic reasoning using a general framework that was independent of the specific form of the transition and sensor models. In this Section, we discuss more concrete models and applications that illustrate the power of the basic algorithms and implementation issues

Hidden Markov Models

- ▷ **Definition 23.3.1** A **hidden Markov model (HMM)** is a temporal probabilistic model in which the state of the process is described by a single discrete random variable X_t with domain $\{1, \dots, S\}$.
- ▷ **Example 23.3.2** The Example 23.1.4 (the umbrella example) is a HMM.
- ▷ **Intuition:** The possible values of X_t are the possible states of the world.
- ▷ **Observation:** Transition model $P(X_t | X_{t-1}) \cong$ a single $S \times S$ matrix.
- ▷ **Definition 23.3.3** **Transition matrix** $\mathbf{T}_{ij} = P(X_t = j | X_{t-1} = i)$
- ▷ **Example 23.3.4 (Umbrellas)** $\mathbf{T} = P(X_t | X_{t-1}) = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$.
- ▷ **Definition 23.3.5** **Sensor matrix** \mathbf{O}_t for each time step \cong diagonal matrix with $\mathbf{O}_{tii} = P(e_t | X_t = i)$.
- ▷ **Example 23.3.6 (Umbrellas)** With $U_1 = T$ and $U_3 = F$ we have

$$\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix} \quad \text{and} \quad \mathbf{O}_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix}$$



HMM Algorithm

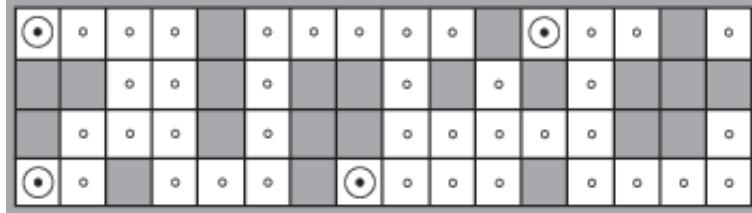
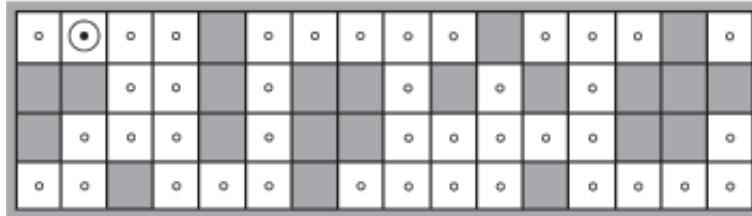
- ▷ **Idea:** Forward and backward messages are column vectors in HMMs
- ▷ **Definition 23.3.7**

HMM filtering equation: $\mathbf{f}_{1:t+1} = \alpha \cdot (\mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t})$
 HMM smoothing equation: $\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}$
- ▷ **Observation 23.3.8** *Forward-backward algorithm needs time $\mathcal{O}(S^2 t)$ and space $\mathcal{O}(S t)$*



Example: Robot Localization using Common Sense

- ▷ **Example 23.3.9 (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 23.3.10 (Filter out Impossible States)**

a) Possible robot locations after $e_1 = \text{NSW}$ b) Possible robot locations after $e_1 = \text{NSW}$ and $e_2 = \text{NS}$

▷ **Remark 23.3.11** This only works for perfect sensors (else no impossible states)



HMM Example: Robot Localization (Modeling)

▷ **Example 23.3.12 (HMM-based Robot Localization)**

- ▷ Random variable X_t for robot location (domain: 42 empty squares)
- ▷ Transition table for the move action: (\mathbf{T} has $42^2 = 1764$ entries)

$$P(X_{t+1} = j | X_t = i) = \mathbf{T}_{ij} = \begin{cases} \frac{1}{\#(N(i))} & \text{if } j \in N(i) \\ 0 & \text{else} \end{cases}$$

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 $n = 42$)
- ▷ Sensor 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.

$$P(E_t = e_t | X_t = i) = \mathbf{O}_{tii} = (1 - \epsilon)^{4-d_{it}} \cdot \epsilon^{d_{it}}$$

- ▷ 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 HMM filtering equation $\mathbf{f}_{1:t+1} = \alpha \cdot (\mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t})$ for localization.
(next)



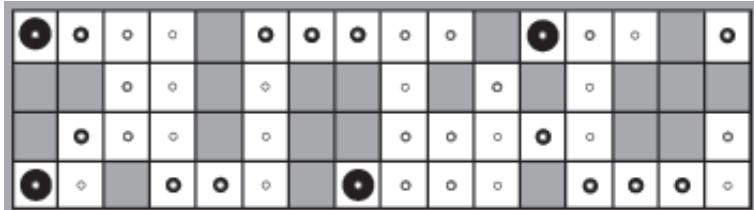
HMM Example: Robot Localization

- ▷ **Idea:** Use HMM filtering equation $\mathbf{f}_{1:t+1} = \alpha \cdot (\mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t})$ to compute pos-

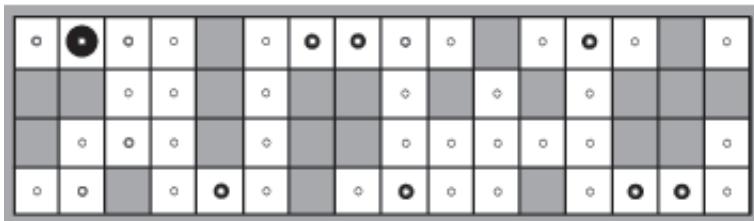
terior distribution over locations
localization)

(i.e. robot

▷ **Example 23.3.13** Redoing Example 23.3.9, with $\epsilon = 0.2$.



a) Posterior distribution over robot location after $E_1 = \text{NSW}$



b) Posterior distribution over robot location after $E_1 = \text{NSW}$ and $E_2 = \text{NS}$

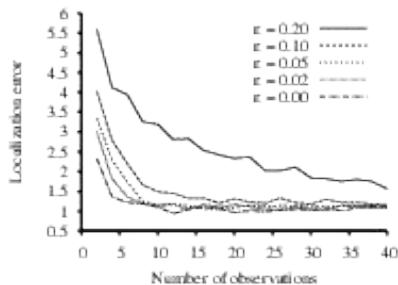
Still the same locations as in the “perfect sensing” case, but now other locations have non-zero probability.



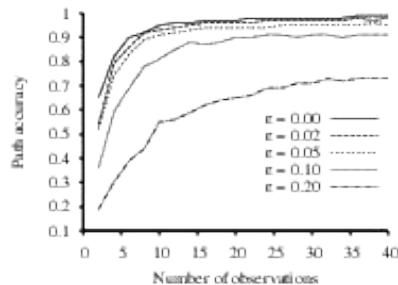
HMM Example: Further Inference Applications

▷ **Idea:** Use smoothing: $\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}$ to find out where it started and the Viterbi algorithm to find the most likely path it took.

▷ **Example 23.3.14** 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)

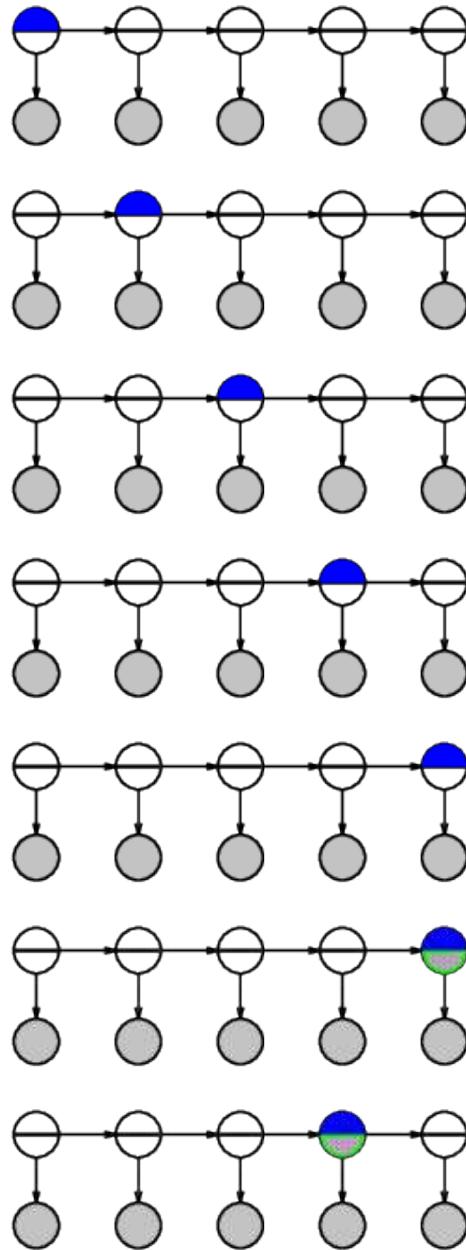


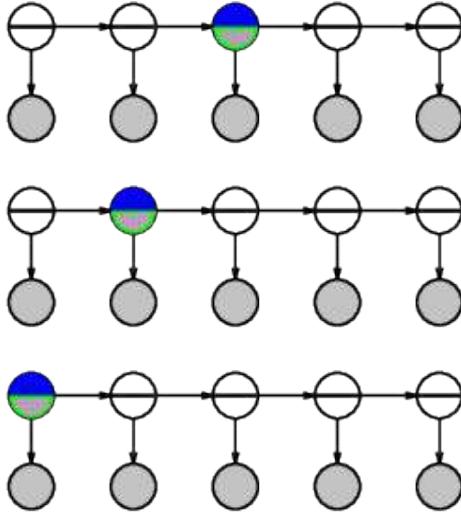
Country dance algorithm

- ▷ Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \cdot (\mathbf{O}_{t+1} \mathbf{T}^t \mathbf{f}_{1:t}) \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \cdot (\mathbf{T}^t \mathbf{f}_{1:t}) \\ \alpha' (\mathbf{T}^{t-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1}) &= \mathbf{f}_{1:t} \end{aligned}$$

- ▷ Algorithm: forward pass computes $\mathbf{f}_{1:t}$, backward pass does $\mathbf{f}_{1:i}, \mathbf{b}_{t-i:t}$.





Observation: backwards pass only needs to store one copy of $\mathbf{f}_{1:i}$, $\mathbf{b}_{t:t-i} \sim$ constant space.

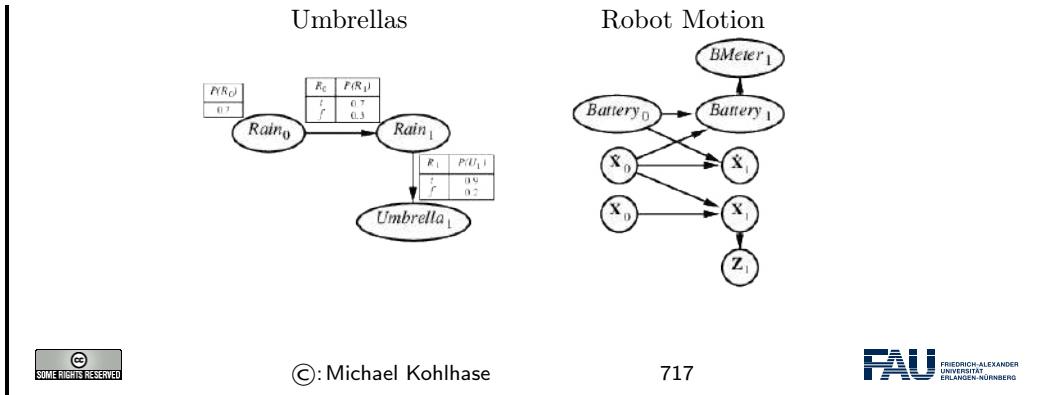
▷ **Problem:** Algorithm is severely limited: transition matrix must be invertible and sensor matrix cannot have zeroes – that is, that every observation be possible in every state.



23.4 Dynamic Bayesian Networks

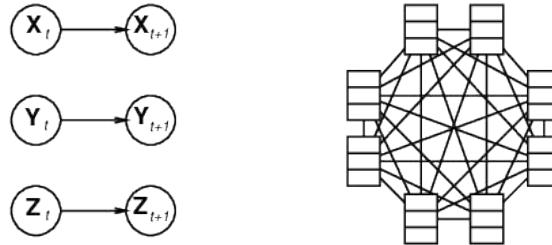
Dynamic Bayesian networks

- ▷ **Definition 23.4.1** A Bayesian network D is called **dynamic** (a **DBN**), iff its random variables are indexed by a time structure. We assume that its structure is
 - ▷ **time sliced**, i.e. that the **time slices** D_t – the subgraphs of t -indexed random variables and the edges between them – are isomorphic.
 - ▷ a first-order Markov process, i.e. that variables X_t can only have parents in D_t and D_{t-1} .
 - ▷ \mathbf{X}_t , \mathbf{E}_t contain arbitrarily many variables in a replicated Bayes net
 - ▷ **Example 23.4.2**



DBNs vs. HMMs

- ▷ **Observation 23.4.3** ▷ Every HMM is a single-variable DBN (trivially)
- ▷ Every discrete DBN is an HMM (combine variables into tuple)
- ▷ DBNs have sparse dependencies \leadsto exponentially fewer parameters;



- ▷ **Example 23.4.4 (Sparse Dependencies)**
- With 20 state variables, three parents each, a DBN has $20 \cdot 2^3 = 160$ parameters, the corresponding HMM has $2^{20} \cdot 2^{20} \approx 10^{12}$.

Exact inference in DBNs

- ▷ **Definition 23.4.5 (Naive method)** Unroll the network and run any exact algorithm



Problem: inference cost for each update grows with t

- ▷ **Definition 23.4.6 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 $\mathcal{O}(d^{2n})$)



Summary

- ▷ Temporal models use state and sensor variables replicated over time
- ▷ Markov assumptions and stationarity assumption, so we need
 - ▷ transition model $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1})$
 - ▷ sensor model $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$
- ▷ 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)
- ▷ Dynamic Bayes nets subsume HMMs, exact update intractable



Chapter 24

Making Complex Decisions

We will now pick up the thread from Chapter 22 but using temporal models instead of simply probabilistic ones. We will first look at a sequential decision theory in the special case, where the environment is stochastic, but fully observable ([Markov decision procedures](#)) and then lift that to obtain [POMDPs](#) and present an agent design based on that.

Outline

- ▷ Markov Decision Problems (for Sequential Environments)
- ▷ Value/Policy iteration for computing utilities in MDPs
- ▷ Partially Observable MDPs (POMDPs)
- ▷ Decision-theoretic agents for POMDPs



©: Michael Kohlhase

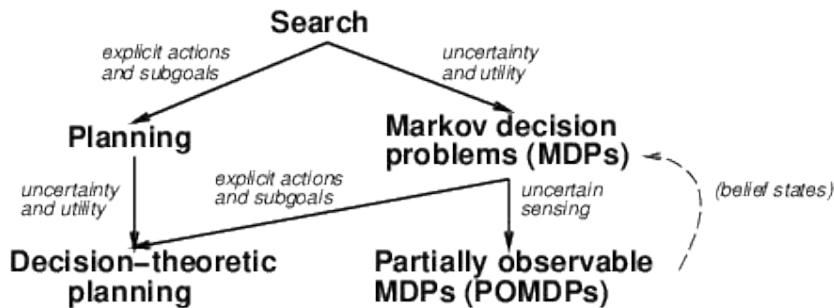
721



24.1 Sequential Decision Problems

Sequential decision problems

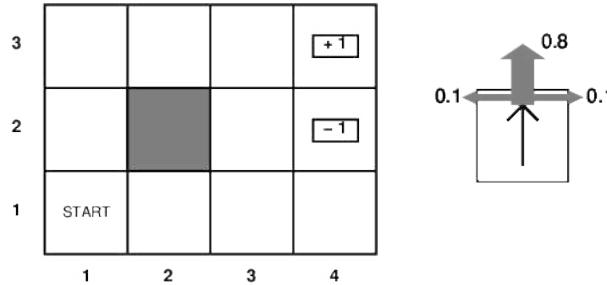
- ▷ In sequential decision problems, the agent's utility depends on a sequence of decisions.
- ▷ Sequential decision problems incorporate utilities, uncertainty, and sensing
- ▷ search and planning problems are special cases





Example MDP

- ▷ Example 24.1.1 A (fully observable) environment with uncertain 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

$$R(s, a, s') := \begin{cases} -0.04 & \text{if (small penalty) for nonterminal states} \\ \pm 1 & \text{if for terminal states} \end{cases}$$



Markov Decision Process

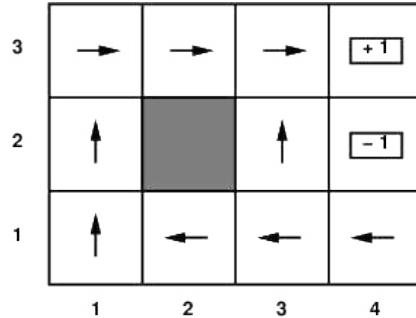
- ▷ Definition 24.1.2 A sequential decision problem in a fully observable, stochastic environment with a Markovian transition model and an additive reward function is called a **Markov decision process (MDP)**. It consists of
 - ▷ a set of **states** (with initial state $s_0 \in S$),
 - ▷ sets **Actions(s)** 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, 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)
- ▷ The optimal policy maximizes (say) the expected sum of rewards

▷ **Example 24.1.3** Optimal policy when state penalty $R(s)$ is -0.04 :

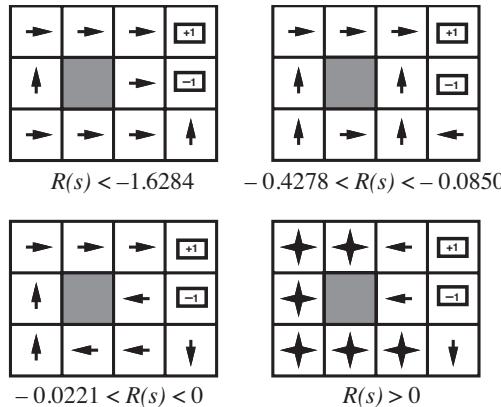


Note: When you run against a wall, you stay in your square.



Risk and Reward (Example and Questionnaire)

▷ **Example 24.1.4** Optimal policy depends on the reward $R(s)$ on non-terminals



Explanation: Explain what you see

$-\infty \leq R(s) \leq -1.6284 \rightsquigarrow$ Life is so painful that agent heads for the next exit.

$-0.4278 \leq R(s) \leq -0.0850$, life is quite unpleasant; the agent takes the shortest route to the $+1$ state and is willing to risk falling into the -1 state by accident. In particular, the agent takes the shortcut from (3,1).

Life is slightly dreary ($-0.0221 < R(s) < 0$) \rightsquigarrow take no risks at all. In (4,1) and (3,2) head directly away from the -1 \rightsquigarrow cannot fall in by accident.

If $R(s) > 0$, then life is positively enjoyable \rightsquigarrow avoid both exits \rightsquigarrow reap infinite rewards.
The optimal policy depends on the reward function

▷ **Remark:** The careful balancing of risk and reward is characteristic of MDPs.



Utility of state sequences

- ▷ Need to understand preferences between **sequences** of states
- ▷ **Definition 24.2.1** 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 24.2.2** For stationary preferences, there are only two ways to combine rewards over time.

- ▷ **additive rewards:** $U([s_0, s_1, \dots]) = R(s_0) + R(s_1) + \dots$
- ▷ **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 the **discount factor**



©: Michael Kohlhase

727



Utilities of State Sequencers (contd.)

- ▷ Problem: infinite lifetimes \leadsto **additive** utilities become infinite
 1. Finite horizon: termination at a **fixed time** $T \leadsto$ **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



©: Michael Kohlhase

728



Utility of States

- ▷ **Intuition:** Utility of a state $\hat{=} \text{expected (discounted) sum of rewards (until termination) assuming optimal actions}$
- ▷ **Definition 24.2.3** 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

$$U^\pi(s) := E \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right]$$

we define the $\pi_s^* := \operatorname{argmax}_{\pi}(U^\pi(s))$.

▷ **Observation 24.2.4** π_s^* 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 24.2.5** We call $\pi^* := \pi_s^*$ for some s the **optimal policy**.

▷ **⚠**: This does not hold for finite-horizon policies.

▷ **Definition 24.2.6** The **utility** $U(s)$ of a state s is $U^{\pi^*}(s)$.



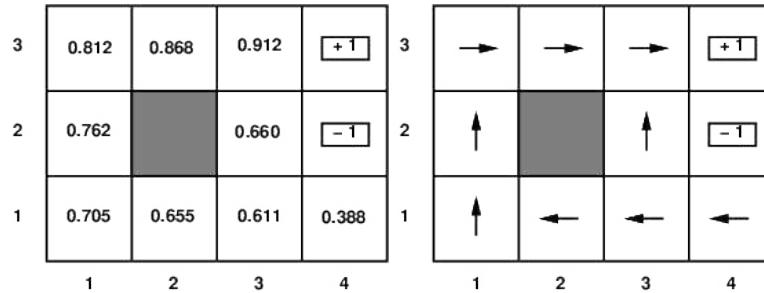
Utility of States (continued)

▷ **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

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \left(\sum_{s'} P(s' | s, a), U(s') \right)$$



▷ **Questionnaire:** Why do we go left in (3, 1) and not up? (compute the utility)



24.3 Value/Policy Iteration

Dynamic programming: the Bellman equation

▷ Definition of utility of states leads to a simple relationship among utilities of neighboring states:

expected sum of rewards = current reward + $\gamma \cdot \text{exp. reward sum after best action}$

▷ **Theorem 24.3.1 (Bellman equation (1957))**

$$U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \left(\sum_{s'} U(s') \cdot P(s' | s, a) \right)$$

▷ **Example 24.3.2** $U(1,1) = -0.04$

$$\begin{aligned} &+ \gamma \max\{0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \\ &\quad 0.9U(1,1) + 0.1U(1,2) \quad \text{up} \\ &\quad 0.9U(1,1) + 0.1U(2,1) \quad \text{left} \\ &\quad 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\} \quad \text{down} \\ &\quad \qquad \qquad \qquad \text{right} \end{aligned}$$

▷ **Problem:** One equation per state $\leadsto n$ nonlinear equations in n unknowns (**max** not linear) \leadsto cannot use linear algebra techniques for solving them.



Value Iteration Algorithm

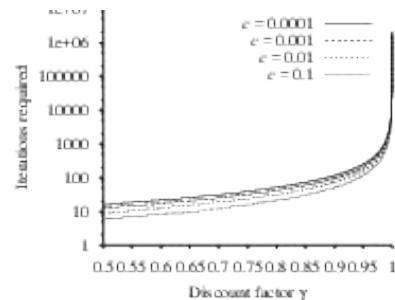
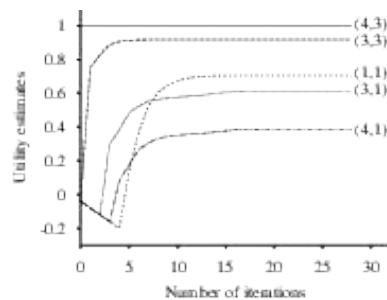
▷ **Idea:** we use a simple iteration scheme to find a fixpoint:

1. start with arbitrary utility values,
2. update to make them **locally consistent** with Bellman equation,
3. everywhere locally consistent \leadsto global optimality.

▷ **Definition 24.3.3** The **value iteration** algorithm for utility functions is given by

```
function VALUE-ITERATION (mdp,  $\epsilon$ ) returns a utility fn.
  inputs: mdp, an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ ,
           rewards  $R(s)$ , and discount  $\gamma$ 
   $\epsilon$ , the maximum error allowed in the utility of any state
  local variables:  $U$ ,  $U'$ , vectors of utilities for states in  $S$ , initially zero
   $\delta$ , the maximum change in the utility of any state in an iteration
repeat
   $U := U'$ ;  $\delta := 0$ 
  for each state  $s$  in  $S$  do
     $U'[s] := R(s) + \gamma \cdot \max_a \sum_{s'} U[s'] \cdot P(s' | s, a)$ 
    if  $|U'[s] - U[s]| > \delta$  then  $\delta := |U'[s] - U[s]|$ 
  until  $\delta < \epsilon(1 - \gamma)/\gamma$ 
return  $U$ 
```

▷ **Example 24.3.4 (Iteration on 4x3)**





Convergence

- ▷ Define the **max-norm** $\|U\| = \max_s |U(s)|$, so $\|U - V\| = \text{maximum difference between } U \text{ and } V$
- ▷ Let U^t and U^{t+1} be successive approximations to the true utility U
- ▷ **Theorem 24.3.5** *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 24.3.6** *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



Policy Iteration

- ▷ 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 24.3.7** *Upon termination U_i is a fixed point of Bellman update \rightsquigarrow Solution to Bellman equation $\rightsquigarrow \pi_i$ is an optimal policy.*
- ▷ **Observation 24.3.8** *Policy improvement improves policy and policy space is finite \rightsquigarrow termination.*



Policy Iteration Algorithm

- ▷ **Definition 24.3.9** 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
 π 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 π

```



## Policy Evaluation

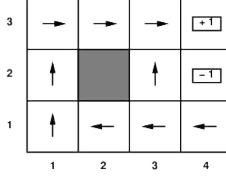
▷ **Problem:** How to implement the POLICY-EVALUATION algorithm?

▷ **Solution:** To compute utilities given a fixed  $\pi$ : For all  $s$  we have

$$U(s) = R(s) + \gamma \sum_{s'} U(s') \cdot P(s' | s, \pi(s))$$

▷ **Example 24.3.10 (Simplified Bellman Equations for  $\pi$ )**

$$\begin{aligned} U_i(1,1) &= -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1) \\ U_i(1,2) &= -0.04 + 0.8U_i(1,3) + 0.1U_i(1,2) \\ &\vdots \end{aligned}$$



▷ **Observation 24.3.11**  $n$  simultaneous linear equations in  $n$  unknowns, solve in  $\mathcal{O}(n^3)$  with standard linear algebra methods



## Modified policy iteration

▷ Policy iteration often converges in few iterations, but each is expensive

▷ **Idea:** Use a few steps of value iteration (but with  $\pi$  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



## 24.4 Partially Observable MDPs

We will now lift the last restriction we made in the decision problems for our agents: in the definition of [Markov decision procedures](#) we assumed that the environment was fully observable. As we have seen<sup>9</sup> this entails that the optimal policy only depends on the current state.

10

EdN:9  
EdN:10

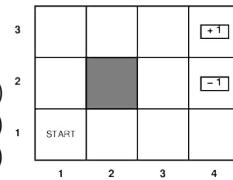
### Partial Observability

- ▷ **Definition 24.4.1** A [partially observable MDP](#) (a **POMDP** for short) is a MDP together with an [observation model](#)  $O$  that is stationary and has the sensor Markov property:  $O(s, e) = P(e | s)$ .

#### Example 24.4.2 (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  
 If sensor reports 1, we are in  $(3, ?)$  (noise)  
(probably)



- ▷ **Problem:** Agent does not know which state it is in  $\leadsto$  makes no sense to talk about policy  $\pi(s)$ !

- ▷ **Theorem 24.4.3 (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 POMDP, 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

$$b'(s') = \alpha P(e | s') \sum_s P(s' | s, a) \cdot b(s)$$

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!)

<sup>9</sup>EDNOTE: did we?<sup>10</sup>EDNOTE: MK: the observation model should be defined already

- ▷ **Consequence:** the optimal policy can be written as a function  $\pi^*(b)$  from belief states to actions.
- ▷ **POMDP decision cycle:** 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.



©: Michael Kohlhase

739



## Partial Observability contd.

- ▷ **Recap:** POMDP decision cycle is search in belief state space.
- ▷ **Observation 24.4.4** *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 24.4.5** The belief state of the 4x3 world is a 11-dimensional continuous space  
(11 states)
- ▷ **Theorem 24.4.6** *Solving POMDPs is very hard!* (actually, **PSPACE-hard**)
- ▷ **In particular** none of the algorithms we have learned applied (discreteness assumption)
- ▷ The real world is a POMDP (with initially unknown transition model  $T$  and observation model  $O$ )



©: Michael Kohlhase

740



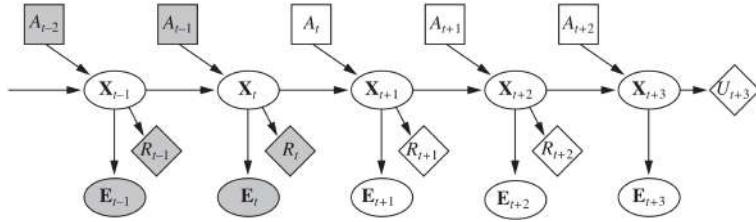
## 24.5 Online Agents with POMDPs

### Designing Online Agents for POMDPs

- ▷ **Definition 24.5.1 (Dynamic Decision Networks)**
  - ▷ transition and sensor models are represented as a DBN (a dynamic Bayesian network).

<sup>11</sup>EDNOTE: Commented out something about algorithms, make planning-based slides after AIMA3

- ▷ action nodes and utility nodes are added to create a **dynamic decision network** (DDN).
- ▷ a filtering algo 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  $t$

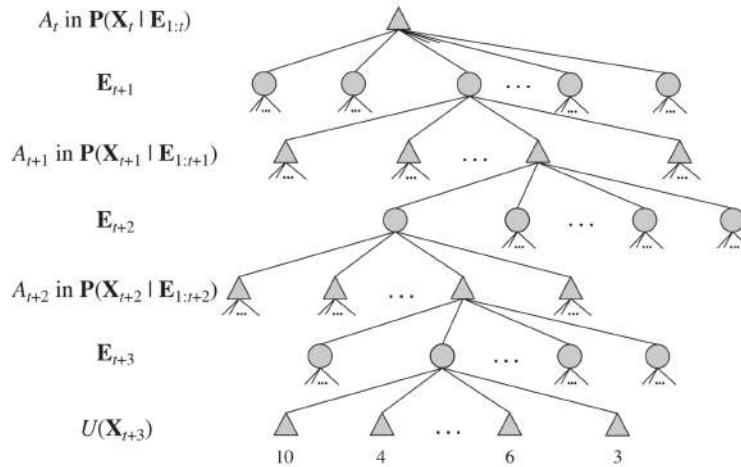


Variables with known values are gray, agent must choose a value for  $A_t$ .  
 Rewards for  $t = 0, \dots, t + 2$ , but utility for  $t + 3$  ( $\cong$  discounted sum of rest)



## Designing Online Agents for POMDPs (continued)

- ▷ Part of the lookahead solution of the DDN above      (search over action tree)

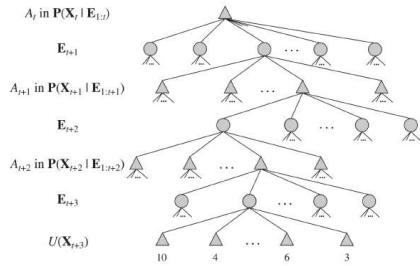


circle  $\cong$  chance nodes  
 triangle  $\cong$  belief state

(the environment decides)  
 (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  $E_{t+1:t+i}$  (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  $\mathcal{O}(|A|^d \cdot |\mathbf{E}|^d)$  ( $|A| \triangleq$  number of actions,  $|\mathbf{E}| \triangleq$  number of percepts)



## Summary

- ▷ Decision theoretic agents for sequential environments
- ▷ Building on temporal, probabilistic models/inference (dynamic Bayesian networks)
- ▷ MDPs for fully observable case
- ▷ Value/Policy Iteration for MDPs ↵ optimal policies
- ▷ POMDPs for partially observable case
- ▷ ≡ MDP on belief state space.
- ▷ The world is a POMDP with (initially) unknown transition and sensor models.



# Part VI

# Machine Learning



This part introduces the foundations of machine learning methods in AI. We discuss the problem learning from observations in general, study inference-based techniques, and then go into elementary statistical methods for learning.



# Chapter 25

## Learning from Observations

### Outline

- ▷ Learning agents
- ▷ Inductive learning
- ▷ Decision tree learning
- ▷ Measuring learning performance
- ▷ Computational Learning Theory
- ▷ Linear Regression and Classification
- ▷ Neural Networks
- ▷ Support Vector Machines



©: Michael Kohlhase

745



### 25.1 Forms of Learning

#### Learning (why is this a good idea)

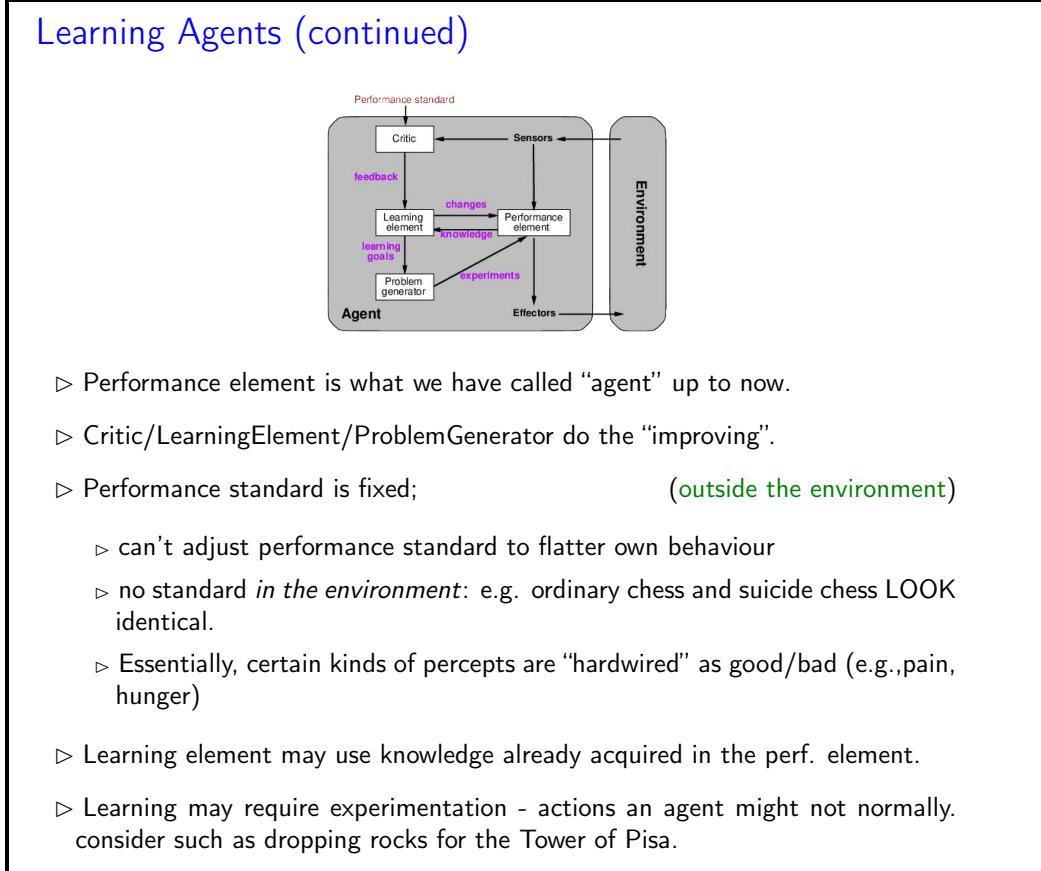
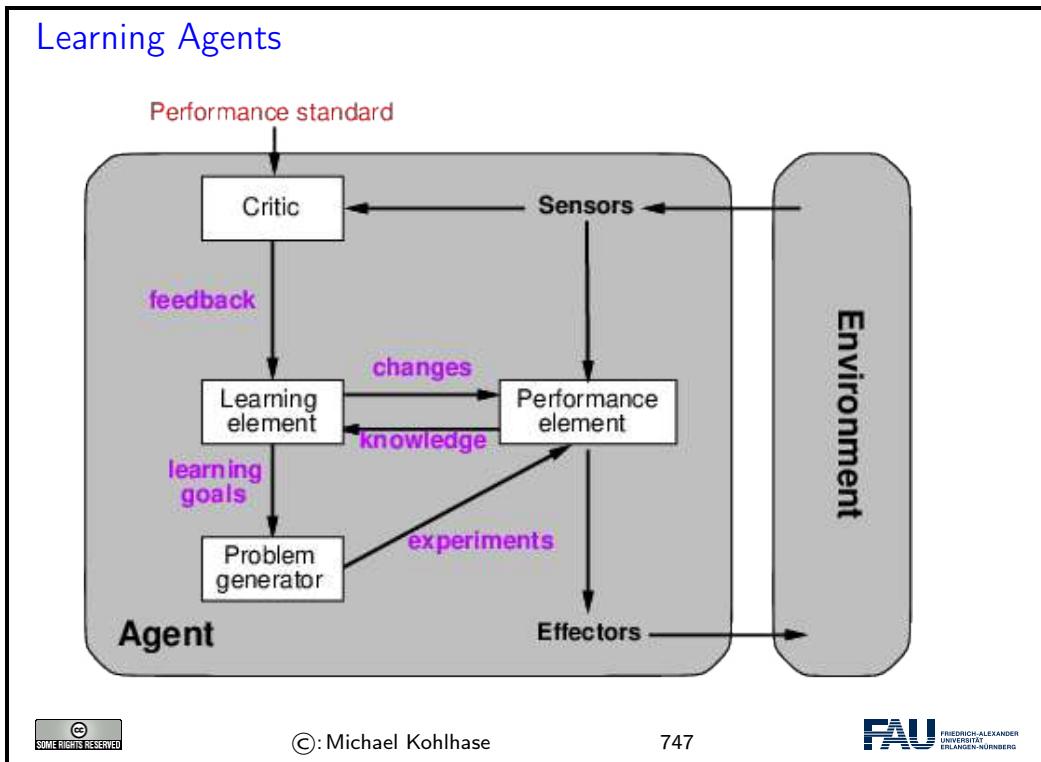
- ▷ Learning is essential for unknown environments, (i.e., when designer lacks omniscience)
  - ▷ The world is a POMDP with (initially) unknown transition and sensor models.
- ▷ Learning is useful as a system construction method, (i.e., expose the agent to reality rather than trying to write it down)
- ▷ Learning modifies the agent's decision mechanisms to improve performance



©: Michael Kohlhase

746







## Learning Element

- ▷ Design of learning element is dictated by
  - ▷ what type of performance element is used
  - ▷ which functional component is to be learned
  - ▷ how that functional component is represented
  - ▷ what kind of feedback is available

- ▷ **Example 25.1.1 (Scenarios)**

| Performance Elt.    | Component          | Representation      | Feedback     |
|---------------------|--------------------|---------------------|--------------|
| Alpha-beta search   | Evaluation fn.     | Weighted linear fn. | Win/loss     |
| Logical agent       | Transition model   | Successor-state ax. | Outcome      |
| Utility-based agent | Transition model   | Dynamic Bayes net   | Outcome      |
| Simple Reflex agent | Percept-action fn. | Neural net          | Corr. Action |

- ▷ **Supervised learning:** correct answers for each instance
- ▷ **Reinforcement learning:** occasional rewards



## 25.2 Inductive Learning

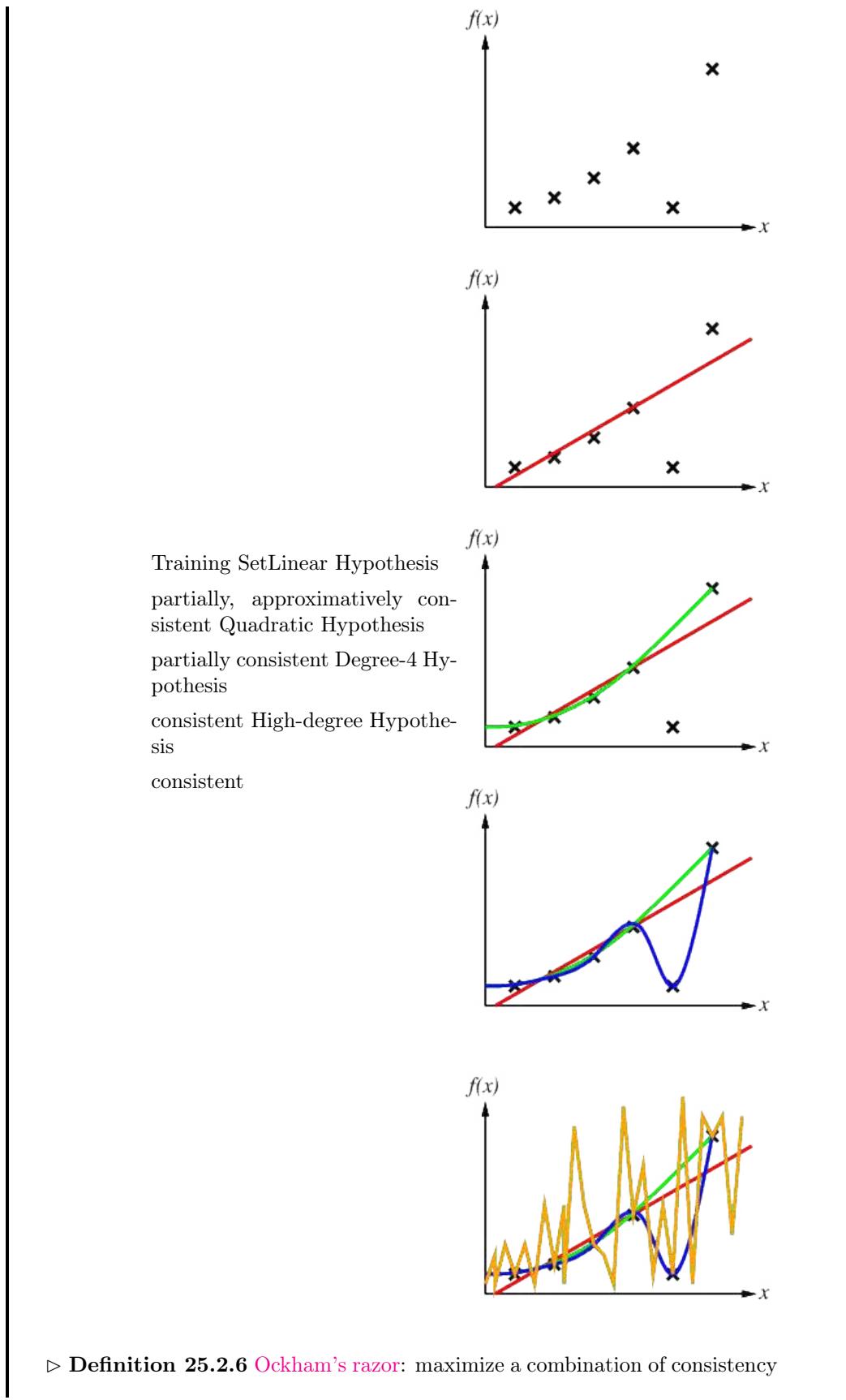
### Inductive learning (a.k.a. Science)

- ▷ Simplest form: learn a function from examples (tabula rasa)
- ▷ **Definition 25.2.1** An **example** is a pair  $(x, y)$  of an **input sample**  $x$  and an **outcome**  $y$ . We call a set  $S$  of examples **consistent**, if  $S$  is a **function**.
- ▷ **Example 25.2.2 (Examples in Tic-Tac-Toe)**  $\left(\begin{array}{|c|c|c|} \hline O & O & x \\ \hline X & X & \\ \hline X & & \\ \hline \end{array}, +1\right)$
- ▷ **Definition 25.2.3** The **inductive learning problem**  $\mathcal{P} := \langle \mathcal{H}, f \rangle$  consists in finding a **hypothesis**  $h \in \mathcal{H}$  such that  $f \simeq h|_{\text{dom}(f)}$  for a consistent **training set**  $f$  of **examples** and a **hypothesis space**  $\mathcal{H}$ . We also call  $f$  the **target function**.
- ▷ This is a highly simplified model of real learning:
  - ▷ Ignores prior knowledge.
  - ▷ Assumes a deterministic, observable “environment”.
  - ▷ Assumes examples are **given**.
  - ▷ Assumes that the agent **wants** to learn  $f$ . (why?)



## Inductive Learning Method

- ▷ **Idea:** Construct/adjust hypothesis  $h \in \mathcal{H}$  to agree with a training set  $f$ .
- ▷ **Definition 25.2.4** We call  $h$  **consistent with**  $f$ , if it agrees with  $f$  on all examples in  $T$ .
- ▷ **Example 25.2.5 (Curve Fitting)**



and simplicity



## Choosing the Hypothesis Space

- ▷ **Observation:** Whether we can find a consistent hypothesis for a given **training set** depends on the chosen hypothesis space.
- ▷ **Definition 25.2.7** We say that an inductive learning problem  $\langle \mathcal{H}, f \rangle$  is **realizable**, iff there is a  $h \in \mathcal{H}$  consistent with  $f$ .
- ▷ **Problem:** We do not know whether a given learning problem is realizable, unless we have prior knowledge.
- ▷ **Solution?**: Make  $\mathcal{H}$  large, e.g. the class of all Turing machines
- ▷ **Tradeoff:** The computational complexity of the inductive learning problem is tied to the size of the hypothesis space. E.g. consistency is not even decidable for general Turing machines.
- ▷ Much of the research in machine learning has concentrated on simple hypothesis spaces.
- ▷ We will concentrate on propositional logic and related languages.



## 25.3 Learning Decision Trees

### Attribute-based Representations

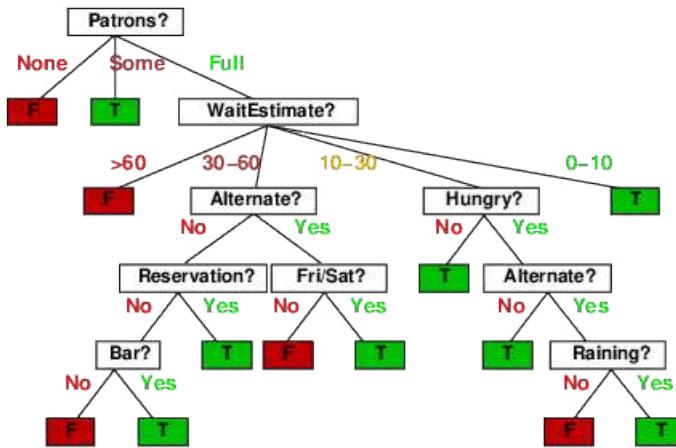
- ▷ **Definition 25.3.1** In **attribute-based representations**, examples are described by **attributes** and their **values** (Boolean, discrete, continuous, etc.)
- ▷ **Example 25.3.2 (In a Restaurant)** Situations where I will/won't wait for a table:

| Example  | Attributes |     |     |     |      |        |      |     |         |       | Target<br>WillWait |
|----------|------------|-----|-----|-----|------|--------|------|-----|---------|-------|--------------------|
|          | Alt        | Bar | Fri | Hun | Pat  | Price  | Rain | Res | Type    | Est   |                    |
| $X_1$    | T          | F   | F   | T   | Some | \$\$\$ | F    | T   | French  | 0–10  | T                  |
| $X_2$    | T          | F   | F   | T   | Full | \$     | F    | F   | Thai    | 30–60 | F                  |
| $X_3$    | F          | T   | F   | F   | Some | \$     | F    | F   | Burger  | 0–10  | T                  |
| $X_4$    | T          | F   | T   | T   | Full | \$     | F    | F   | Thai    | 10–30 | T                  |
| $X_5$    | T          | F   | T   | F   | Full | \$\$\$ | F    | T   | French  | >60   | F                  |
| $X_6$    | F          | T   | F   | T   | Some | \$\$   | T    | T   | Italian | 0–10  | T                  |
| $X_7$    | F          | T   | F   | F   | None | \$     | T    | F   | Burger  | 0–10  | F                  |
| $X_8$    | F          | F   | F   | T   | Some | \$\$   | T    | T   | Thai    | 0–10  | T                  |
| $X_9$    | F          | T   | T   | F   | Full | \$     | T    | F   | Burger  | >60   | F                  |
| $X_{10}$ | T          | T   | T   | T   | Full | \$\$\$ | F    | T   | Italian | 10–30 | F                  |
| $X_{11}$ | F          | F   | F   | F   | None | \$     | F    | F   | Thai    | 0–10  | F                  |
| $X_{12}$ | T          | T   | T   | T   | Full | \$     | F    | F   | Burger  | 30–60 | T                  |

- ▷ **Definition 25.3.3** Classification of examples is positive (T) or negative (F)

## Decision Trees

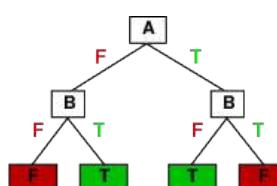
- ▷ One possible representation for hypotheses
- ▷ **Example 25.3.4** here is the “true” tree for deciding whether to wait:



## Expressiveness

- ▷ Decision trees can express any function of the input attributes.
- ▷ **Example 25.3.5** for Boolean functions, truth table row  $\rightsquigarrow$  path to leaf

| A | B | $A \oplus B$ |
|---|---|--------------|
| F | F | F            |
| F | T | T            |
| T | F | T            |
| T | T | F            |



- ▷ Trivially, there is a consistent decision tree for any training set w/ one path to leaf for each example (unless  $f$  nondeterministic in  $x$ ) but it probably won't generalize to new examples
- ▷ Prefer to find more *compact* decision trees

## Questionnaire: Hypothesis spaces

- ▷ **Question:** How many distinct decision trees with  $n$  Boolean attributes
- ▷ **Answer:** As many as there are Boolean functions = number of distinct truth tables with  $2^n$  rows =  $2^{2^n}$
- ▷ E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
- ▷ **Question:** How many purely conjunctive hypotheses (e.g., Hungry  $\wedge \neg$  Rain)
- ▷ Each attribute can be in (positive), in (negative), or out  $\rightsquigarrow 3^n$  distinct conjunctive hypotheses
- ▷ More expressive hypothesis space:
  - ▷ increases chance that target function can be expressed ☺
  - ▷ increases number of hypotheses consistent w/ training set  $\rightsquigarrow$  may get worse predictions ☹



©: Michael Kohlhase

756



## Decision Tree learning

- ▷ **Aim:** find a small tree consistent with the training examples
- ▷ **Idea:** (recursively) choose “most significant” attribute as root of (sub)tree
- ▷ **Definition 25.3.6** The following algorithm performs **decision tree learning**

```

function DTL(examples, attributes, default) returns a decision tree
 if examples is empty then return default
 else if all examples have the same classification then return the classification
 else if attributes is empty then return MODE(examples)
 else
 best := Choose-Attribute(attributes, examples)
 tree := a new decision tree with root test best
 m := MODE(examples)
 for each value vi of best do
 examplesi := {elements of examples with best = vi}
 subtree := DTL(examplesi, attributes \ best, m)
 add a branch to tree with label vi and subtree subtree
 return tree

```

MODE(*examples*)= most frequent value in *example*.



©: Michael Kohlhase

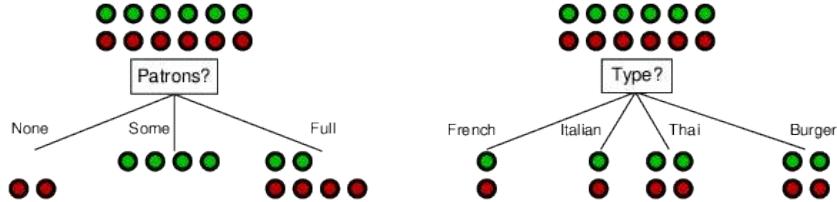
757



## Choosing an attribute

- ▷ **Idea:** a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”.

## ▷ Example 25.3.7



*Patrons?* is a better choice—gives *information* about the classification

▷ Can we make this more formal? ↗ use information theory



## 25.4 Using Information Theory

### Information Entropy

- ▷ **Intuition:** Information answers questions
- ▷ The more clueless I am about the answer initially, the more information is contained in the answer
- ▷ **Scale:**  $1 \text{ b} \hat{=} 1 \text{ bit} \hat{=} \text{answer to Boolean question with prior } (0.5, 0.5)$
- ▷ **Definition 25.4.1** If the prior is  $\langle P_1, \dots, P_n \rangle$ , then the **information** in an answer (also called **entropy** of the prior) is

$$I(\langle P_1, \dots, P_n \rangle) := \sum_{i=1}^n -P_i \log_2(P_i)$$

#### ▷ Example 25.4.2 (Information of a Coin Toss)

- ▷ For a fair coin toss we have  $I(\langle \frac{1}{2}, \frac{1}{2} \rangle) = -\frac{1}{2} \log_2(\frac{1}{2}) - \frac{1}{2} \log_2(\frac{1}{2}) = 1 \text{ b.}$
- ▷ With a loaded coin (99% heads) we have  $I(\langle \frac{1}{100}, \frac{99}{100} \rangle) = 0.08 \text{ b.}$
- ▷ **Intuition:** Information goes to 0 as head probably goes to 1.



### Information Gain

- ▷ Suppose we have  $p$  positive and  $n$  negative examples at the root
- ▷  $\sim I(\langle p/(p+n), n/(p+n) \rangle)$  bits needed to classify a new example
- ▷ **Example 25.4.3** For 12 restaurant examples,  $p = n = 6$  so we need 1 b of information

- ▷ An attribute splits the examples  $E$  into subsets  $E_i$ , each of which (we hope) needs less information to complete the classification
- ▷ Let  $E_i$  have  $p_i$  positive and  $n_i$  negative examples
- ▷  $\sim I(\langle p_i/(p_i + n_i), n_i/(p_i + n_i) \rangle)$  bits needed to classify a new example
- ▷  $\sim$  expected number of bits per example over all branches is

$$\sum_i \frac{p_i + n_i}{p + n} I(\langle p_i/(p_i + n_i), n_i/(p_i + n_i) \rangle)$$

- ▷ **Definition 25.4.4** The information gain from an attribute test  $A$  is

$$\text{Gain}(A) := I\left(\left\langle \frac{p}{p+n}, \frac{n}{p+n} \right\rangle\right) - \sum_i \frac{p_i + n_i}{p + n} I(\langle p_i/(p_i + n_i), n_i/(p_i + n_i) \rangle)$$



©: Michael Kohlhase

760



## Information Gain (continued)

- ▷ **Example 25.4.5**

$$\begin{aligned} \text{Gain(Patrons?)} &= 1 - \frac{2}{12} I(\langle 0, 1 \rangle) + \frac{4}{12} I(\langle 1, 0 \rangle) + \frac{6}{12} I(\langle \frac{2}{6}, \frac{4}{6} \rangle) \\ &\approx 0.541 \text{ b} \\ \text{Gain(Type)} &= 1 - \frac{2}{12} I(\langle \frac{1}{2}, \frac{1}{2} \rangle) + \frac{2}{12} I(\langle \frac{1}{2}, \frac{1}{2} \rangle) + \frac{4}{12} I(\langle \frac{2}{4}, \frac{2}{4} \rangle) + \frac{4}{12} I(\langle \frac{2}{4}, \frac{2}{4} \rangle) \\ &\approx 0 \text{ b} \end{aligned}$$

- ▷ choose the attribute that maximises information gain



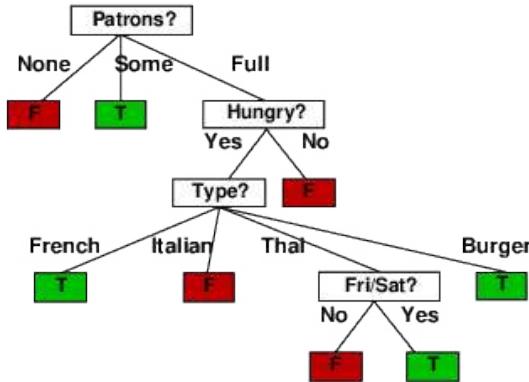
©: Michael Kohlhase

761



## Restaurant Example contd.

- ▷ Decision tree learned by DTL from the 12 examples using information gain maximization for Choose-Attribute:



- ▷ Substantially simpler than “true” tree—a more complex hypothesis isn’t justified by small amount of data



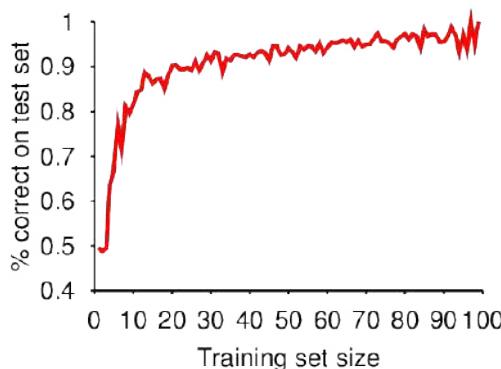
©: Michael Kohlhase

762



## Performance measurement

- ▷ How do we know that  $h \approx f$ ? (Hume’s *Problem of Induction*)
- 1. Use theorems of computational/statistical learning theory
- 2. Try  $h$  on a new **test set** of examples (use same distribution over example space as training set)
- 3. **Definition 25.4.6** The **learning curve** = % correct on test set as a function of training set size



4.



©: Michael Kohlhase

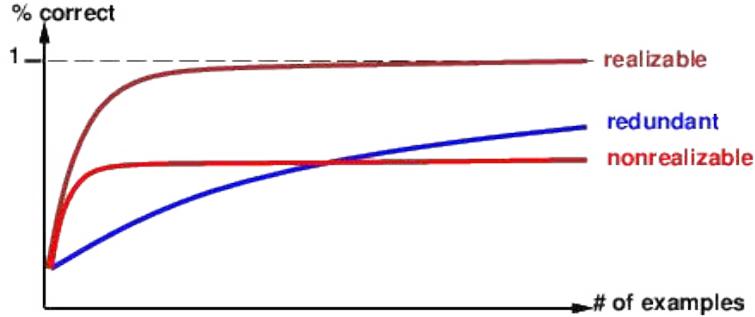
763



## Performance measurement contd.

- ▷ **Observation 25.4.7** The learning curve depends on
  - ▷ realizable (can express target function) vs. non-realizable
  - non-realizability can be due to missing attributes or restricted hypothesis

- class (e.g., thresholded linear function)*
- ▷ *redundant expressiveness (e.g., loads of irrelevant attributes)*



## Generalization and Overfitting

- ▷ Sometimes a learned hypothesis is more specific than the experiments warrant
- ▷ **Definition 25.4.8** We speak of *overfitting*, if a hypothesis  $h$  describes random error rather than the underlying relationship. *Underfitting* occurs when  $h$  cannot capture the underlying trend of the data.
- ▷ *Qualitatively:* Overfitting increases with the size of hypothesis space and the number of attributes, but decreases with number of examples.
- ▷ *Idea:* Combat overfitting “generalizing” decision trees computed by DTL.



## Decision Tree Pruning

- ▷ *Idea:* Combat overfitting “generalizing” decision trees  $\rightsquigarrow$  prune “irrelevant” nodes.
- ▷ **Definition 25.4.9** For *decision tree pruning* repeat the following on a learned decision tree:
  - ▷ Find a *terminal* test node  $n$  (only result leaves as descendent)
  - ▷ if test is *irrelevant*, i.e. has low information gain, *prune* it by replacing  $n$  by with a leaf node.
- ▷ **Question:** How big should the information gain be to split ( $\rightsquigarrow$  keep) a node?
- ▷ *Idea:* Use a statistical significance test.
- ▷ **Definition 25.4.10** A result has *statistical significance*, if the probability they could arise from the *null hypothesis* (i.e. the assumption that there is no underlying pattern) is very low (usually 5%).



## Determining Attribute Irrelevance

- ▷ For **decision tree pruning**, the **null hypothesis** is that the attribute is **irrelevant**.
  - ▷ Compute the probability that the example distribution ( $p$  positive,  $n$  negative) for a terminal node deviates from the expected distribution under the null hypothesis.
  - ▷ For an attribute  $A$  with  $d$  values, compare the actual numbers  $p_k$  and  $n_k$  in each subset  $s_k$  with the expected numbers **(expected if  $A$  is irrelevant)**
  - $\hat{p}_k = p \frac{p_k+n_k}{p+n}$  and  $\hat{n}_k = n \frac{p_k+n_k}{p+n}$ .
  - ▷ A convenient measure of the total deviation is **(sum of squared errors)**
- $$\Delta = \sum_{k=1}^d \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$
- ▷ **Lemma 25.4.11 (Neyman-Pearson)** *Under the null hypothesis, the value of  $\Delta$  is distributed according to the  $\chi^2$  distribution with  $d - 1$  degrees of freedom.* [JN33]
  - ▷ **Definition 25.4.12** **Decision tree pruning** with Pearson's  $\chi^2$  with  $d - 1$  degrees of freedom for  $\Delta$  is called  **$\chi^2$  pruning**. ( $\chi^2$  values from stats library.)
  - ▷ **Example 25.4.13** The **type** attribute has four values, so three degrees of freedom, so  $\Delta = 7.82$  would reject the null hypothesis at the 5% level.



## 25.5 Evaluating and Choosing the Best Hypothesis

### Independent and Identically Distributed

- ▷ We want to learn a hypothesis that fits the future data best.
- ▷ To make predictions about the future, we need a **stationarity assumption** for the probability distribution.
- ▷ **Idea:** Each example – before we see it – is a random variable  $E_j$ , the observed value  $e_j = (x_j, y_j)$  samples its distribution.
- ▷ **Definition 25.5.1** A sequence of  $E_j$  of random variables is **independent and identically distributed** (short **IID**), iff they are
  - ▷ **independent**, i.e.  $\mathbf{P}(E_j | E_{j-1}, E_{j-2}, \dots) = \mathbf{P}(E_j)$  and
  - ▷ **identically distributed**, i.e.  $\mathbf{P}(E_i) = \mathbf{P}(E_j)$  for all  $i$  and  $j$ .
- ▷ **Example 25.5.2** A sequence of (fair or loaded) dice tosses is IID.

**Stationarity Assumption:** We assume that the set  $\mathcal{E}$  of possible examples is IID in the future.



## ▷ Error Rates and Cross-Validation

- ▷ We want to learn a hypothesis that fits the future data best.
- ▷ **Definition 25.5.3** Given an inductive learning problem  $\langle \mathcal{H}, f \rangle$ , we define the **error rate** of a hypothesis  $h \in \mathcal{H}$  as the fraction of errors:

$$\frac{\#(\{x \in \text{dom}(f) \mid h(x) \neq f(x)\})}{\#(\text{dom}(f))}$$

- ▷ **Caveat:** A low error rate on the **training set** does not mean that a hypothesis generalizes well.
- ▷ **Idea:** Do not use homework questions in the exam.
- ▷ **Definition 25.5.4** Splitting the data available for learning into a
  - ▷ **training set** from which the learning algorithm produces a hypothesis  $h$
  - ▷ and a **test set**, which is used for evaluating  $h$
 is called **holdout cross-validation**. (no peeking at test set allowed)



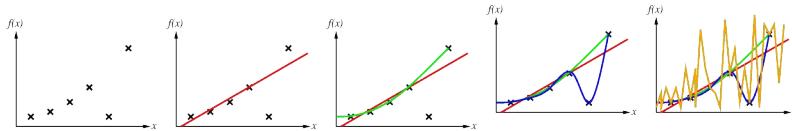
## Error Rates and Cross-Validation

- ▷ **Question:** What is a good ration between **training set** and test set size?
  - ▷ small **training set**  $\rightsquigarrow$  poor hypothesis
  - ▷ small test set  $\rightsquigarrow$  poor estimate of the accuracy
- ▷ **Definition 25.5.5** In  **$k$ -fold cross-validation**, we perform  $k$  rounds of learning, each with  $1/k$  of the data as test set and average over the  $k$  test scores.
- ▷ each example does double duty: for training and testing.
- ▷  $k = 5$  and  $k = 10$  are popular  $\rightsquigarrow$  good accuracy at  $k$  times computation time.
- ▷ **Definition 25.5.6** If  $k = \#(\text{dom}(f))$ , then  $k$ -fold cross-validation is called **leave-one-out cross-validation (LOOCV)**.



## Model Selection

- ▷ **Definition 25.5.7** The **model selection** problem is to determine – given data – a good hypothesis space.
- ▷ **Example 25.5.8** What is the best polynomial degree to fit the data



- ▷ **Observation 25.5.9** We can solve the problem of “learning from observations  $f$ ” in a two-part process

- ▷ **model selection** determines a hypotheses space  $\mathcal{H}$ ,
- ▷ **optimization** solves the induced inductive learning problem  $\langle \mathcal{H}, f \rangle$ .

**Idea:** Solve the two parts together by iteration over “size” (they inform each other)

▷ Need a notion of “size” ↵ e.g. number of nodes in a decision tree

▷ **Concrete Problem:** find the “size” that best balances overfitting and underfitting to optimize test set accuracy.



## Model Selection Algorithm (Wrapper)

```

function CROSS-VALIDATION-WRAPPER(Learner,k,examples) returns a hypothesis
 local variables: errT, an array, indexed by size, storing training-set error rates
 errV, an array, indexed by size, storing validation-set error rates
 for size = 1 to ∞ do
 errT[size], errV[size] := CROSS-VALIDATION(Learner,size,k,examples)
 if errT has converged then do
 best_size := the value of size with minimum errV[size]
 return Learner(best_size,examples)

function CROSS-VALIDATION(Learner,size,k,examples) returns two values:
 average training set error rate, average validation set error rate
 fold_errT := 0; fold_errV := 0
 for fold = 1 to k do
 training_set, validation_set := PARTITION(examples,fold,k)
 h := Learner(size,training_set)
 fold_errT := fold_errT + ERROR-RATE(h,training_set)
 fold_errV := fold_errV + ERROR-RATE(h,validation_set)
 return fold_errT/k, fold_errV/k

function PARTITION(examples,fold,k) returns two sets:
 a validation set of size $|examples|/k$ and the rest; the split is different for each fold value

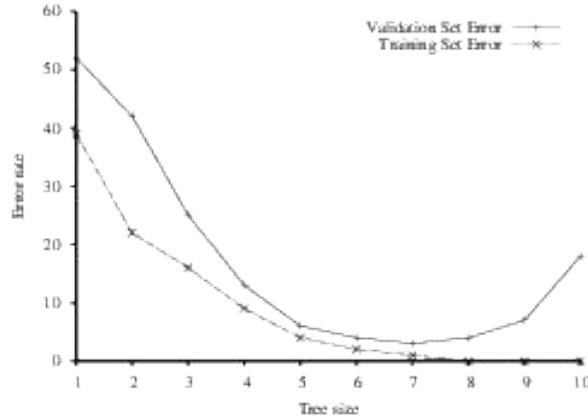
```



## Error Rates on Training/Validation Data

- ▷ **Example 25.5.10 (An Error Curve for Restaurant Decision Trees)**

Modify DTL to be breadth-first, information gain-sorted, stop after  $k$  nodes.



Stops when training set error rate converges, choose optimal tree for validation curve  
(here a tree with 7 nodes)



## From Error Rates to Loss Functions

- ▷ So far we have been minimizing error rates. (better than maximizing ☺)
- ▷ Example 25.5.11 (Classifying Spam) It is much worse to classify ham as spam than vice versa (message loss)
- ▷ Recall Rationality: Decision-makers should maximize expected utility (MEU)
- ▷ So: machine learning should maximize utility (not only minimize error rates)
- ▷ Machine learning traditionally deals with utilities in form of “loss functions”
- ▷ Definition 25.5.12 The loss function  $L$  is defined by setting  $L(x, y, \hat{y})$  to be the amount of utility lost by prediction  $h(x) = \hat{y}$  instead of  $f(x) = y$ . If  $L$  is independent of  $x$ , we often use  $L(y, \hat{y})$ .
- ▷ Example 25.5.13  $L(\text{spam}, \text{ham}) = 1$ , while  $L(\text{ham}, \text{spam}) = 10$ .



## Generalization Loss

- ▷ Note:  $L(y, y) = 0$  (no loss if you are exactly correct)
- ▷ Definition 25.5.14 (Popular general loss functions)

|                     |                                                                      |                       |
|---------------------|----------------------------------------------------------------------|-----------------------|
| absolute value loss | $L_1(y, \hat{y}) :=  y - \hat{y} $                                   | small errors are good |
| squared error loss  | $L_2(y, \hat{y}) := (y - \hat{y})^2$                                 | dito                  |
| 0/1 loss            | $L_{0/1}(y, \hat{y}) := 0, \text{ if } y = \hat{y}, \text{ else } 1$ | error rate            |

▷ **Idea:** Maximize expected utility by choosing hypothesis  $h$  that minimizes expected loss over all  $(x, y) \in f$ .

▷ **Definition 25.5.15** Let  $\mathcal{E}$  be the set of all possible examples and  $\mathbf{P}(X, Y)$  the prior probability distribution over its components, then the expected **generalization loss** for a hypothesis  $h$  with respect to a **loss function**  $L$  is

$$\text{GenLoss}_L(h) := \sum_{(x,y) \in \mathcal{E}} L(y, h(x)) \cdot P(x, y)$$

and the best hypothesis  $h^* := \operatorname{argmin}_{h \in \mathcal{H}} (\text{GenLoss}_L(h))$ .



## Empirical Loss

▷ **Problem:**  $\mathbf{P}(X, Y)$  is unknown so the learner can only estimate the generalization loss:

▷ **Definition 25.5.16** Let  $L$  be a loss function and  $E$  a set of examples with  $\#(E) = N$ , then we call

$$\text{EmpLoss}_{L,E}(h) := \frac{1}{N} \sum_{(x,y) \in E} L(y, h(x))$$

the **empirical loss** and  $\hat{h}^* := \operatorname{argmin}_{h \in \mathcal{H}} (\text{EmpLoss}_{L,E}(h))$  the **estimated best hypothesis**.

▷ There are four reasons why  $\hat{h}^*$  may differ from  $f$ .

- ▷ **realizability:** then we have to settle for an approximation  $\hat{h}^*$  of  $f$
- ▷ **variance:** different subsets of  $f$  give different  $\hat{h}^* \rightsquigarrow$  more examples
- ▷ **noise:** if  $f$  is non-deterministic, then we cannot expect perfect results
- ▷ **computational complexity:** if  $\mathcal{H}$  is too large to systematically explore, we make due with subset and get an approximation.



## Regularization

▷ **Idea:** Directly use empirical loss to solve model selection. (**finding a good  $\mathcal{H}$** )

Minimize the weighted sum of empirical loss and hypothesis complexity      (**to avoid overfitting**).

▷ **Definition 25.5.17** Let  $\lambda \in \mathbb{R}$ ,  $h \in \mathcal{H}$ , and  $E$  a set of examples, then we

call

$$\text{Cost}_{L,E}(h) := \text{EmpLoss}_{L,E}(h) + \lambda \text{ Complexity}(h)$$

the **total cost** of  $h$  on  $E$ .

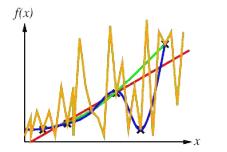
- ▷ **Definition 25.5.18** The process of finding a total cost minimizing hypothesis

$$\hat{h}^* := \operatorname{argmin}_{h \in \mathcal{H}} (\text{Cost}_{L,E}(h))$$

is called **regularization**; Complexity is called the **regularization function** or **hypothesis complexity**.

- ▷ **Example 25.5.19 (Regularization for Polynomials)**

A good regularization function for polynomials is the sum of squares of exponents.  $\sim$  keep away from wriggly curves!



## Minimal Description Length

- ▷ **Remark:** In regularization, empirical loss and hypothesis complexity are not measured in the same scale  $\sim \lambda$  mediates between scales.

- ▷ **Idea:** Measure both in the same scale  $\sim$  use information content, i.e. in **bits**.

- ▷ **Definition 25.5.20** Let  $h \in \mathcal{H}$  be a hypothesis and  $E$  a set of examples, then the **description length** of  $(h, E)$  is computed as follows:

1. encode the hypothesis as a **Turing machine program**, count **bits**.
2. count data **bits**:
  - ▷ correctly predicted example  $\sim 0$  b
  - ▷ incorrectly predicted example  $\sim$  according to size of error.

The **minimum description length** or **MDL hypothesis** minimizes the total number of **bits** required.

- ▷ This works well in the limit, but for smaller problems there is a difficulty in that the choice of encoding for the program affects the outcome.

- ▷ e.g., how best to encode a decision tree as a bit string



## The Scale of Machine Learning

- ▷ Traditional methods in statistics and early machine learning concentrated on **small-scale learning** (**50-5000 examples**)

- ▷ generalization error mostly comes from
  - ▷ approximation error of not having the true  $f$  in the hypothesis space
  - ▷ estimation error of too few training examples to limit variance.
  
- ▷ In recent years there has been more emphasis on large-scale learning (millions of examples)
  - ▷ generalization error is dominated by limits of computation
    - ▷ there is enough data and a rich enough model that we could find an  $h$  that is very close to the true  $f$ ,
    - ▷ but the computation to find it is too complex, so we settle for a sub-optimal approximation.
  - ▷ hardware advances (GPU farms, Amazon EC2, Google Data Centers, ...) help



## 25.6 Computational Learning Theory

### A (General) Theory of Learning?

- ▷ Main Question: How can we be sure that our learning algorithm has produced a hypothesis that will predict the correct value for previously unseen inputs?
  
- ▷ Formally: How do we know that the hypothesis  $h$  is close to the target function  $f$  if we don't know what  $f$  is?
  
- ▷ Other - more recent - Questions:
  - ▷ how many examples do we need to get a good  $h$ ?
  - ▷ What hypothesis space  $\mathcal{H}$  should we use?
  - ▷ If the  $\mathcal{H}$  is very complex, can we even find the best  $h$ , or do we have to settle for a local maximum in  $\mathcal{H}$ .
  - ▷ How complex should  $h$  be?
  - ▷ How do we avoid overfitting?
  
- ▷ “Computational Learning Theory” tries to answer these using concepts from AI, statistics, and theoretical CS.



### PAC Learning

- ▷ Basic idea of Computational Learning Theory:
  - ▷ any hypothesis  $h$  that is seriously wrong will almost certainly be “found out” with high probability after a small number of examples, because it will make an incorrect prediction.

- ▷ Thus, if  $h$  is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong.
- ▷  $\rightsquigarrow h$  is probably approximately correct.
- ▷ **Definition 25.6.1** Any learning algorithm that returns hypotheses that are probably approximately correct is called a PAC learning algorithm.
- ▷ Derive performance bounds for PAC learning algorithms in general, using the
- ▷ **Stationarity Assumption (again):** We assume that the set  $\mathcal{E}$  of possible examples is IID  $\rightsquigarrow$ , we have a fixed distribution  $\mathbf{P}(E) = \mathbf{P}(X, Y)$  on examples.
- ▷ **Simplifying Assumptions:**  $f$  is a function (deterministic) and  $f \in \mathcal{H}$ .



## PAC Learning

- ▷ Start with PAC theorems for Boolean functions, for which  $L_{0/1}$  is appropriate.
- ▷ **Definition 25.6.2** The error rate  $\text{error}(h)$  of a hypothesis  $h$  is the probability that  $h$  misclassifies a new example.

$$\text{error}(h) := \text{GenLoss}_{L_{0/1}}(h) = \sum_{(x,y) \in \mathcal{E}} L_{0/1}(y, h(x)) \cdot P(x, y)$$

▷ **Intuition:**  $\text{error}(h)$  is the probability that  $h$  misclassifies a new example.

▷ This is the same quantity as measured in the learning curves above.

▷ **Definition 25.6.3** A hypothesis  $h$  is called approximatively correct, iff  $\text{error}(h) \leq \epsilon$  for some small  $\epsilon > 0$ .

We write  $\mathcal{H}_b := \{h \in \mathcal{H} \mid \text{error}(h) > \epsilon\}$  for the “seriously bad” hypotheses.



## Sample Complexity

- ▷ Let’s compute the probability that  $h_b \in \mathcal{H}_b$  is consistent with the first  $N$  examples.
- ▷ We know  $\text{error}(h_b) > \epsilon$ 
  - $\rightsquigarrow P(h_b \text{ agrees with } N \text{ examples}) \leq (1 - \epsilon)^N$ . (independence)
  - $\rightsquigarrow P(\mathcal{H}_b \text{ contains consistent hyp.}) \leq \#(\mathcal{H}_b) \cdot (1 - \epsilon)^N \leq \#(\mathcal{H}) \cdot (1 - \epsilon)^N$ . ( $\mathcal{H}_b \subseteq \mathcal{H}$ )
  - $\rightsquigarrow$  to bound this by a small  $\delta$ , show the algorithm  $N \geq \frac{1}{\epsilon} \cdot (\log_2(\frac{1}{\delta}) + \log_2(\#(\mathcal{H})))$  examples.
- ▷ **Definition 25.6.4** The number of required examples as a function of  $\epsilon$  and  $\delta$  is called the sample complexity of  $\mathcal{H}$ .

- ▷ **Example 25.6.5** If  $\mathcal{H}$  is the set of  $n$ -ary Boolean functions, then  $\#(\mathcal{H}) = 2^{2^n}$ .
    - ↪ sample complexity grows with  $\mathcal{O}(\log_2(2^{2^n})) = \mathcal{O}(2^n)$ .
    - There are  $2^n$  possible examples,
    - ↪ PAC learning for Boolean functions needs to see (nearly) all examples.



# Escaping Sample Complexity

- ▷ **Problem:** PAC learning for Boolean functions needs to see (nearly) all examples.
    - ▷  $\mathcal{H}$  contains enough hypotheses to classify any given set of examples in all possible ways.
    - ▷ In particular, for any set of  $N$  examples, the set of hypotheses consistent with those examples contains equal numbers of hypotheses that predict  $x_{N+1}$  to be positive and hypotheses that predict  $x_{N+1}$  to be negative.
  - ▷ **Idea/Problem:** restrict the  $\mathcal{H}$  in some way (but we may lose realizability)
  - ▷ **Three Ways out of this Dilemma:**
    1. bring prior knowledge into the problem (Section 27.4)
    2. prefer simple hypotheses (e.g. decision tree pruning)
    3. focus on “learnable subsets” of  $\mathcal{H}$  (next)

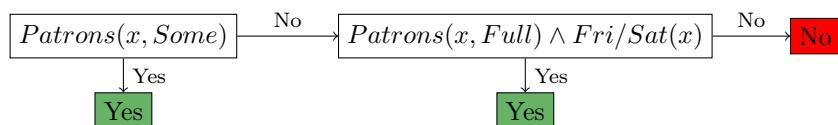


PAC Learning: Decision Lists

- ▷ **Idea:** Apply PAC learning to a “learnable hypothesis space”.
  - ▷ **Definition 25.6.6** A **decision list** consists of a sequence of tests, each of which is a conjunction of literals.
    - ▷ If a test succeeds when applied to an example description, the decision list specifies the value to be returned.
    - ▷ If the test fails, processing continues with the next test in the list.

**Remark:** Like [decision trees](#), but restricted branching, but more complex tests.

#### ► Example 25.6.7 (A decision list for the Restaurant Problem)



- ▷ **Lemma 25.6.8** Given arbitrary size conditions, decision lists can represent arbitrary Boolean functions. (equivalent to CNF)
- ▷ This directly defeats our purpose of finding a “learnable subset” of  $\mathcal{H}$ .



## Decision Lists: Learnable Subsets (Size-Restricted Cases)

- ▷ **Definition 25.6.9** The set of decision lists where tests are of conjunctions of at most  $k$  literals is called  **$k$ -DL**.
- ▷ **Example 25.6.10** The decision list from Example 25.6.7 is in 2-DL.
- ▷ **Observation 25.6.11**  $k$ -DL contains  $k$ -DT, the set of **decision trees** of depth at most  $k$ .
- ▷ **Definition 25.6.12** We denote the set of  $k$ -DT **decision trees** with at most  $n$  Boolean attributes with  $k$ -DT( $n$ ), and analogously  $k$ -DL( $n$ ). The language of conjunctions of at most  $k$  literals using  $n$  attributes is written as Conj( $k, n$ ).
- ▷ Decision lists are constructed of optional yes/no-tests, so there are at most  $3^{|Conj(k,n)|}$  distinct sets of component tests. Each of these sets of tests can be in any order, so  $|k\text{-DL}(n)| \leq 3^{|Conj(k,n)|} \cdot |Conj(k,n)|!$



## Decision Lists: Learnable Subsets (Sample Complexity)

- ▷ The number of conjunctions of  $k$  literals from  $n$  attributes is given by

$$|Conj(k, n)| = \sum_{i=1}^k \binom{2n}{i}$$

thus  $|Conj(k, n)| = \mathcal{O}(n^k)$ . Hence, we obtain (after some work)

$$|k\text{-DL}(n)| = 2^{\mathcal{O}(n^k \log_2(n^k))}$$

- ▷ plug this into the equation for the sample complexity:  $N \geq \frac{1}{\epsilon} \cdot (\log_2(\frac{1}{\delta}) + \log_2(|\mathcal{H}|))$  to obtain

$$N \geq \frac{1}{\epsilon} \cdot (\log_2(\frac{1}{\delta}) + \log_2(\mathcal{O}(n^k \log_2(n^k))))$$

- ▷ **Intuitively:** any algorithm that returns a consistent decision list will PAC-learn a  **$k$ -DL** function in a reasonable number of examples, for small  $k$ .



## Decision Lists Learning

- ▷ **Idea:** Use a greedy algorithm that repeats
    1. finds test that agrees exactly with some subset  $E$  of the training set
    2. adds it to the decision list under construction and removes  $E$
    3. construct the remainder of the DL using just the remaining examples.

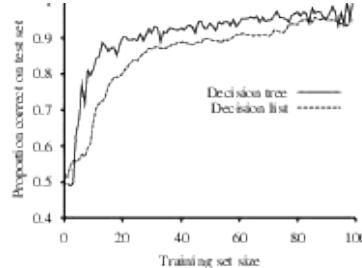
until there are no examples left
  - ▷ **Definition 25.6.13** The following algorithm performs **decision list learning**
- ```

function DLL( $E$ ) returns a decision list, or failure
  if  $E$  is empty then return (the trivial decision list) No
   $t :=$  a test that matches a nonempty subset  $E_t$  of  $E$ 
    such that the members of  $E_t$  are all positive or all negative
  if there is no such  $t$  then return failure
  if the examples in  $E_t$  are positive then  $o :=$  Yes else  $o :=$  No
  return a decision list with initial test  $t$  and outcome  $o$  and remaining tests given by
    DLL( $E \setminus E_t$ )
```



Decision Lists Learning in Comparison

- ▷ **Learning curves:** for DLL (and DTL for comparison)



i.e. The simpler DLL works quite well!



25.7 Regression and Classification with Linear Models

Linear Regression and Classification

- ▷ We pass on to another hypothesis space: linear functions over continuous-valued inputs.
- ▷ **Definition 25.7.1** We call an inductive learning problem $\langle \mathcal{H}, f \rangle$ a **classification problem**, iff $\text{codom}(f)$ is **discrete**, and a **regression problem** if $\text{codom}(f)$ is **continuous**, i.e. non-discrete – usually real-valued.

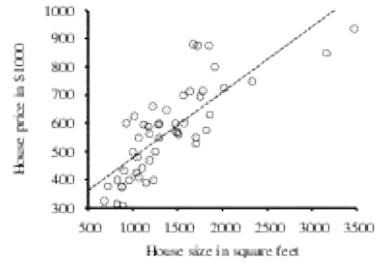


Univariate Linear Regression

- ▷ **Definition 25.7.2** A **univariate** or **unary** function is a function with one argument.
- ▷ **Recall:** A mapping between **vector spaces** is called **linear**, iff it preserves **vector addition** and **scalar multiplication**.
- ▷ **Observation 25.7.3** A **univariate, linear function** $f: \mathbb{R} \rightarrow \mathbb{R}$ is of the form $f(x) = w_1 x + w_0$ for some $w_i \in \mathbb{R}$.
- ▷ **Definition 25.7.4** Given a vector $\mathbf{w} := (w_0, w_1)$, we define $h_{\mathbf{w}}(x) := w_1 x + w_0$.
- ▷ **Definition 25.7.5** Given a set of examples $E \subseteq \mathbb{R} \times \mathbb{R}$, the task of finding $h_{\mathbf{w}}$ that best fits E is called **linear regression**.
- ▷ **Example 25.7.6**

Examples of house price vs. square feet in houses sold in Berkeley in July 2009.

Also: linear function hypothesis that minimizes squared error loss $y = 0.232 x + 246$.



Univariate Linear Regression by Loss Minimization

- ▷ **Idea:** minimize squared error loss over $\{(x_i, y_i) | i \leq N\}$ (**used already by Gauss**)

$$\text{Loss}(h_{\mathbf{w}}) = \sum_{j=1}^N L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^N (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2$$

Task: find $\mathbf{w}^* := \underset{\mathbf{w}}{\operatorname{argmin}}(\text{Loss}(h_{\mathbf{w}}))$.

- ▷ **Recall:** $\sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2$ is minimized, when the partial derivatives wrt. the w_i are zero, i.e. when

$$\frac{\partial}{\partial w_0} \left(\sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2 \right) = 0 \quad \text{and} \quad \frac{\partial}{\partial w_1} \left(\sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2 \right) = 0$$

- ▷ **Observation:** These equations have a unique solution:

$$w_1 = \frac{N \sum_j x_j y_j - \sum_j x_j \sum_j y_j}{N \sum_j x_j^2 - \sum_j x_j^2} \quad w_0 = \frac{\sum_j y_j - w_1 \sum_j x_j}{N}$$

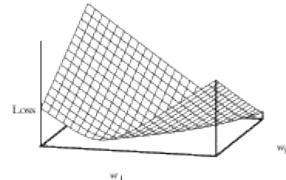
- ▷ **Remark:** Closed-form solutions only exist for linear regression, for other (differentiable) hypothesis spaces use gradient descent methods.



A Picture of the Weight Space

- ▷ Many forms of learning involve adjusting weights to minimize loss
- ▷ **Definition 25.7.7** The **weight space** is the space of all possible combinations of weights

- ▷ The weight space of univariate linear regression is \mathbb{R}^2 .
 ↳ graph the loss function over \mathbb{R}^2 .
 Note: it is **convex**



- ▷ **Observation 25.7.8** The squared error loss function is convex for any linear regression problem ↳ there are no local minima.



Gradient Descent Methods

- ▷ If we do not have closed form solutions for minimizing loss, we need to search.
- ▷ **Idea:** use local search (hill climbing) methods
- ▷ **Definition 25.7.9** The **gradient descent** algorithm for finding a minimum of a continuous function f is hill-climbing in the direction of the steepest descent, which can be computed by the partial derivatives of f .

```

function gradient-descent( $f, \mathbf{w}, \alpha$ ) returns a local minimum of  $f$ 
  inputs: a differentiable function  $f$  and initial weights  $\mathbf{w} = (w_0, w_1)$ .
  loop until  $\mathbf{w}$  converges do
    for each  $w_i$  do
       $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i}(f(\mathbf{w}))$ 
    end for
  end loop

```

The parameter α is called the **learning rate**. It can be a fixed constant or it can decay as learning proceeds.



Gradient-Descent for Loss

- ▷ Let's try gradient descent for Loss.
- ▷ Work out the partial derivatives for one example (x, y) :

$$\frac{\partial \text{Loss}(\mathbf{w})}{\partial w_i} = \frac{\partial((y - h_{\mathbf{w}}(x))^2)}{\partial w_i} = 2(y - h_{\mathbf{w}}(x)) \frac{\partial(y - (w_1 x + w))}{\partial w_i}$$

and thus

$$\frac{\partial \text{Loss}(\mathbf{w})}{\partial w_0} = -2(y - h_{\mathbf{w}}(x)) \quad \frac{\partial \text{Loss}(\mathbf{w})}{\partial w_1} = -2(y - h_{\mathbf{w}}(x)) x$$

Plug this into the gradient descent updates:

$$w_0 \leftarrow w_0 - \alpha (-2(y - h_{\mathbf{w}}(x))) \quad w_1 \leftarrow w_1 - \alpha (-2(y - h_{\mathbf{w}}(x))) x$$



Gradient-Descent for Loss (continued)

- ▷ Analogously for n training examples (x_j, y_j) :

Definition 25.7.10

$$w_0 \leftarrow w_0 - \alpha \sum_j -2(y_j - h_{\mathbf{w}}(x_j)) \quad w_1 \leftarrow w_1 - \alpha \sum_j -2(y_j - h_{\mathbf{w}}(x_n)) x_n$$

These updates constitute the **batch gradient-descent learning rule** for univariate linear regression.

- ▷ Convergence to the unique global minimum is guaranteed (as long as we pick α small enough) but may be very slow.



Multivariate Linear Regression

- ▷ **Definition 25.7.11** A **multivariate** or **n -ary** function is a function with one or more arguments.
- ▷ We can use it for **multivariate linear regression**.
- ▷ **Idea:** Every example \vec{x}_j is an n -element vector and the hypothesis space is the set of functions

$$h_{sw}(\vec{x}_j) = w_0 + w_1 x_{j,1} + \dots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}$$

▷ Trick: Invent $x_{j,0} := 1$ and use matrix notation:

$$h_{sw}(\vec{x}_j) = \vec{w} \cdot \vec{x}_j = \vec{w}^t \vec{x}_j = \sum_i w_i x_{j,i}$$

▷ Definition 25.7.12 The best vector of weights, \mathbf{w}^* , minimizes squared-error loss over the examples: $\mathbf{w}^* := \underset{\mathbf{w}}{\operatorname{argmin}}(\sum_j L_2(y_j, \mathbf{w} \cdot \vec{x}_j))$.

▷ Gradient descent will reach the (unique) minimum of the loss function; the update equation for each weight w_i is

$$w_i \leftarrow w_i - \alpha \sum_j x_{j,i} (y_j - h_{\mathbf{w}}(\vec{x}_j))$$



©: Michael Kohlhase

797



Multivariate Linear Regression (Analytic Solutions)

▷ We can also solve analytically for the \mathbf{w}^* that minimizes loss.

▷ Let \vec{y} be the vector of outputs for the training examples, and \mathbf{X} be the data matrix, i.e., the matrix of inputs with one n -dimensional example per row.

Then the solution $\mathbf{w}^* = (\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t \vec{y}$ minimizes the squared error.



©: Michael Kohlhase

798



Multivariate Linear Regression (Regularization)

▷ Remark: Univariate linear regression does not overfit, but in the multivariate case there might be “redundant dimensions” that result in overfitting.

▷ Idea: Use regularization with a complexity function based on weights.

▷ Definition 25.7.13 Complexity($h_{\mathbf{w}}$) = $L_q(\mathbf{w}) = \sum_i |w_i|^q$

▷ Caveat: Do not confuse this with the loss functions L_1 and L_2 .

▷ Problem: Which q should be pick? (L_1 and L_2 minimize sum of absolute values/squares)

▷ Answer: depends on the application.

▷ Remark: L_1 -regularization tends to produce a sparse model, i.e. it sets many weights to 0, effectively declaring the corresponding attributes to be irrelevant.

Hypotheses that discard attributes can be easier for a human to understand, and may be less likely to overfit. (see [RN03, Section 18.6.2])



©: Michael Kohlhase

799



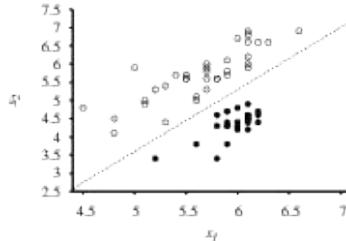
Linear Classifiers with a hard Threshold

▷ **Idea:** The result of [linear regression](#) can be used for classification.

▷ **Example 25.7.14**

Plots of seismic data parameters: body wave magnitude x_1 vs. surface wave magnitude x_2 . White: earthquakes, black: underground explosions

Also: $h_{\mathbf{w}^*}$ as a decision boundary
 $x_2 = 17x_1 - 4.9$.



▷ **Definition 25.7.15** A [decision boundary](#) is a line (or a surface, in higher dimensions) that separates two classes of points. A linear decision boundary is called a [linear separator](#) and data that admits one are called [linearly separable](#).

▷ **Example 25.7.16** The linear separator is defined by $-4.9 + 1.7x_1 - x_2 = 0$, the white points (explosions) are characterized by $-4.9 + 1.7x_1 - x_2 > 0$, the black ones (earthquakes) by $-4.9 + 1.7x_1 - x_2 < 0$.

▷ If we introduce dummy coordinate $x_0 = 1$, then we can write the classification hypothesis as $h_{\mathbf{w}}(\mathbf{x}) = 1$ if $\mathbf{w} \cdot \mathbf{x} > 0$ and 0 otherwise.



Linear Classifiers with a hard Threshold (Perceptron Rule)

▷ So $h_{\mathbf{w}}(\mathbf{x}) = 1$ if $\mathbf{w} \cdot \mathbf{x} > 0$ and 0 otherwise is well-defined, how to choose \mathbf{w} ?

▷ Think of $h_{\mathbf{w}}(\mathbf{x}) = \mathcal{T}(\mathbf{w} \cdot \mathbf{x}) > 0$, where $\mathcal{T}(z) = 0$, if $z \geq 0$ and $\mathcal{T}(z) = 0$ otherwise. We call \mathcal{T} a [threshold function](#).

▷ **Problem:** \mathcal{T} is not differentiable and $\frac{\partial \mathcal{T}}{\partial z} = 0$ where defined

▷ no closed-form solutions by setting $\frac{\partial \mathcal{T}}{\partial z} = 0$ and solving.

▷ gradient-descent methods in weight-space do not work either

▷ We can learn weights by iterating over the following rule:

▷ **Definition 25.7.17** Given an example (\mathbf{x}, y) , the [perceptron learning rule](#) is

$$w_i \leftarrow w_i + \alpha \cdot (y - h_{\mathbf{w}}(\mathbf{x})) \cdot x_i$$

▷ As we are considering 0/1 classification, there are three possibilities

▷ if $y = h_{\mathbf{w}}(\mathbf{x})$, then w_i remain unchanged

▷ if $y = 1$ and $h_{\mathbf{w}}(\mathbf{x}) = 0$, then w_i is increased if x_i is positive/negative
 (we want to make $\mathbf{w} \cdot \mathbf{x}$ bigger so that $\mathcal{T}(\mathbf{w} \cdot \mathbf{x}) = 1$)

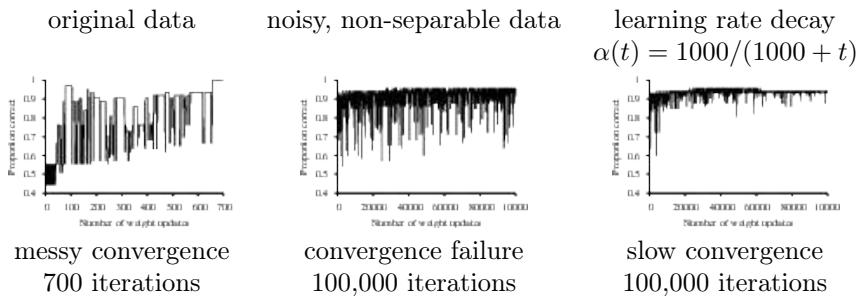
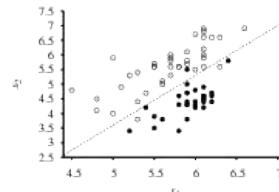
- ▷ if $y = 0$ and $h_{\mathbf{w}}(\mathbf{x}) = 1$, then w_i is de/increased if x_i is positive/negative
(we want to make $\mathbf{w} \cdot \mathbf{x}$ smaller so that $\mathcal{T}(\mathbf{w} \cdot \mathbf{x}) = 0$)



Learning Curves for Linear Classifiers (Perceptron Rule)

▷ Example 25.7.18

Learning curves (plots of total training-set accuracy vs. number of iterations) for the perceptron rule on the earthquake/explosions data.



- ▷ **Theorem 25.7.19** *Finding the minimal-error is NP-hard, but possible with learning rate decay.*



Linear Classification with Logistic Regression

- ▷ **So far:** passing the output of a linear function through a threshold function \mathcal{T} yields a linear classifier.
- ▷ **Problem:** the hard nature of \mathcal{T} brings problems
 - ▷ \mathcal{T} is not differentiable nor continuous \leadsto learning via perceptron rule becomes unpredictable
 - ▷ \mathcal{T} is “overly precise” near the boundary \leadsto need more graded judgements
- ▷ **Idea:** soften the threshold, approximate it with a differentiable function.

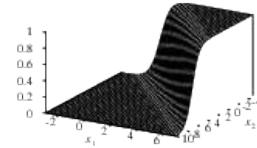
We use the **standard logistic function** $l(x) = \frac{1}{1+e^{-x}}$ So we have $h_{\mathbf{w}}(\mathbf{x}) = l(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w} \cdot \mathbf{x})}}$



▷ Example 25.7.20 (Logistic Regression Hypothesis in Weight Space)

Plot of a logistic regression hypothesis for the earthquake/explosion data.

The value at (w_0, w_1) is the probability of belonging to the class labeled 1.



We speak of the **cliff** in the classifier intuitively.



Logistic Regression

- ▷ **Definition 25.7.21** The process of weight-fitting in $h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x})}}$ is called **logistic regression**.
- ▷ There is no easy closed form solution, but gradient descent is straightforward,
- ▷ As our hypotheses have continuous output, we use the **squared error loss** function L_2 .
- ▷ For an example (\mathbf{x}, y) we compute the **partial derivatives**: (via chain rule)

$$\begin{aligned}\frac{\partial}{\partial w_i}(L_2(\mathbf{w})) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\ &= 2 \cdot h_{\mathbf{w}}(\mathbf{x}) \cdot \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) \\ &= -2 \cdot h_{\mathbf{w}}(\mathbf{x}) \cdot l'(\mathbf{w} \cdot \mathbf{x}), \frac{\partial}{\partial w_i}(\mathbf{w} \cdot \mathbf{x}) \\ &= -2 \cdot h_{\mathbf{w}}(\mathbf{x}) \cdot l'(\mathbf{w} \cdot \mathbf{x}) \cdot x_i\end{aligned}$$



Logistic Regression (continued)

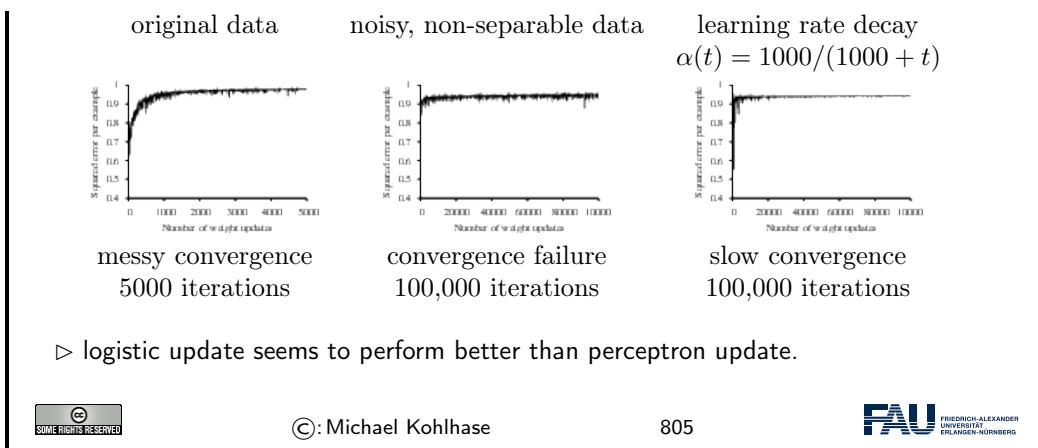
- ▷ The derivative of the **logistic function** satisfies $l'(z) = l(z)(1 - l(z))$, thus

$$l'(\mathbf{w} \cdot \mathbf{x}) = l(\mathbf{w} \cdot \mathbf{x})(1 - l(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

So the weight update for minimizing the loss is

$$w_i \leftarrow w_i + \alpha \cdot (y - h_{\mathbf{w}}(\mathbf{x})) \cdot h_{\mathbf{w}}(\mathbf{x}) \cdot (1 - h_{\mathbf{w}}(\mathbf{x})) \cdot x_i$$

- ▷ Example 25.7.22 (Redoing the Training Curves)



25.8 Artificial Neural networks

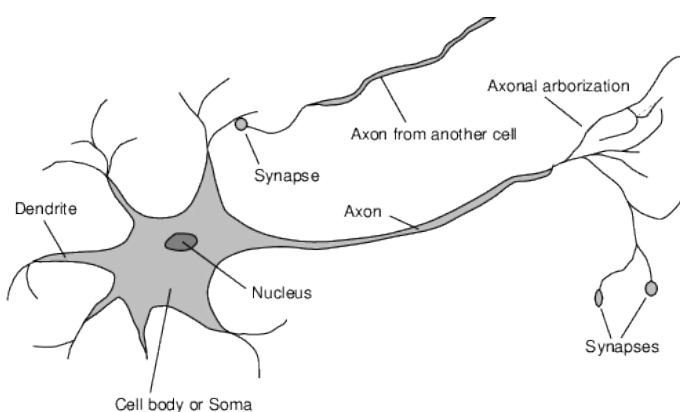
Outline

- ▷ Brains
- ▷ Neural networks
- ▷ Perceptrons
- ▷ Multilayer perceptrons
- ▷ Applications of neural networks



Brains

- ▷ **Axiom 25.8.1 (Neuroscience Hypothesis)** Mental activity consists primarily of electrochemical activity in networks of brain cells called **neurons**.



- ▷ **Definition 25.8.2** The animal brain is a **biological neural network**

- ▷ with 10^{11} neurons of > 20 types, 10^{14} synapses, 1ms – 10ms cycle time.
- ▷ Signals are noisy “spike trains” of electrical potential



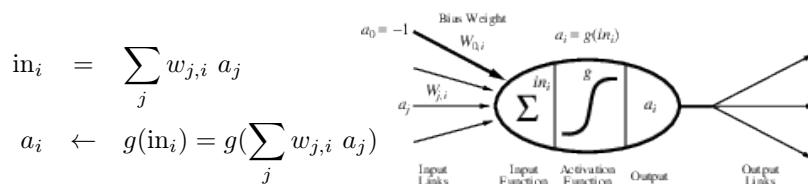
Neural Networks as an approach to Artificial Intelligence

- ▷ One approach to Artificial Intelligence is to model and simulate brains (and hope that AI comes along naturally)
- ▷ **Definition 25.8.3** The AI sub-field of neural networks (also called connectionism, parallel distributed processing, and neural computation) studies computing systems inspired by the biological neural networks that constitute brains.
- ▷ neural networks are attractive computational devices, since they perform important AI tasks – most importantly learning and distributed, noise-tolerant computation – naturally and efficiently.



Neural Networks – McCulloch-Pitts “unit”

- ▷ **Definition 25.8.4** An artificial neural network is a directed graph of units and links. A link from unit i to unit j propagates the activation a_i from unit i to unit j , it has a weight $w_{i,j}$ associated with it.
- ▷ In 1943 McCulloch and Pitts proposed a simple model for a neuron/brain
- ▷ **Definition 25.8.5** A McCulloch-Pitts unit first computes a weighted sum of all inputs and then applies an activation function g to it.



If g is a threshold function, we call the unit a perceptron unit, if g is a logistic function a sigmoid perceptron unit.

A McCullorch-Pitts network is a neural network with McCullorch-Pitts units.



Implementing Logical Functions as Units

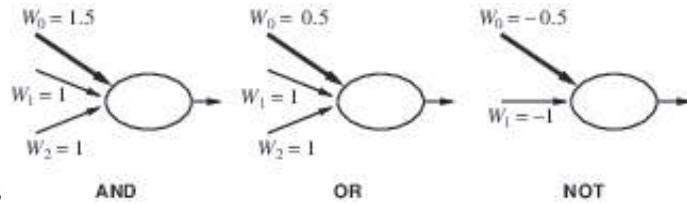
- ▷ McCullorch-Pitts units are a gross oversimplification of real neurons, but its purpose is to develop understanding of what neural networks of simple units can do.

▷ **Theorem 25.8.6 (McCulloch and Pitts)** Every *Boolean function* can be implemented as *McCulloch-Pitts networks*.

▷ **Proof:** by construction

P.1 Recall that $a_i \leftarrow g(\sum_j w_{j,i} a_j)$.

P.2 As for linear regression we use $a_0 = 1 \rightsquigarrow w_{0,i}$ as a **bias weight** (or **intercept**) (determines the threshold)



P.4 Any *Boolean function* can be implemented as a DAG of these units. □



Network Structures: Feed-Forward Networks

▷ We have models for neurons/units \rightsquigarrow connect them to neural networks.

▷ **Definition 25.8.7** A neural network is called a **feed-forward network**, if it is **acyclic**.

▷ **Intuition:** feed-forward networks implement functions, have no internal state

▷ **Definition 25.8.8** Feed-forward networks are usually organized in **layers**: a **n-layer** network has a **partition** $\{L_0, \dots, L_n\}$ of the **nodes**, such that **edges** only connect **nodes** from subsequent layer.

▷ **Definition 25.8.9** L_0 is called the **input layer** and its members **input units**, and L_n the **output layer** and its members **output units**. Any unit that is not in the input layer or the output layer is called **hidden**.



Network Structures: Recurrent Networks

▷ **Definition 25.8.10** A neural network is called **recurrent**, iff it has **cycles**.

▷ **Hopfield networks** have symmetric weights ($w_{i,j} = w_{j,i}$) $g(x) = \text{sign}(x)$, $a_i = \pm 1$; (holographic associative memory)

▷ **Boltzmann machines** use stochastic activation functions, \approx Monte Carlo Markov Chains (MCMC) in Bayes nets

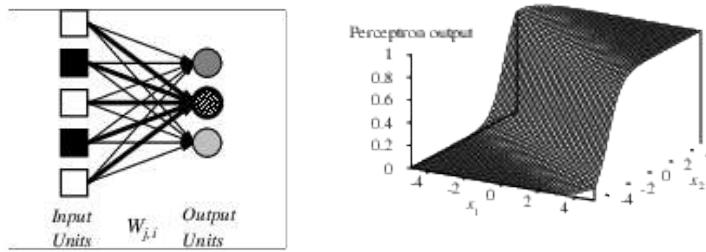
▷ Recurrent neural nets have directed cycles with delays \rightsquigarrow have internal state (like flip-flops), can oscillate etc.



Single-layer Perceptrons

▷ **Definition 25.8.11** A **perceptron network** is a feed-forward network of **perceptron units**. A **single-layer** perceptron network is called a **perceptron**.

▷ **Example 25.8.12**



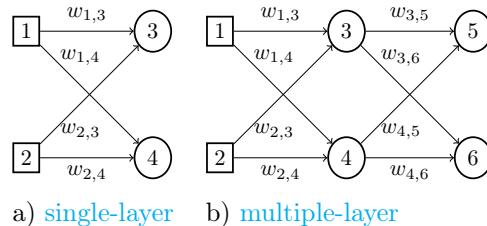
- ▷ All input units are directly connected to output units.
- ▷ Output units all operate separately, no shared weights \leadsto treat as the combination of n **perceptron units**.
- ▷ Adjusting weights moves the location, orientation, and steepness of cliff.



Feed-forward Neural Networks (Example)

▷ Feed-forward network $\hat{=}$ a parameterized family of nonlinear functions:

▷ **Example 25.8.13** We show two perceptron networks:



a) single-layer b) multiple-layer

$$\begin{aligned} a_5 &= g(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4) \\ &= g(w_{3,5} \cdot g(w_{1,3} \cdot a_1 + w_{2,3} \cdot a_2) + w_{4,5} \cdot g(w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2)) \end{aligned}$$

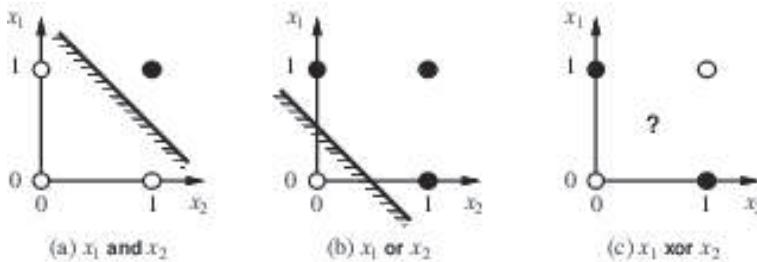
▷ **Idea:** Adjusting weights changes the function: do learning this way!



Expressiveness of perceptrons

- ▷ Consider a perceptron with $g = \text{step function}$ (Rosenblatt, 1957, 1960)
- ▷ Can represent AND, OR, NOT, majority, etc., but not XOR (and thus no adders)
- ▷ Represents a linear separator in input space:

$$\sum_j w_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$



- ▷ Minsky & Papert (1969) pricked the first neural network balloon!



Perceptron Learning

- ▷ Idea: Wlog. treat only single-output perceptrons $\sim \mathbf{w}$ is a “weight vector”. Learn by adjusting weights in \mathbf{w} to reduce generalization loss on training set.
- ▷ Let us compute with the squared error loss of a weight vector \mathbf{w} for an example (\mathbf{x}, y) .

$$\text{Loss}(\mathbf{w}) = \text{Err}^2 = (y - h_{\mathbf{w}}(\mathbf{x}))^2$$

- ▷ Perform optimization search by gradient descent for any weight w_i :

$$\begin{aligned} \frac{\partial \text{Loss}(\mathbf{w})}{\partial w_j} &= 2 \cdot \text{Err} \cdot \frac{\partial \text{Err}}{\partial w_j} = 2 \cdot \text{Err} \cdot \frac{\partial}{\partial w_j} (y - g(\sum_{j=0}^n w_j x_j)) \\ &= -2 \cdot \text{Err} \cdot g'(\mathbf{in}_i) \cdot x_j \end{aligned}$$

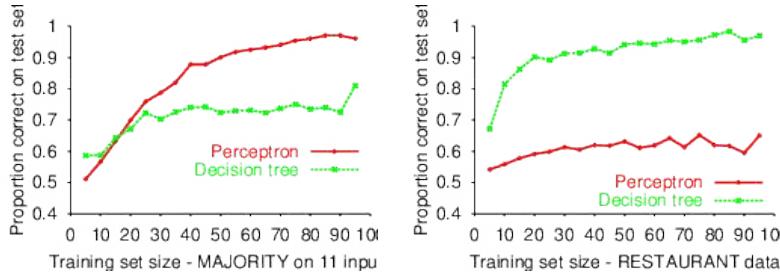
- ▷ Simple weight update rule:

$$w_{j,k} \leftarrow w_{j,k} + \alpha \cdot \text{Err} \cdot g'(\mathbf{in}_i) \cdot x_j$$



Perceptron learning contd.

- ▷ Perceptron learning rule converges to a consistent function for any *linearly separable data set*

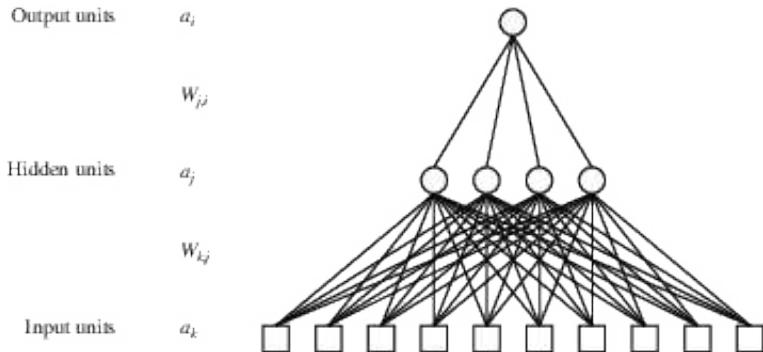


- ▷ Perceptron learns majority function easily, DTL is hopeless
- ▷ DTL learns restaurant function easily, perceptron cannot represent it



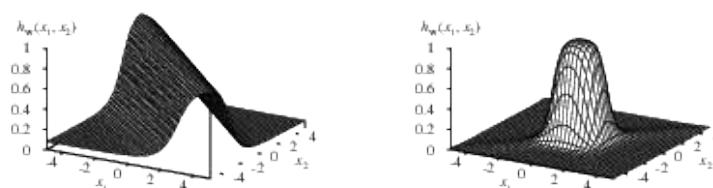
Multilayer perceptrons

- ▷ **Definition 25.8.14** Layers are usually fully connected; numbers of **hidden units** typically chosen by hand



Expressiveness of MLPs

- ▷ All continuous functions w/ 2 layers, all functions w/ 3 layers



- ▷ Combine two opposite-facing threshold functions to make a ridge
- ▷ Combine two perpendicular ridges to make a bump
- ▷ Add bumps of various sizes and locations to fit any surface
- ▷ Proof requires exponentially many hidden units (cf. DTL proof)



Learning in Multilayer Networks (Output Layer)

- ▷ **Idea:** Learn by adjusting weights to reduce error on training set.
- ▷ **Problem:** neural networks have multiple outputs.
- ▷ we use \mathbf{h}_w with output vector \mathbf{y} .
- ▷ The squared error loss of a weight matrix w for an example (\mathbf{x}, \mathbf{y}) is

$$\text{Loss}(\mathbf{w}) = \|(\mathbf{y} - \mathbf{h}_w(\mathbf{x}))\|_2^2 = \sum_{k=1}^n (y_k - a_k)^2$$

- ▷ **Output layer:** analogous to that for single-layer perceptron, but multiple output units

$$w_{j,i} \leftarrow w_{j,i} + \alpha \cdot a_j \cdot \Delta_i$$

where $\Delta_i = Err_i \cdot g'(\mathbf{in}_i)$ and $Err = \mathbf{y} - \mathbf{h}_w(\mathbf{x})$. (error vector)



Learning in Multilayer Networks (Hidden Layers)

- ▷ **Problem:** The error Err is well-defined only for the output layer \rightsquigarrow the examples do not say anything about the hidden layers
- ▷ **Idea:** back-propagate the error from the output layer; actually back-propagate Δ_k
The hidden node j is “responsible” for some fraction of Δ_k (by connection weight)
- ▷ **Definition 25.8.15** The back-propagation rule for hidden nodes of a multilayer perceptron is

$$\Delta_j \leftarrow g'(\mathbf{in}_i) \cdot \sum_i w_{j,i} \Delta_i$$

- ▷ Update rule for weights in hidden layer:

$$w_{k,j} \leftarrow w_{k,j} + \alpha \cdot a_k \cdot \Delta_j$$

- ▷ **Remark:** Most neuroscientists deny that back-propagation occurs in the brain.



Back-Propagation Process

▷ The back-propagation process can be summarized as follows:

1. Compute the Δ values for the output units, using the observed error.
2. Starting with output layer, repeat the following for each layer in the network, until the earliest hidden layer is reached:
 - (a) Propagate the Δ values back to the previous (hidden) layer.
 - (b) Update the weights between the two layers.

▷ Details (algorithm) later.



Backpropagation-Learning Algorithm

▷ **Definition 25.8.16** The **back-propagation learning algorithm** is given the following pseudocode

```

function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with L layers, weights  $w_{i,j}$ , activation function g
  local variables:  $\Delta$ , a vector of errors, indexed by network node
  repeat
    foreach weight  $w_{i,j}$  in network do
       $w_{i,j} :=$  a small random number
    foreach example (x, y) in examples do
      /* Propagate the inputs forward to compute the outputs */
      foreach node i in the input layer do  $a_i := x_i$ 
      for l = 2 to L do
        foreach node j in layer l do
           $in_i := \sum_i w_{i,j} a_i$ 
           $a_j := g(in_i)$ 
      /* Propagate deltas backward from output layer to input layer */
      foreach node j in the output layer do  $\Delta[j] := g'(in_i) \cdot (y_j - a_j)$ 
      for l = L - 1 to 1 do
        foreach node i in layer l do  $\Delta[i] := g'(in_i) \cdot \sum_j w_{i,j} \Delta[j]$ 
      /* Update every weight in network using deltas */
      foreach weight  $w_{i,j}$  in network do  $w_{i,j} := w_{i,j} + \alpha \cdot a_i \cdot \Delta[j]$ 
    until some stopping criterion is satisfied
  return network

```



Back-Propagation Derivation from First Principles

- ▷ This is very similar to the gradient calculation for logistic regression
- ▷ Compute the loss gradient wrt. the weights between the output and hidden

layers:

$$\begin{aligned}
 \frac{\partial Loss_k}{\partial w_{j,k}} &= -2 (y_k - a_k) \frac{\partial a_k}{\partial w_{j,k}} = -2 (y_k - a_k) \frac{\partial g(\text{in}_i)}{\partial w_{j,k}} \\
 &= -2 (y_k - a_k) g'(\text{in}_i) \frac{\partial \text{in}_i}{\partial w_{j,k}} \\
 &= -2 (y_k - a_k) g'(\text{in}_i) \frac{\partial}{\partial w_{j,k}} \left(\sum_j w_{j,k} a_j \right) \\
 &= -2 (y_k - a_k) g'(\text{in}_i) a_j = -2 a_j \Delta_k
 \end{aligned}$$



Back-propagation derivation contd.

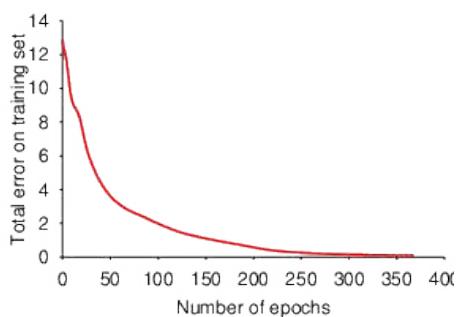
▷ we continue computing where we left off

$$\begin{aligned}
 \frac{\partial Loss_k}{\partial w_{i,j}} &= -2 (y_k - a_k) \frac{\partial a_k}{\partial w_{i,j}} = -2 (y_k - a_k) \frac{\partial g(\text{in}_i)}{\partial w_{i,j}} \\
 &= -2 (y_k - a_k) g'(\text{in}_i) \frac{\partial \text{in}_i}{\partial w_{i,j}} = -2 \Delta_k \frac{\partial}{\partial w_{i,j}} \left(\sum_j w_{j,i} a_j \right) \\
 &= -2 \Delta_k w_{j,k} \frac{\partial a_j}{\partial w_{i,j}} = -2 \Delta_k w_{j,k} \frac{\partial g(\text{in}_i)}{\partial w_{i,j}} \\
 &= -2 \Delta_k w_{j,k} g'(\text{in}_i) \frac{\partial \text{in}_i}{\partial w_{i,j}} \\
 &= -2 \Delta_k w_{j,k} g'(\text{in}_i) \frac{\partial}{\partial w_{i,j}} \left(\sum_k w_{i,j} a_k \right) \\
 &= -2 \Delta_k w_{j,k} g'(\text{in}_i) a_k = -a_k \Delta_j
 \end{aligned}$$



Back-Propagation – Properties

- ▷ At each **epoch**, sum gradient updates for all examples and apply
- ▷ **training curve** for 100 restaurant examples: finds exact fit



- ▷ Typical problems: slow convergence, local minima



Back-Propagation – Properties (contd.)

- ▷ Learning curve for MLP with 4 hidden units:



- ▷ MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily



Handwritten digit recognition

0	1	2	3	4	5	6	7	8	9
0	1	0	3	4	5	6	7	9	8

- ▷ 400–300–10 unit MLP = 1.6% error
- ▷ LeNet: 768–192–30–10 unit MLP = 0.9% error
- ▷ Current best (kernel machines, vision algorithms) $\approx 0.6\%$ error



Summary

- ▷ Most brains have lots of neurons; each neuron \approx linear-threshold unit (?)
- ▷ Perceptrons (one-layer networks) insufficiently expressive
- ▷ Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation
- ▷ Many applications: speech, driving, handwriting, fraud detection, etc.
- ▷ Engineering, cognitive modelling, and neural system modelling subfields have largely diverged



XKCD on Machine Learning

- ▷ A Skeptics View: see <https://xkcd.com/1838/>



25.9 Support Vector Machines

Support Vector Machines

- ▷ Currently the most popular approach for “off-the-shelf” supervised learning
- ▷ SVMs construct a **maximum margin separator** a decision boundary with the

largest possible distance to example points. This helps them generalize well.

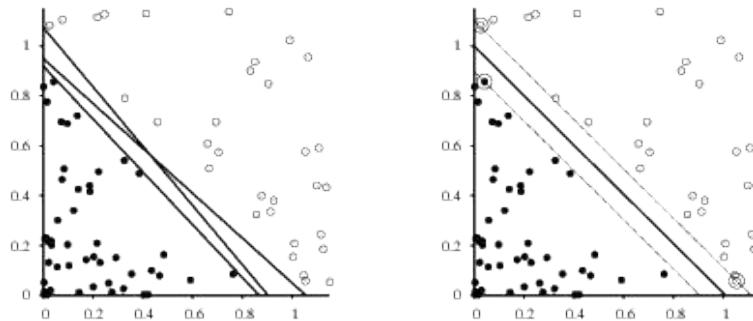
- ▷ SVMs can embed data into a higher-dimensional space, where it is linearly separable by the *kernel trick* \leadsto the separating hyperplane is hyper-surface in original data.
- ▷ SVMs prioritize critical examples (*support vectors*) (better generalization)



Support Vector Machines (Separation with Margin)

- ▷ **Definition 25.9.1** Given a linearly separable data set E the **maximal margin separator** is the **linear separator** s that maximizes the **margin**, i.e. the distance of the E from s .

- ▷ **Example 25.9.2 (Linear Separators)**



Finding the Maximum Margin Separator

- ▷ Before we see now find the Maximum Margin Separator
- ▷ **Notations and Conventions:** We use label classes “+1” and “-1” instead of “1” and “0”.
We keep the **intercept** as a separate parameter: The linear separator is represented as the set $\{x \mid (w \cdot x) + b = 0\}$.
- ▷ **Idea:** Use gradient decent to search the space of all w and b for maximizing combinations.
- ▷ This works, but we follow a different route (that shows more of the structure)



Finding the Maximum Margin Separator (Separable Case)

- ▷ **Alternative Representation:** Find the optimal solution by solving the **SVM equation**

$$\underset{\alpha}{\operatorname{argmax}} \left(\sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k) \right)$$

under the constraints $\alpha_j \geq 0$ and $\sum_j \alpha_j y_j = 0$.

- ▷ **Observations:** This equation has three important properties

1. the expression is **convex** \rightsquigarrow the single global maximum can be found efficiently.
2. data enter the expression only in the form of dot products of point pairs \rightsquigarrow once the optimal α_i have been calculated, we have

$$h(\mathbf{x}) = \operatorname{sign} \left(\sum_j \alpha_j y_j (\mathbf{x} \cdot \mathbf{x}_j) - b \right)$$

3. the weights α_j associated with each data point are zero except at the **support vectors** – the points closest to the separator.

- ▷ There are good software packages for solving such **quadratic programming** optimizations

- ▷ Once we found an optimal vector α , use $\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$.



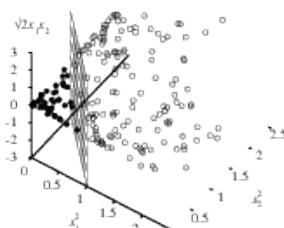
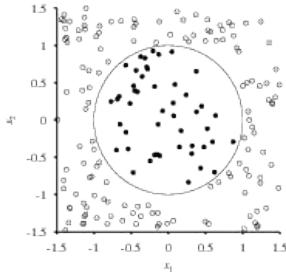
Support Vector Machines (Kernel Trick)

- ▷ **Problem:** What if the data is not **linearly separable**?

- ▷ **Idea:** Transform the data into a higher-dimensional space, where they are.

- ▷ **Example 25.9.3 (Projecting Up a Non-Separable Data Set)**

left: The true decision boundary is $x_1^2 + x_2^2 \leq 1$.



right: mapping into a three-dimensional input space $\langle x_1^2, x_2^2, \sqrt{2} x_1 x_2 \rangle \rightsquigarrow$ separable by a hyperplane.

- ▷ **Upshot:** We map each input vector \mathbf{x} to a $F(\mathbf{x})$ with $f_1 = x_1^2$, $f_2 = x_2^2$, and $f_3 = \sqrt{2} x_1 x_2$.



Support Vector Machines (Kernel Trick continued)

- ▷ **Idea:** Replace $\mathbf{x}_j \cdot \mathbf{x}_j$ by $F(\mathbf{x}_j) \cdot F(\mathbf{x}_j)$ in the SVM equation. (compute in high-dim space.)
- ▷ Often we can compute $F(\mathbf{x}_j) \cdot F(\mathbf{x}_j)$ without computing F everywhere.
- ▷ **Example 25.9.4** If $F(\mathbf{x}) = \langle x_1^2, x_2^2, \sqrt{2} x_1 x_2 \rangle$, then $F(\mathbf{x}_j) \cdot F(\mathbf{x}_j) = (\mathbf{x}_j \cdot \mathbf{x}_j)^2$ (have added the $\sqrt{2}$ in F so that this works)
- ▷ **Definition 25.9.5** We call the function $(\mathbf{x}_j \cdot \mathbf{x}_j)^2$ a **Kernel function**. (there are others)
- ▷ **Generally:** We can learn non-linear separators by solving

$$\underset{\alpha}{\operatorname{argmax}} \left(\sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k K(\mathbf{x}_j, \mathbf{x}_k) \right)$$

where K is a **kernel function**

- ▷ **Definition 25.9.6** The function $K(\mathbf{x}_j, \mathbf{x}_k) = (1 + (\mathbf{x}_j \cdot \mathbf{x}_k))^d$ is a kernel function corresponding to a feature space whose dimension is exponential in d . It is called the **polynomial kernel**.
- ▷ **Theorem 25.9.7 (Mercer's Theorem)** Every kernel function where $K(\mathbf{x}_j, \mathbf{x}_k)$ is positive definite corresponds to some feature space.



Summary

- ▷ Learning needed for unknown environments, lazy designers
- ▷ Learning agent = performance element + learning element
- ▷ Learning method depends on type of performance element, available feedback, type of component to be improved, and its representation
- ▷ For supervised learning, the aim is to find a simple hypothesis that is approximately consistent with training examples
- ▷ Decision tree learning using information gain
- ▷ Learning performance = prediction accuracy measured on test set
- ▷ PAC learning as a general theory of learning boundaries.
- ▷ Linear regression (hypothesis space of univariate linear functions)

▷ Linear classification by linear regression with hard and soft thresholds



©: Michael Kohlhase

837



Chapter 26

Statistical Learning

In Part V we learned how to reason in non-deterministic, partially observable environments by quantifying uncertainty and reasoning with it. The key resource there were probabilistic models and their efficient representations: Bayesian networks.

In Part V we assumed that these models were given, perhaps designed by the agent developer. We will now learn how these models can – at least partially – be learned from observing the environment.

Outline

- ▷ Bayesian learning, i.e. learning probabilistic models (e.g. Bayesian networks) from observations.
- ▷ Maximum *a posteriori* and maximum likelihood learning
- ▷ Bayes net learning
 - ▷ ML Parameter Learning with Complete Data
 - ▷ Linear Regression
 - ▷ Naive Bayes Models/Learning



©: Michael Kohlhase

838



26.1 Full Bayesian Learning

Candy Flavors Example

- ▷ Suppose there are five kinds of bags of candies:
 1. 10% are h_1 : 100% cherry candies
 2. 20% are h_2 : 75% cherry candies + 25% lime candies
 3. 40% are h_3 : 50% cherry candies + 50% lime candies
 4. 20% are h_4 : 25% cherry candies + 75% lime candies
 5. 10% are h_5 : 100% lime candies
- ▷ Then we observe candies drawn from some bag:

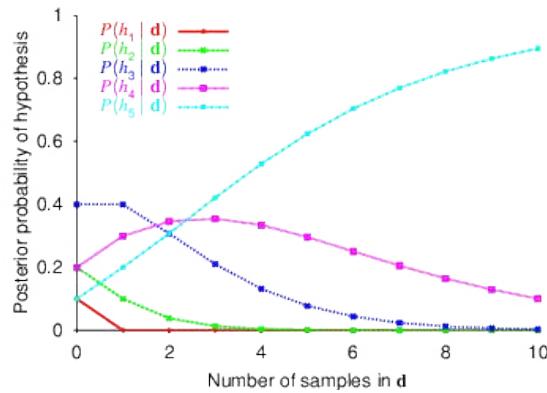


▷ What kind of bag is it? What flavour will the next candy be?



Candy Flavors: Posterior probability of hypotheses

▷ **Example 26.1.1** The probability of hypothesis h_i after n limes are observe
 $\hat{=}$



if the observation are IID, i.e. $P(\mathbf{d} | h_i) = \prod_j P(d_j | h_i)$ and the [hypothesis prior](#) is as advertised.
(e.g. $P(h_3 | \mathbf{d}) = 0.5^{10} = 0.1\%$)

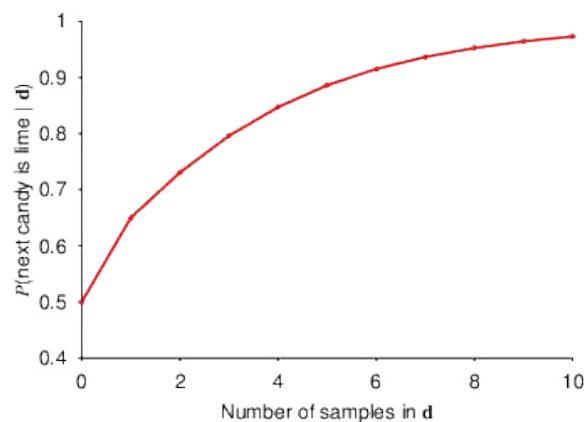
The posterior probabilities start with the [hypothesis priors](#), change with data.



Candy Flavors: Prediction Probability

▷ We calculate that the $n + 1$ -th candy is lime:

$$\mathbf{P}(d_{n+1} = \text{lime} | \mathbf{d}) = \sum_i \mathbf{P}(d_{n+1} = \text{lime} | h_i) \cdot P(h_i | \mathbf{d})$$



Full Bayesian Learning

- ▷ **Idea:** View learning as Bayesian updating of a probability distribution over the hypothesis space
 - ▷ H is the hypothesis variable, values h_1, h_2, \dots , prior $\mathbf{P}(H)$.
 - ▷ j th observation d_j gives the outcome of random variable D_j training data d_1, \dots, d_N
 - ▷ Given the data so far, each hypothesis has a posterior probability:

$$P(h_i | \mathbf{d}) = \alpha \cdot P(\mathbf{d} | h_i) \cdot P(h_i)$$

where $P(\mathbf{d} | h_i)$ is called the **likelihood** (of the data under each hypothesis) and $P(h_i)$ the **hypothesis prior**.

- ▷ Predictions use a likelihood-weighted average over the hypotheses:

$$\mathbf{P}(\mathbf{X} | \mathbf{d}) = \sum_i \mathbf{P}(\mathbf{X} | \mathbf{d}, h_i) \cdot P(h_i | \mathbf{d}) = \sum_i \mathbf{P}(\mathbf{X} | h_i) \cdot P(h_i | \mathbf{d})$$

- ▷ No need to pick one best-guess hypothesis!



Full Bayesian Learning: Properties

- ▷ **Observation:** The Bayesian prediction eventually agrees with the true hypothesis.
- ▷ The probability of generating “uncharacteristic” data indefinitely is vanishingly small.
- ▷ Argument analogous to PAC learning
- ▷ **Problem:** Summing over the hypothesis space is often intractable

▷ **Example 26.1.2** e.g., 18,446,744,073,709,551,616 Boolean functions of 6 attributes

▷ **Solution:** approximate the learning methods to simplify



26.2 Approximations of Bayesian Learning

Maximum A Posteriori (MAP) Approximation

▷ **Goal:** Get rid of summation over the space of all hypotheses in predictions.

▷ **Idea:** Make predictions wrt. the “most probable hypothesis”,

▷ **Definition 26.2.1** For Maximum a posteriori (MAP learning) choose h_{MAP} maximizing $P(h_i | \mathbf{d})$.

I.e., maximize $P(\mathbf{d} | h_i) \cdot P(h_i)$ or (even better) $\log_2(P(\mathbf{d} | h_i)) + \log_2(P(h_i))$.

▷ Predictions made according to an MAP hypothesis h_{MAP} are approximately Bayesian to the extent that $\mathbf{P}(X | \mathbf{d}) \approx \mathbf{P}(X | h_{\text{MAP}})$.

▷ In our candy example, $h_{\text{MAP}} = h_5$ after three lime candies in a row

▷ a MAP learner then predicts that the fourth candy is lime with probability 1.0

▷ compare with Bayesian prediction of 0.8 (see prediction curves above)

▷ As more data arrive, the MAP and Bayesian predictions become closer, because the competitors to the MAP hypothesis become less and less probable.

▷ For deterministic hypotheses, $P(\mathbf{d} | h_i)$ is 1 if consistent, 0 otherwise \rightsquigarrow MAP = simplest consistent hypothesis (cf. science)

▷ **Remark:** finding MAP hypotheses is often much easier than Bayesian learning, because it requires solving an optimization problem instead of a large summation (or integration) problem.



Digression From MAP-learning to MDL-learning

▷ Log terms in MAP learning: $\log_2(P(\mathbf{d} | h_i)) + \log_2(P(h_i))$ can be viewed as

– bits to encode data given hypothesis + additional bits to encode hypothesis

Indeed, if hypothesis hypothesis predicts the data exactly – e.g. h_5 in candy example – then $\log_2(1) = 0 \rightsquigarrow$ preferred hypothesis.

▷ This is more directly modeled by the following approximation to full Bayesian learning:

- ▷ **Definition 26.2.2** In minimum description length learning (**MDL learning**) the MDL hypothesis h_{MDL} minimizes the **information entropy** of the hypothesis likelihood.



©: Michael Kohlhase

845



Maximum Likelihood (ML) approximation

- ▷ **Observation:** For large data sets, the prior becomes irrelevant. (**we might not trust it anyways**)
- ▷ **Idea:** Use this to simplify learning
- ▷ **Definition 26.2.3 Maximum likelihood learning (ML learning):** choose h_{ML} maximizing $P(\mathbf{d} | h_i)$ (**simply get the best fit to the data**)
- ▷ **Remark:** ML learning $\hat{=} \text{MAP learning}$ for uniform prior (**reasonable if all hypotheses are of the same complexity**)
- ▷ ML is the “standard” (non-Bayesian) statistical learning method.



©: Michael Kohlhase

846



26.3 Parameter Learning for Bayesian Networks

ML Parameter Learning in Bayes Nets

- ▷ **Example 26.3.1** Bag from a new manufacturer; fraction θ of cherry candies?



- ▷ Any θ is possible: continuum of hypotheses h_θ
 θ is a **parameter** for this simple (**binomial**) family of models

- ▷ Suppose we unwrap N candies, c cherries and $\ell = N - c$ limes

- ▷ These are IID (independent identically distributed) observations, so the likelihood is

$$P(\mathbf{d} | h_\theta) = \prod_{j=1}^N P(d_j | h_\theta) = \theta^c \cdot (1 - \theta)^\ell$$

- ▷ **Trick:** When optimizing a product, optimize the logarithm instead! (**\log_2 is monotonic and turns products into sums**)

- ▷ **Definition 26.3.2** The **log-likelihood** is just the **binary logarithm** of the likelihood.

$$L(\mathbf{d} | h) := \log_2(P(\mathbf{d} | h))$$



ML Parameter Learning in Bayes Nets

▷ Compute the log-likelihood as

$$\begin{aligned} L(\mathbf{d} \mid h_\theta) &= \log_2(P(\mathbf{d} \mid h_\theta)) \\ &= \sum_{j=1}^N \log_2(P(d_j \mid h_\theta)) \\ &= c \log_2(\theta) + \ell \log_2(1 - \theta) \end{aligned}$$

▷ Maximize this w.r.t. θ

$$\frac{\partial}{\partial \theta}(L(\mathbf{d} \mid h_\theta)) = \frac{c}{\theta} - \frac{\ell}{1 - \theta} = 0$$

$$\leadsto \theta = \frac{c}{c+\ell} = \frac{c}{N}$$

▷ In English: $h_{ML}\theta$ asserts that the actual proportion of cherries in the bag is equal to the observed proportion in the candies unwrapped so far!

▷ Seems sensible, but causes problems with 0 counts!

▷ Question: Haven't we done a lot of work to obtain the obvious?

▷ Answer: So far yes, but this is a general methods of broad applicability



ML Parameter Learning for Multiple Parameters in Bayesian Networks

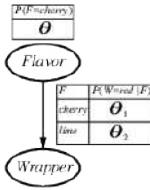
▷ Cooking Recipie:

1. Write down an expression for the likelihood of the data as a function of the parameter(s).
2. Write down the derivative of the log likelihood with respect to each parameter.
3. Find the parameter values such that the derivatives are zero



Multiple Parameters Example

▷ Example 26.3.3 Red/green wrapper depends probabilistically on flavour:



▷ Likelihood for, e.g., cherry candy in green wrapper:

$$\begin{aligned} P(F = \text{cherry}, W = \text{green} \mid h_{\theta, \theta_1, \theta_2}) \\ = P(F = \text{cherry} \mid h_{\theta, \theta_1, \theta_2}) \cdot P(W = \text{green} \mid F = \text{cherry}, h_{\theta, \theta_1, \theta_2}) \\ = \theta \cdot (1 - \theta_1) \end{aligned}$$

▷ N candies, r_c red-wrapped cherry candies, etc.:

$$P(\mathbf{d} \mid h_{\theta, \theta_1, \theta_2}) = \theta^c \cdot (1 - \theta)^{\ell} \cdot \theta_1^{r_c} \cdot (1 - \theta_1)^{g_c} \cdot \theta_2^{r_\ell} \cdot (1 - \theta_2)^{g_\ell}$$



Multiple Parameters Example (contd.)

▷ minimize the log-likelihood instead:

$$\begin{aligned} L &= c \log_2(\theta) + \ell \log_2(1 - \theta) \\ &+ r_c \log_2(\theta_1) + g_c \log_2(1 - \theta_1) \\ &+ r_\ell \log_2(\theta_2) + g_\ell \log_2(1 - \theta_2) \end{aligned}$$

▷ Derivatives of L contain only the relevant parameter:

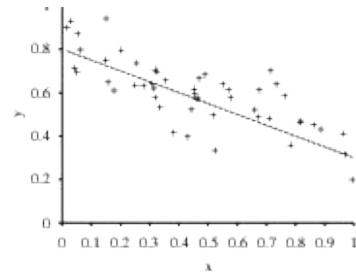
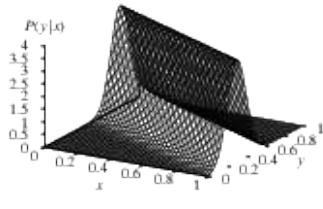
$$\begin{aligned} \frac{\partial L}{\partial \theta} &= \frac{c}{\theta} - \frac{\ell}{1 - \theta} = 0 \quad \leadsto \quad \theta = \frac{c}{c + \ell} \\ \frac{\partial L}{\partial \theta_1} &= \frac{r_c}{\theta_1} - \frac{g_c}{1 - \theta_1} = 0 \quad \leadsto \quad \theta_1 = \frac{r_c}{r_c + g_c} \\ \frac{\partial L}{\partial \theta_2} &= \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1 - \theta_2} = 0 \quad \leadsto \quad \theta_2 = \frac{r_\ell}{r_\ell + g_\ell} \end{aligned}$$

▷ **Upshot:** With complete data, *parameters can be learned separately* in Bayesian networks.

▷ **Remaining Problem:** have to be careful with zero values (division by zero)



Example: Linear Gaussian Model



- ▷ Maximizing $P(y | x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-(\theta_1 x + \theta_2))^2}{2\sigma^2}}$ w.r.t. θ_1, θ_2
- $=$ minimizing $E = \sum_{j=1}^N (y_j - (\theta_1 x_j + \theta_2))^2$
- ▷ That is, minimizing the sum of squared errors gives the ML solution for a linear fit *assuming Gaussian noise of fixed variance*



26.4 Naive Bayes Models

Naive Bayes Models for Learning

- ▷ **native Bayes models** are probably the most commonly used Bayesian network model in machine learning.
 - ▷ the “class” variable C (which is to be predicted) is the root
 - ▷ the “attribute” variables X_i are the leaves.



Before we look at the learning aspects, let us recapitulate what we have learned about naive Bayesian models above.

Naive Bayes Models

- ▷ **Definition 26.4.1** 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 classifiers**. (also called **idiot Bayes model** by Bayesian fundamentalists)
- ▷ **Observation 26.4.2** In a naive Bayes model, the full joint probability distribution can be written as

$$P(\text{cause} | \text{effect}_1, \dots, \text{effect}_n) = P(\text{cause}) \cdot \prod_i P(\text{effect}_i | \text{cause})$$

- ▷ This kind of model is called “naive” or “idiot” since it is often used as a simplifying model if the effects are not conditionally independent after all.
- ▷ In practice, naive Bayes systems can work surprisingly well, even when the conditional independence assumption is not true.

- ▷ **Example 26.4.3** The dentistry example is a (true) naive Bayes model.



Naive Bayes Models for Learning (continued)

- ▷ native Bayes models are probably the most commonly used Bayesian network model in machine learning.
 - ▷ the “class” variable C (which is to be predicted) is the root
 - ▷ the “attribute” variables X_i are the leaves.
- ▷ The candy with wrappers example is native Bayes model. (only one effect)
- ▷ Assuming Boolean variables the parameters are $\theta = P(c = \text{T})$, $\theta_{i1} = P(X_i = \text{T} | C = \text{T})$, and $\theta_{i2} = P(X_i = \text{T} | C = \text{F})$
- ▷ the maximum likelihood parameters can be found exactly like above
- ▷ once trained, use this model to classify new examples, where C is unobserved
- ▷ With observed values x_1, \dots, x_n , the probability of each class is given by

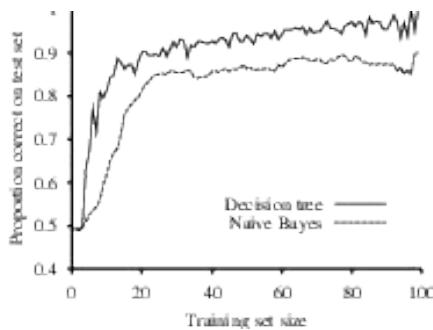
$$\mathbf{P}(C | x_1, \dots, x_n) = \alpha \cdot \mathbf{P}(C) \cdot \prod_i \mathbf{P}(x_i | C)$$

- ▷ A deterministic prediction can be obtained by choosing the most likely class.



Naive Bayes Models for Learning (Properties)

- ▷ Naive Bayes learning turns out to do surprisingly well in a wide range of applications
- ▷ **Example 26.4.4** Learning curve for naive Bayes learning on the restaurant example



Naive Bayes learning scales well: with n Boolean attributes, there are just $2n + 1$ parameters, and no search is required to find h_{ML} .

- ▷ Naive Bayes learning systems have no difficulty with noisy or missing data and can give probabilistic predictions when appropriate.



Summary

- ▷ Full Bayesian learning gives best possible predictions but is intractable
- ▷ MAP learning balances complexity with accuracy on training data
- ▷ Maximum likelihood assumes uniform prior, OK for large data sets
 1. Choose a parameterized family of models to describe the data
requires substantial insight and sometimes new models
 2. Write down the likelihood of the data as a function of the parameters
may require summing over hidden variables, i.e., inference
 3. Write down the derivative of the log likelihood w.r.t. each parameter
 4. Find the parameter values such that the derivatives are zero
may be hard/impossible; modern optimization techniques help
- ▷ naive Bayes models as a fall-back solution for machine learning
 - ▷ conditional independence of all attributes as simplifying assumption



Chapter 27

Knowledge in Learning

27.1 Logical Formulations of Learning

Knowledge in Learning: Motivation

- ▷ Recap: learning from examples (last chapter)
- ▷ Idea: Construct a function that has the input/output behavior observed in the data
- ▷ Method: Search for suitable functions in the hypothesis space (e.g. decision trees)
- ▷ Observation 27.1.1 *Every learning task begins from zero* (except for the choice of hypothesis space)
- ▷ Problem: We have to forget everything before we can learn something new.
- ▷ Idea: Utilize *prior knowledge about the world* (represented in logic)



©: Michael Kohlhase

858



A logical Formulation of Learning

- ▷ Recall: Examples are composed of descriptions and classifications
- ▷ Idea: Represent all three as logical formulae.
- ▷ Example 27.1.2 For *attribute-based representations*, we can use PL¹: we use predicate constants for Boolean attributes and classification and function constants for the other attributes.
- ▷ Definition 27.1.3 *Logic-based inductive learning* tries to learn an hypothesis h that explains the classifications of the examples given their descriptions, i.e. $h, \mathcal{D} \models \mathcal{C}$, where
 - ▷ \mathcal{D} is the conjunction of the descriptions of the examples, and

▷ \mathcal{C} the conjunction of their classifications.

We use Occam's razor to avoid $h = \mathcal{C}$. (would be too easy otherwise)



©: Michael Kohlhase

859



A logical Formulation of Learning (Restaurant Example)

▷ **Example 27.1.4 (Restaurant Example again)** Descriptions are conjunctions of literals built up from

- ▷ predicates Alt, Bar, Fri, Hun, Rain, and Res
- ▷ equations about the functions Pat, Price, Type, and Est.

For instance the first example X_1 from Example 25.3.2, can be described as

$$\text{Alt}(X_1) \wedge \neg \text{Bar}(X_1) \wedge \text{Fri}(X_1) \wedge \text{Hun}(X_1) \wedge \dots$$

The classification is given by the goal predicate WillWait, in this case $\text{WillWait}(X_1)$ or $\neg \text{WillWait}(X_1)$.

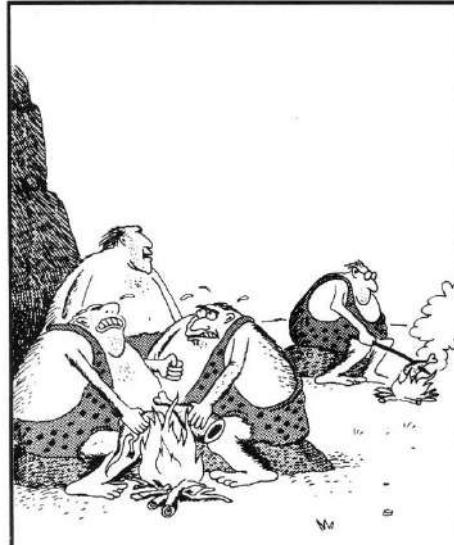


©: Michael Kohlhase

860



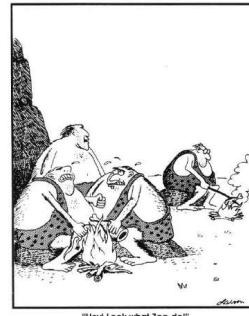
Cumulative Development



"Hey! Look what Zog do!"

Example 27.1.5 where background knowledge is vital

- ▷ Caveman Zog and the fish on a stick
 - ▷ Generalizing from one Brazilian
 - ▷ general rules about effectiveness of antibiotics
- ▷ to use background knowledge, need a method to obtain knowledge (use learning)



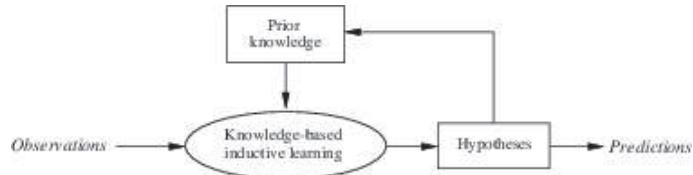
©: Michael Kohlhase

861



Cumulative Development (cont.)

- ▷ **Question:** How to use knowledge to learn more efficiently?
- ▷ **Answer:** Cumulative development



©: Michael Kohlhase

862



Adding Background Knowledge to Learning

- ▷ Explanation-based learning (EBL)
- ▷ Relevance-based learning (RBL)
- ▷ Knowledge-based inductive learning (KBIL)



©: Michael Kohlhase

863



Explanation-based Learning

- ▷ Use explanation of success to infer a general rule

- ▷ General rule follows logically from the background knowledge

$$\begin{array}{l} \textit{Hypothesis} \wedge \textit{Descriptions} \models \textit{Classifications} \\ \textit{Background} \models \textit{Hypothesis} \end{array}$$

- ▷ Does not learn anything factually new
 - ▷ Converting first-principles theories into useful, special purpose knowledge



Relevance-based Learning

- ▷ The prior knowledge concerns the relevance of a set of features to the goal predicate
 - ▷ **Example 27.1.6** In a given country most people speak the same language, but do not have the same name
 - ▷
- $$\begin{array}{l} \textit{Hypothesis} \wedge \textit{Descriptions} \models \textit{Classifications} \\ \textit{Background} \wedge \textit{Descriptions} \wedge \textit{Classifications} \models \textit{Hypothesis} \end{array}$$
- ▷ Deductive learning: Makes use of the observations, but does not produce hypothesis beyond the background knowledge and the observations



Knowledge-based Inductive Learning

- ▷ The background knowledge and the new hypothesis combine to explain the examples
- ▷ **Example 27.1.7** Inferring disease D from the symptoms is not enough to explain the prescription of medicine M .
- ▷ A rule that M is effective against D is needed

$$\textit{Background} \wedge \textit{Hypothesis} \wedge \textit{Descriptions} \models \textit{Classifications}$$



Inductive Logic Programming

- ▷ Main field of study for KBIL algorithms
- ▷ Prior knowledge plays two key roles
 - ▷ The effective hypothesis space is reduced to include only those theories that

- are consistent with what is already known
- ▷ Prior knowledge can be used to reduce the size of the hypothesis explaining the observations
 - ▷ Smaller hypotheses are easier to find
- ▷ ILP systems can formulate hypotheses in first-order logic
 - ▷ Can learn in environments not understood by simpler systems



27.2 Explanation-Based Learning

Explanation-Based Learning

- ▷ Extracting general rules from individual observations
- ▷ **Example 27.2.1** differentiating and simplifying algebraic expressions
 - ▷ Differentiate X^2 with respect to X to get $2X$
 - ▷ Logical reasoning system ask(Deriv(X^2 , X) = d , KB) with solution $d = 2X$
 - ▷ Solving this for the first time using standard rules of differentiation gives $1 \times (2 \times (X^{2-1}))$
 - ▷ Takes a first-time program 136 proof steps with 99 dead end branches
- ▷ Memoization
 - ▷ Speed up by saving the results of computation
 - ▷ Create a database of input/output pairs



Creating general rules

- ▷ Memoization in explanation-based learning
 - ▷ Create general rules that cover an entire class of cases
- ▷ **Example 27.2.2** extract the general rule ArithVar(u) \Rightarrow Deriv(u^2 , u) = $2u$
- ▷ Once something is understood, it can be generalized and reused in other circumstances
 - ▷ “Civilization advances by extending the number of important operations that we can do without thinking about them”
- ▷ Explaining why something is a good idea is much easier than coming up with the idea in the first place
 - ▷ Watch caveman Zog roast his lizard vs. thinking about putting the fish on a stick

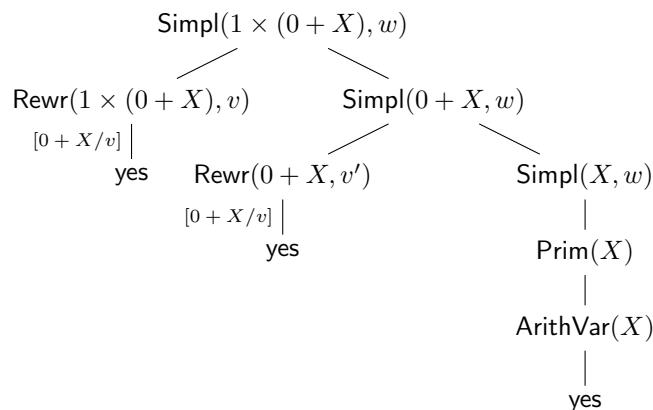


Extracting rules from examples

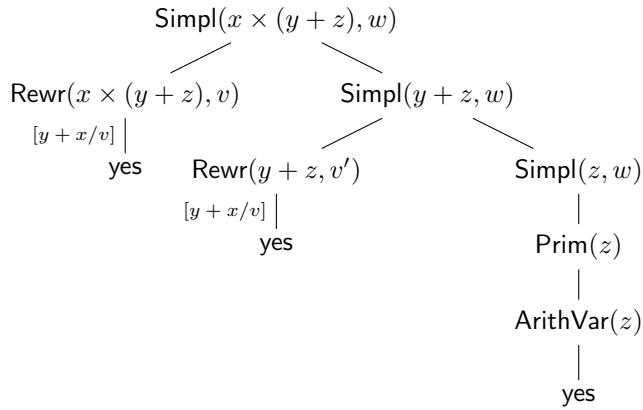
- ▷ Basic idea behind EBL
 - ▷ Construct an explanation of the observation using prior knowledge
 - ▷ Establish a definition of the class of cases for which the same explanation can be used
- ▷ **Example 27.2.3** simplifying $1 \times (0 + X)$ using a knowledge base with the following rules
 - ▷ $\text{Rewr}(u, v) \wedge \text{Simpl}(v, w) \Rightarrow \text{Simpl}(u, w)$
 - ▷ $\text{Prim}(u) \Rightarrow \text{Simpl}(u, u)$
 - ▷ $\text{ArithVar}(u) \Rightarrow \text{Prim}(u)$
 - ▷ $\text{Num}(u) \Rightarrow \text{Prim}(u)$
 - ▷ $\text{Rewr}(1 \times u, u)$
 - ▷ $\text{Rewr}(0 + u, u)$
 - ▷ ...



Proof Tree for the Original Problem



Generalized Proof Tree



Generalizing proofs

- ▷ The variabilized proof proceeds using exactly the same rule applications
- ▷ May lead to variable instantiation
- ▷ **Example 27.2.4** Take the leaves of the generalized proof tree to get the general rule

$$\text{Rewr}(1 \times (0 + z), 0 + z) \wedge \text{Rewr}(0 + z, z) \wedge \text{ArithVar}(z) \Rightarrow \text{Simpl}(1 \times (0 + z), z)$$

- ▷ The first two conditions are true independently of z , so this becomes

$$\text{ArithVar}(z) \Rightarrow \text{Simpl}(1 \times (0 + z), z)$$

- ▷ Recap
- ▷ Use background knowledge to construct a proof for the example
- ▷ In parallel, construct a generalized proof tree
- ▷ New rule is the conjunction of the leaves of the proof tree and the variabilized goal
- ▷ Drop conditions that are true regardless of the variables in the goal

Improving efficiency

- ▷ **Idea:** Pruning the proof tree to get more general rules
- ▷ **Example 27.2.5**

$$\text{Prim}(z) \Rightarrow \text{Simpl}(1 \times (0 + z), z)$$

$$\text{Simpl}(y + z, w) \Rightarrow \text{Simpl}(1 \times (y + z), w)$$

- ▷ **Problem:** Which rules to choose?
 - ▷ Adding large numbers of rules to the knowledge base slows down the reasoning process (increases the branching factor of the search space)
 - ▷ To compensate, the derived rules must offer significant speed increases
 - ▷ Derived rules should be as general as possible to apply to the largest possible set of cases



Improving efficiency

- ▷ Operability of subgoals in the rule
 - ▷ A subgoal must be “easy” to solve
 - ▷ $\text{Prim}(z)$ is easy to solve, but $\text{Simpl}(y + z, w)$ leads to an arbitrary amount of inference
 - ▷ Keep operational subgoals and prune the rest of the tree
- ▷ Trade-off between operability and generality
 - ▷ More specific subgoals are easier to solve but cover fewer cases
 - ▷ How many steps are still called operational?
 - ▷ Cost of a subgoal depends on the rules in the knowledge base

Maximizing the efficiency of an initial knowledge base is a complex optimization problem



Improving efficiency

- ▷ Empirical analysis of efficiency
 - ▷ Average-case complexity on a population of problems that needs to be solved
- ▷ By generalizing from past example problems, EBL makes the knowledge base more efficient for the kind of problems that it is reasonable to expect
 - ▷ Works if the distribution of past problems is roughly the same as for future problems
 - ▷ Can lead to great improvements
 - ▷ Swedish to English translator was made 1200 times faster by using EBL [SR91]



27.3 Relevance-Based Learning

Recap: Relevance-based Learning

- ▷ The prior knowledge concerns the relevance of a set of features to the goal predicate
- ▷ **Example 27.3.1** In a given country most people speak the same language, but do not have the same name

$$\begin{array}{ll} \textit{Hypothesis} \wedge \textit{Descriptions} & \models \textit{Classifications} \\ \textit{Background} \wedge \textit{Descriptions} \wedge \textit{Classifications} & \models \textit{Hypothesis} \end{array}$$

- ▷ Deductive learning: Makes use of the observations, but does not produce hypothesis beyond the background knowledge and the observations



©: Michael Kohlhase

877



Relevance-based Learning: Determinations

- ▷ **Example 27.3.2 (Background knowledge in Brazil)**

$$\forall x, y, n, l. \text{Nationality}(x, n) \wedge \text{Nationality}(y, n) \wedge \text{Language}(x, l) \Rightarrow \text{Language}(y, l)$$

So

$$\text{Nationality}(\text{Fernando}, \text{Brazil}) \wedge \text{Language}(\text{Fernando}, \text{Portuguese})$$

entails

$$\forall x. \text{Nationality}(x, \text{Brazil}) \Rightarrow \text{Language}(x, \text{Portuguese})$$

Special syntax: $\text{Nationality}(x, n) \succ \text{Language}(x, l)$

- ▷ **Definition 27.3.3** If $\forall x, y. P(x) \wedge P(y) \wedge Q(x) \Rightarrow Q(y)$, then we say that P determines Q and write $P \succ Q$; we call this formula a **determination**.



©: Michael Kohlhase

878



Determining the Hypothesis Space

- ▷ Determinations limit the hypothesis space
- ▷ Only consider the important features (i.e. not day of the week, hair style of David Beckham)
- ▷ Determinations specify a sufficient basis vocabulary from which to construct hypotheses
- ▷ Reduction of the hypothesis space makes it easier to learn the target predicate
 - ▷ Learning boolean functions of n variables in CNF: Size of the hypothesis space $\#(H) = \mathcal{O}(2^{2^n})$
 - ▷ For boolean functions $\log_2(\#(H))$ examples are needed in a $\#(H)$ size hypothesis space: Without restrictions, this is $\mathcal{O}(2^n)$ examples

- ▷ If the determination contains d predicates on the left, only $\mathcal{O}(2^d)$ examples are needed
- ▷ Reduction of size by $\mathcal{O}(2^{n-d})$.



Learning Relevance Information

- ▷ Prior knowledge also needs to be learned
- ▷ **Idea:** Learning algorithm for **determinations**:
 - ▷ Find the simplest **determination** consistent with the observations
 - ▷ A determination $P \succ Q$ says that if examples match P they must also match Q
 - ▷ A determination is consistent with a set of examples if every pair that matches on the predicates on the left-hand side also matches on the target predicate



Learning relevance information

Example 27.3.4

Sample	Mass	Temp	Material	Size	Conductance
S1	12	26	Copper	3	0.59
S1	12	100	Copper	3	0.57
S2	24	26	Copper	6	0.59
S3	12	26	Lead	2	0.05
S3	12	100	Lead	2	0.04
S4	24	26	Lead	4	0.05

- ▷ Minimal consistent determination $Material \wedge Temperature \succ Conductance$
- ▷ Non-minimal consistent determination $Mass \wedge Size \wedge Temperature \succ Conductance$



Learning Relevance Information (Algorithm)

- ▷ **Definition 27.3.5 function** Minimimal–Consistent–Det(E, A) **returns** a determination
 - inputs:** E , a set of examples
 - A , a set of attributes, of size n
 - for** $i := 1, \dots, n$ **do**
 - for** each subset A_i of A of size i **do**
 - if** Consistent–Det?(A_i, E) **then return** A_i
 - end**
 - end**

```

function Consistent-Det?( $A, E$ ) returns a truth-value
  inputs:  $A$ , a set of attributes
             $E$ , a set of examples
  local variables:  $H$ , a hash table
  for each example  $e$  in  $E$  do
    if some  $h \in H$  has the same  $A$ -value as  $e$  but different class
    then return False
    store the class of  $e$  in  $H$ , indexed by the  $A$ -values of  $e$ 
  end
  return True

```



Complexity

- ▷ Time complexity depends on the size of the minimal consistent determination
 - ▷ In case of p attributes and a total of n attributes, the algorithm has to search all subsets of A of size p
 - ▷ There are $\mathcal{O}(n^p)$ of these, so the algorithm is exponential
 - ▷ The general problem is NP-complete
 - ▷ In most domains there is sufficient local structure to make p small



Deriving Hypotheses

- ▷ **Idea:** Use decision tree learning for computing hypotheses
- ▷ **Goal:** Minimize size of hypotheses
- ▷ **Idea:** Use relevance-based decision tree learning



Relevance-based Decision Tree Learning

- ▷ **Idea:** Use determinations to tune attribute selection in decision tree learning (DTL).
- ▷ **Definition 27.3.6**

```

function RBDTL( $E, A, v$ ) returns a decision tree
return DTL( $E$ , Minimal-Consistent-Det( $E, A$ ),  $v$ )

function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return Majority(examples)

```

```

else
    best := Choose-Attribute(attributes,examples)
    tree := a new decision tree with root test best
    for each value  $v_i$  of best do
        examples $_i$  := {elements of examples with best =  $v_i$ }
        subtree := DTL(examples $_i$ ,attributes - best, Mode(examples))
        add a branch to tree with label  $v_i$  and subtree subtree
    return tree

```

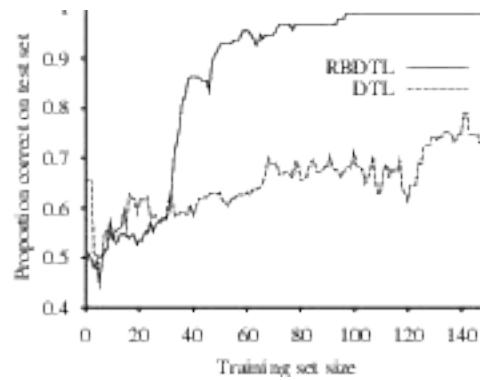


Exploiting Knowledge

- ▷ RBDTL simultaneously learns and uses relevance information to minimize its hypothesis space
- ▷ Declarative bias
 - ▷ How can prior knowledge be used to identify the appropriate hypothesis space to search for the correct target definition?
- ▷ Unanswered questions
 - ▷ How to handle noise?
 - ▷ How to use other kinds of prior knowledge besides determinations?
 - ▷ How can the algorithms be generalized to cover any first- order theory?



RBDTL vs. DTL



27.4 Inductive Logic Programming

Inductive Logic Programming

- ▷ Combines inductive methods with the power of first-order representations
- ▷ Offers a rigorous approach to the general KBIL problem
- ▷ Offers complete algorithms for inducing general, first-order theories from examples



27.4.1 An Example

ILP: An example

- ▷ General knowledge-based induction problem

Background \wedge Descriptions \wedge Classifications \models Hypothesis

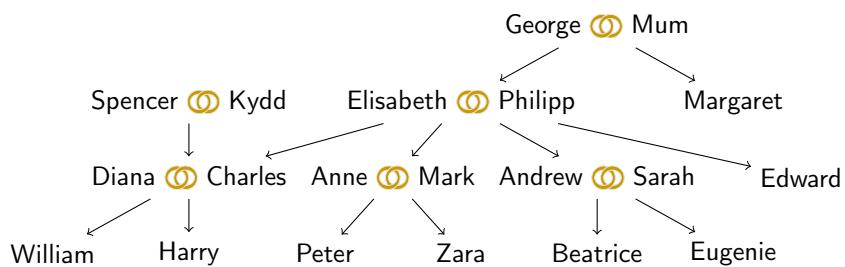
- ▷ Example 27.4.1 (Learning family relations from examples)

- ▷ Observations are an extended family tree
 - ▷ mother, father and married relations
 - ▷ male and female properties
- ▷ Target predicates: grandparent, BrotherInLaw, Ancestor



British Royalty Family Tree (not quite up to date)

- ▷ The facts about kinship and relations can be visualized as a family tree



Example

- ▷ Descriptions include facts like
 - ▷ $\text{father}(\text{Philip}, \text{Charles})$
 - ▷ $\text{mother}(\text{Mum}, \text{Margaret})$
 - ▷ $\text{married}(\text{Diana}, \text{Charles})$

- ▷ $\sigma Philip$
- ▷ $\varphi Beatrice$
- ▷ Sentences in Classifications depend on the target concept being learned (in the example: 12 positive, 388 negative)
 - ▷ $\text{grandparent}(\text{Mum}, \text{Charles})$
 - ▷ $\neg \text{grandparent}(\text{Mum}, \text{Harry})$
- ▷ Goal: find a set of sentences for Hypothesis such that the entailment constraint is satisfied
 - ▷ Without background knowledge, define "grandparent" in terms of "mother" and "father".
$$\text{gp}(x, y) \Leftrightarrow (\exists z. \text{m}(x, z) \wedge \text{m}(z, y)) \vee (\exists z. \text{m}(x, z) \wedge \text{f}(z, y)) \vee \dots \vee (\exists z. \text{f}(x, z) \wedge \text{f}(z, y))$$



©: Michael Kohlhase

891



Why Attribute-based Learning Fails

- ▷ Decision-Tree-Learning will get nowhere
 - ▷ To express Grandparent as a (Boolean) attribute, pairs of people need to be objects $\text{Grandparent}(\langle \text{Mum}, \text{Charles} \rangle)$
 - ▷ But then the example descriptions can not be represented

$$\text{FirstElementIsMotherOfElizabeth}(\langle \text{Mum}, \text{Charles} \rangle)$$
 - ▷ A large disjunction of specific cases without any hope of generalization to new examples
- ▷ Attribute-based learning algorithms are incapable of learning relational predicates



©: Michael Kohlhase

892



Background knowledge

- ▷ A little bit of background knowledge helps a lot
 - ▷ Background knowledge contains $\text{parent}(x, y) \Leftrightarrow \text{mother}(x, y) \vee \text{father}(x, y)$
 - ▷ Grandparent is now reduced to $\text{grandparent}(x, y) \Leftrightarrow (\exists z. \text{parent}(x, z) \wedge \text{parent}(z, y))$
- ▷ Constructive induction algorithm
 - ▷ Create new predicates to facilitate the expression of explanatory hypotheses
 - ▷ Example: introduce a predicate Parent to simplify the definitions of the target predicates



©: Michael Kohlhase

893



27.4.2 Top-Down Inductive Learning: FOIL

Top-Down Inductive Learning

- ▷ Top-down learning method
 - ▷ Decision-tree learning: start from the observations and work backwards
 - ▷ Decision tree is gradually grown until it is consistent with the observations
 - ▷ Top-down learning: start from a general rule and specialize it



©: Michael Kohlhase

894



Top-Down Inductive Learning: FOIL

- ▷ Split positive and negative examples
 - ▷ Positive: $\langle George, Anne \rangle, \langle Philip, Peter \rangle, \langle Spencer, Harry \rangle$
 - ▷ Negative: $\langle George, Elizabeth \rangle, \langle Harry, Zara \rangle, \langle Charles, Philip \rangle$
- ▷ Construct a set of Horn clauses with $Grandfather(x,y)$ as the head with the positive examples instances of the Grandfather relationship
 - ▷ Start with a clause with an empty body $\Rightarrow Grandfather(x,y)$
 - ▷ All examples are now classified as positive, so specialize to rule out the negative examples: Here are 3 potential additions:
 1. $father(x,y) \Rightarrow Grandfather(x,y)$
 2. $parent(x,z) \Rightarrow Grandfather(x,y)$
 3. $father(x,z) \wedge parent(z,y) \Rightarrow Grandfather(x,y)$
 - ▷ The first one incorrectly classifies the 12 positive examples
 - ▷ The second one is incorrect on a larger part of the negative examples
 - ▷ Prefer the third clause and specialize $father(x,z) \wedge parent(z,y) \Rightarrow Grandfather(x,y)$



©: Michael Kohlhase

895



FOIL

```

function Foil(examples,target) returns a set of Horn clauses
  inputs: examples, set of examples
  target, a literal for the goal predicate
  local variables: clauses, set of clauses, initially empty
  while examples contains positive examples do
    clause := New-Clause(examples,target)
    remove examples covered by clause from examples
    add clause to clauses
  return clauses

```



©: Michael Kohlhase

896



FOIL

```

function New–Clause(examples,target) returns a Horn clause
  local variables: clause, a clause with target as head and an empty body
    l, a literal to be added to the clause
    extendedExamples, a set of examples with values for new variables
    extendedExamples := examples
  while extendedExamples contains negative examples do
    l := Choose–Literal(New–Literals(clause),extendedExamples)
    append l to the body of clause
    extendedExamples := map Extend–Example over extendedExamples
  return clause

function Extend–Example(example,literal) returns a new example
  if example satisfies literal
  then return the set of examples created by extending example with each
    possible constant value for each new variable in literal
  else return the empty set

function New–Literals(clause) returns a set of possibly “useful” literals
function Choose–Literal(literals) returns the “best” literal from literals

```



FOIL: Choosing Literals

- ▷ New-Literals: Takes a clause and constructs all possibly “useful” literals
- ▷ **Example 27.4.2** $\text{father}(x, z) \Rightarrow \text{Grandfather}(x, y)$
 - ▷ Add literals using predicates
 - ▷ Negated or unnegated
 - ▷ Use any existing predicate (including the goal)
 - ▷ Arguments must be variables
 - ▷ Each literal must include at least one variable from an earlier literal or from the head of the clause
 - ▷ Valid: $\text{Mother}(z, u)$, $\text{Married}(z, z)$, $\text{Grandfather}(v, x)$
 - ▷ Invalid: $\text{Married}(u, v)$
 - ▷ Equality and inequality literals
 - ▷ E.g. $z \neq x$, empty list
 - ▷ Arithmetic comparisons
 - ▷ E.g. $x > y$, threshold values



FOIL: Choosing Literals

- ▷ The way New-Literal changes the clauses leads to a very large branching factor
- ▷ Improve performance by using type information

- ▷ E.g., $\text{parent}(x, n)$ where x is a person and n is a number
- ▷ Choose-Literal uses a heuristic similar to information gain
- ▷ Ockham's razor to eliminate hypotheses
 - ▷ If the clause becomes longer than the total length of the positive examples that the clause explains, this clause is not a valid hypothesis
- ▷ Most impressive demonstration
 - ▷ Learn the correct definition of list-processing functions in Prolog from a small set of examples, using previously learned functions as background knowledge



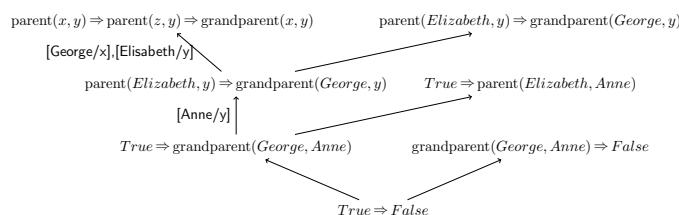
27.4.3 Inverse Resolution

Inverse Resolution

- ▷ **Inverse resolution in a nutshell:**
 - ▷ Classifications follows from $\text{Background} \wedge \text{Hypothesis} \wedge \text{Descriptions}$.
 - ▷ This can be proven by resolution
 - ▷ Run the proof backwards to find Hypothesis
- ▷ **Problem:** How to run the proof backwards?
- ▷ **Recap:** In ordinary resolution we take two clauses $C_1 = L \vee R_1$ and $C_2 = \neg L \vee R_2$ and resolve them to produce the resolvent $C = R_1 \vee R_2$.
- ▷ **Idea:** two possible variants of inverse resolution
 - ▷ Take resolvent C and produce two clauses C_1 and C_2
 - ▷ Take C and C_1 and produce C_2



Generating Inverse Proofs (Example)



Generating Inverse Proofs

- ▷ Inverse resolution is a search
 - ▷ For any C and C_1 there can be several or even an infinite number of clauses C_2 .
 - ▷ Instead of $\text{parent}(Elizabeth, y) \Rightarrow \text{grandparent}(George, y)$ there were numerous alternatives
 $\text{parent}(Elizabeth, Anne) \Rightarrow \text{grandparent}(George, Anne)$
 $\text{parent}(z, Anne) \Rightarrow \text{grandparent}(George, Anne)$
 $\text{parent}(z, y) \Rightarrow \text{grandparent}(George, y)$
- ▷ The clauses C_1 that participate in each step can be chosen from Background, Descriptions, Classifications or from hypothesized clauses already generated
- ▷ ILP needs restrictions to make the search manageable
 - ▷ Eliminate function symbols
 - ▷ Generate only the most specific hypotheses
 - ▷ Use Horn clauses
 - ▷ All hypothesized clauses must be consistent with each other
 - ▷ Each hypothesized clause must agree with the observations



©: Michael Kohlhase

902



New Predicates and New Knowledge

- ▷ An inverse resolution procedure is a complete algorithm for learning first-order theories
 - ▷ If some unknown Hypothesis generates a set of examples, then an inverse resolution procedure can generate Hypothesis from the examples
- ▷ Can inverse resolution infer the law of gravity from examples of falling bodies?
 - ▷ Yes, given suitable background mathematics
- ▷ Monkey and typewriter problem: How to overcome the large branching factor and the lack of structure in the search space?



©: Michael Kohlhase

903



New Predicates and New Knowledge

- ▷ Inverse resolution is capable of generating new predicates
 - ▷ Resolution of C_1 and C_2 into C eliminates a literal that C_1 and C_2 share
 - ▷ This literal might contain a predicate that does not appear in C
 - ▷ When working backwards, one possibility is to generate a new predicate from which to construct the missing literal



New Predicates and New Knowledge

$$\begin{array}{c} \textit{Father}(\textit{George}; y) \Rightarrow P(x, y) \quad P(\textit{George}; y) \Rightarrow \textit{Ancestor}(\textit{George}, y) \\ \textit{George}/\overline{x} \quad \diagup \\ \textit{Father}(\textit{George}; y) \Rightarrow \textit{Ancestor}(\textit{George}, y) \end{array}$$

- ▷ P can be used in later inverse resolution steps
- ▷ **Example 27.4.3** $\text{mother}(x, y) \Rightarrow P(x, y)$ or $\text{father}(x, y) \Rightarrow P(x, y)$ leading to the “Parent” relationship
- ▷ Inventing new predicates is important to reduce the size of the definition of the goal predicate
- ▷ Some of the deepest revolutions in science come from the invention of new predicates (e.g. Galileo’s invention of acceleration)



Applications of ILP

- ▷ ILP systems have outperformed knowledge-free methods in a number of domains
- ▷ Molecular biology: the GOLEM system has been able to generate high-quality predictions of protein structures and the therapeutic efficacy of various drugs
- ▷ GOLEM is a completely general-purpose program that is able to make use of background knowledge about any domain



Knowledge in Learning: Summary

- ▷ Cumulative learning: Improve learning ability as new knowledge is acquired
- ▷ Prior knowledge helps to eliminate hypothesis and fills in explanations, leading to shorter hypotheses
- ▷ Entailment constraints: Logical definition of different learning types
- ▷ Explanation-based learning (EBL): Explain the examples and generalize the explanation
- ▷ Relevance-base learning (RBL): Use prior knowledge in the form of determinations to identify the relevant attributes
- ▷ Knowledge-based inductive learning (KBIL): Finds inductive hypotheses that explain sets of observations
- ▷ Inductive logic programming (ILP):

- ▷ Perform KBIL using knowledge expressed in first-order logic
- ▷ Generates new predicates with which concise new theories can be expressed



Chapter 28

Reinforcement Learning

28.1 Reinforcement Learning: Introduction & Motivation

Unsupervised Learning

- ▷ **So far:** we have studied “learning from examples”. (functions, logical theories, probability models)
- ▷ **Now:** How can agents learn “what to do” in the absence of labeled examples of “what to do”. We call this problem **unsupervised learning**.
- ▷ **Example 28.1.1 (Playing Chess)** learn transition model for own moves and maybe predict opponent’s moves.
- ▷ **Problem:** The agent needs to have some feedback about what is good/bad
 - ↪ cannot decide “what to do” otherwise. (remember: external performance standard for learning agents)
- ▷ **Example 28.1.2** The ultimate feedback in chess is whether you win, lose, or draw.
- ▷ **Definition 28.1.3** We call a learning situation where there are no labeled examples **unsupervised learning** and the feedback involved a **reward** or **reinforcement**.
- ▷ **Example 28.1.4** In soccer, there are intermediate reinforcements in the shape of goals, penalties, . . .



©: Michael Kohlhase

908



Reinforcement Learning as Policy Learning

- ▷ **Definition 28.1.5 Reinforcement learning** is a type of **unsupervised learning** where an agent learn how to behave in a environment by performing actions and seeing the results.
- ▷ **Recap:** In Section 24.1 we introduced rewards as parts of MDPs (Markov decision processes) to define optimal policies.

- ▷ an optimal policy maximizes the expected total reward.
- ▷ **Idea:** The task of reinforcement learning is to use observed rewards to come up with an optimal policy.
- ▷ In MDPs, the agent has total knowledge about the environment **and the** reward function, in reinforcement learning we do not assume this. (↗ POMDPs+rewards)
- ▷ **Example 28.1.6** You know a game without knowing the rules, and at some time the opponent shouts **you lose!**



©: Michael Kohlhase

909



Scope and Forms of Reinforcement Learning

- ▷ **Reinforcement Learning solves all of AI:** an agent is placed in an environment and must learn to behave successfully therein.
- ▷ **KISS:** we will only look at simple environments and simple agent designs
 - ▷ A **utility-based agent** learns a utility function on states and uses it to select actions that maximize the expected outcome utility. (passive learning)
 - ▷ A **Q-learning agent** learns an action-utility function, or **Q-function**, giving the expected utility of taking a given action in a given state. (active learning)
 - ▷ A **reflex agent** learns a policy that maps directly from states to actions.



©: Michael Kohlhase

910



28.2 Passive Learning

Passive Learning

- ▷ **To keep things simple:** agent uses a state-based representation in a fully observable environment
 - ▷ In passive learning, the agent's policy π is fixed: in state s , it always executes the action $\pi(s)$.
 - ▷ Its goal is simply to learn how good the policy is—that is, to learn the utility function $U^\pi(s)$.
- ▷ The passive learning task is similar to the policy evaluation task, part of the policy iteration algorithm, but the agent does not know
 - ▷ the transition model $P(s' | s, a)$, which specifies the probability of reaching state s' from state s after doing action a ,
 - ▷ nor the reward function $R(s)$, which specifies the reward for each state.



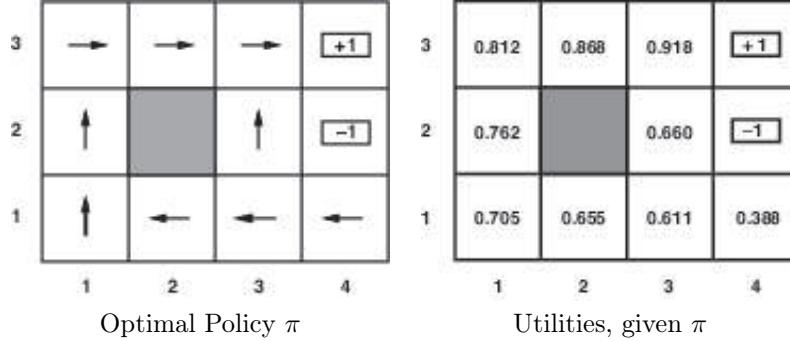
©: Michael Kohlhase

911



Passive Learning by Example

▷ **Example 28.2.1 (passive-learning-ex)** We use the 4×3 world introduced above



▷ The agent executes a set of trials in the environment using its policy π .

▷ In each trial, the agent starts in state $(1,1)$ and experiences a sequence of state transitions until it reaches one of the terminal states, $(4,2)$ or $(4,3)$.

▷ Its percepts supply both the current state and the reward received in that state.



Passive Learning by Example

▷ **Example 28.2.2** Typical trials might look like this:

1. $(1, 1) \xrightarrow{-0.4} (1, 2) \xrightarrow{-0.4} (1, 3) \xrightarrow{-0.4} (1, 2) \xrightarrow{-0.4} (1, 3) \xrightarrow{-0.4} (2, 3) \xrightarrow{-0.4}$
 $\sim (3, 3) \xrightarrow{-0.4} (4, 3) \xrightarrow{+1}$
2. $(1, 1) \xrightarrow{-0.4} (1, 2) \xrightarrow{-0.4} (1, 3) \xrightarrow{-0.4} (2, 3) \xrightarrow{-0.4} (3, 3) \xrightarrow{-0.4} (3, 2) \xrightarrow{-0.4}$
 $\sim (3, 3) \xrightarrow{-0.4} (4, 3) \xrightarrow{+1}$
3. $(1, 1) \xrightarrow{-0.4} (2, 1) \xrightarrow{-0.4} (3, 1) \xrightarrow{-0.4} (3, 2) \xrightarrow{-0.4} (4, 2) \xrightarrow{-1}$.

▷ **Definition 28.2.3** The **utility** is defined to be the expected sum of (discounted) rewards obtained if policy π is followed.

$$U^\pi(s) := E \left[\sum_{t=0}^{\infty} \gamma^t R(S_t) \right]$$

where $R(s)$ is the reward for a state, S_t (a random variable) is the state reached at time t when executing policy π , and $S_0 = s$. (for 4×3 we take the discount factor $\gamma = 1$)



Direct Utility Estimation

- ▷ A simple method for direct utility estimation was invented in the late 1950s in the area of adaptive control theory.
- ▷ **Definition 28.2.4** The utility of a state is the expected total reward from that state onward (called the expected **reward-to-go**)
- ▷ **Idea:** Each trial provides a sample of the reward-to-go for each state visited.
- ▷ **Example 28.2.5** The first trial in Example 28.2.2 provides a sample total reward of 0.72 for state (1,1), two samples of 0.76 and 0.84 for (1,2), two samples of 0.80 and 0.88 for (1,3), ...
- ▷ **Definition 28.2.6** The **direct utility estimation** algorithm cycles over trials, calculates the reward-to-go for each state, and updates the estimated utility for that state by keeping the running average for that for each state in a table.
- ▷ **Observation 28.2.7** *In the limit, the sample average will converge to the true expectation (utility) from Definition 28.2.3.*
- ▷ **Remark 28.2.8** Direct utility estimation is just supervised learning, where each example has the state as input and the observed reward-to-go as output.
- ▷ We have reduced reinforcement learning to an inductive learning problem.



Adaptive Dynamic Programming

- ▷ **Problem:** The utilities of states are not independent in direct utility estimation!
- ▷ The utility of each state equals its own reward plus the expected utility of its successor states.
- ▷ **So:** the utility values obey the **Bellman equation** for a fixed policy.

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^\pi(s')$$

- ▷ **Observation 28.2.9** *By ignoring the connections between states, direct utility estimation misses opportunities for learning.*

- ▷ **Example 28.2.10** Recall trial 2 in the example above; state (3,3) is new.

$$\begin{aligned} 2(1,1) &\xrightarrow{-0.4} (1,2) \xrightarrow{-0.4} (1,3) \xrightarrow{-0.4} (2,3) \xrightarrow{-0.4} (3,3) \xrightarrow{-0.4} (3,2) \xrightarrow{-0.4} \\ &\sim (3,3) \xrightarrow{-0.4} (4,3) \xrightarrow{+1} \end{aligned}$$

- ▷ The next transition reaches (3,3), (known high utility from trial 1)
- ▷ Bellman equation: \sim high $U^\pi(3,2)$ because $(3,2) \xrightarrow{-0.4} (3,3)$
- ▷ But direct utility estimation learns nothing until the end of the trial.

Intuition: Direct utility estimation searches for U in a hypothesis space that too large \leftrightarrow many functions that violate the Bellman equations.

▷ Thus the algorithm often converges very slowly.



Adaptive Dynamic Programming

- ▷ **Idea:** Take advantage of the constraints among the utilities of states by
 - ▷ learning the transition model that connects them,
 - ▷ solving the corresponding Markov decision process using a dynamic programming method.
- This means plugging the learned transition model $P(s' | s, \pi(s))$ and the observed rewards $R(s)$ into the Bellman equations (21.2) to calculate the utilities of the states.
- ▷ **As above:** these equations are linear (no maximization involved) (solve with any any linear algebra package).
- ▷ **Observation 28.2.11** *Learning the model itself is easy, because the environment is fully observable.*
- ▷ **Corollary 28.2.12** *We have a supervised learning task where the input is a state-action pair and the output is the resulting state.*
 - ▷ *In the simplest case, we can represent the transition model as a table of probabilities.*
 - ▷ *Count how often each action outcome occurs and estimate the transition probability $P(s' | s, a)$ from the frequency with which s' is reached by action a in s .*
- ▷ **Example 28.2.13** In the 3 trials from the example above, *Right* is executed 3 times in (1, 3) and 2 times the result is (2, 3), so $P((2, 3) | (1, 3), \text{Right})$ is estimated to be 2/3.



Passive ADP Learning Algorithm

- ▷ **Definition 28.2.14** The **passive ADP algorithm** is given by

```

function PASSIVE-ADP-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal  $r'$ 
  persistent:  $\pi$  a fixed policy
     $mdp$ , an MDP with model  $P$ , rewards  $R$ , discount  $\gamma$ 
     $U$ , a table of utilities, initially empty
     $N_{sa}$ , a table of frequencies for state-action pairs, initially zero
     $N_{s'|sa}$ , a table of outcome frequencies given state-action pairs, initially zero
     $s, a$ , the previous state and action, initially null
  if  $s'$  is new then  $U[s'] := r'; R[s'] := r'$ 
  if  $s$  is not null then
    increment  $N_{sa}[s, a]$  and  $N_{s'|sa}[s', s, a]$ 

```

```

for each  $t$  such that  $N_{s|sa}[t, s, a]$  is nonzero do
     $P(t|s, a) := N_{s'|sa}[t, s, a]/N_{sa}[s, a]$ 
     $U := \text{POLICY-EVALUATION}(\pi, mdp)$ 
    if  $s'.\text{TERMINAL?}$  then  $s, a := \text{null}$  else  $s, a := s', \pi[s']$ 
return  $a$ 

```

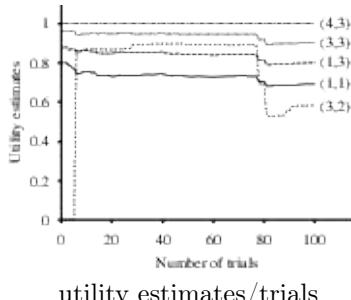
POLICY-EVALUATION computes $U^\pi(s) := E[\sum_{t=0}^{\infty} \gamma^t R(s_t)]$ in a MDP.



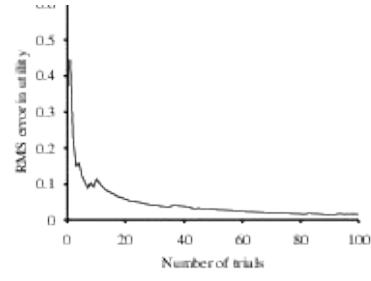
Passive ADP Convergence

▷ Example 28.2.15 (Passive ADP learning curves for the 4x3 world)

Given the optimal policy from the 4x3 Example above



utility estimates/trials



error for $U(1, 1)$: 20 runs of 100 trials

Note the large changes occurring around the 78th trial – this is the first time that the agent falls into the -1 terminal state at (4,2).

▷ Observation 28.2.16 *The ADP agent is limited only by its ability to learn the transition model.* (*intractable for large state spaces*)

▷ Example 28.2.17 In backgammon, roughly 10^{50} equations in 10^{50} unknowns.

▷ Idea: Use this as a baseline to compare passive learning algorithms



28.3 Active Reinforcement Learning

Active Reinforcement Learning

▷ Recap: A passive learning agent has a fixed policy that determines its behavior.

▷ An active agent must also decide what actions to take.

▷ Idea: Adapt the passive ADP algorithm to handle this new freedom.

▷ learn a complete model with outcome probabilities for all actions, rather than just the model for the fixed policy. (*use PASSIVE-ADP-AGENT*)

- ▷ choose actions; the utilities to learn are defined by the optimal policy, they obey the Bellman equation:

$$U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \left(\sum_{s'} U(s') \cdot P(s' | s, a) \right)$$

- ▷ solve with value/policy iteration techniques from Section 24.3.

- ▷ choose a good action, e.g.

- ▷ by one-step lookahead to maximize expected utility, or

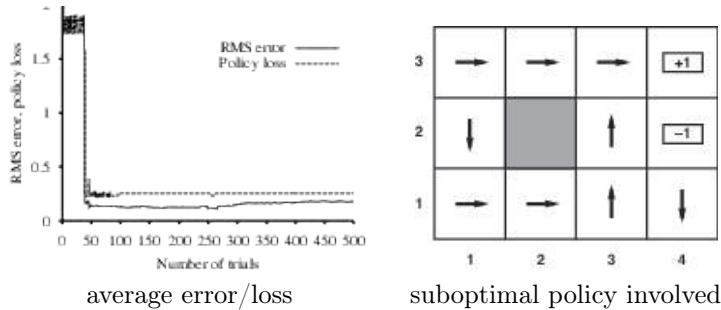
- ▷ if it uses **policy iteration** and has optimal policy, execute that.

This agent/algorithms is **greedy**, since it only optimizes the next step



Greedy ADP Learning (Evaluation)

- ▷ Example 28.3.1 (Greedy ADP learning curves for the 4x3 world)



The agent follows the optimal policy for the learned model at each step.

- ▷ It does not learn the true utilities or the true optimal policy!
- ▷ instead, in the 39th trial, it finds a policy that reaches the +1 reward along the lower route via (2,1), (3,1), (3,2), and (3,3).
- ▷ After experimenting with minor variations, from the 276th trial onward it sticks to that policy, never learning the utilities of the other states and never finding the optimal route via (1,2), (1,3), and (2,3).



Exploration in Active Reinforcement Learning

- ▷ Observation 28.3.2 *Greedy active ADP learning agents very seldom converge against the optimal solution*

- ▷ *The learned model is not the same as the true environment,*

- ▷ *What is optimal in the learned model need not be in the true environment.*

- ▷ What can be done? The agent does not know the true environment.
- ▷ **Idea:** actions do more than provide rewards according to the learned model
 - ▷ they also contribute to learning the true model by affecting the percepts received.
 - ▷ By improving the model, the agent may reap greater rewards in the future.
- ▷ **Observation 28.3.3** *An agent must make a tradeoff between*
 - ▷ *exploitation* to maximize its reward – as reflected in its current utility estimates and
 - ▷ *exploration* to maximize its long-term well-being.
- Pure exploitation risks getting stuck in a rut. Pure exploration to improve one's knowledge is of no use if one never puts that knowledge into practice.
- ▷ Compare with the information gathering agent from earlier in the course.



Part VII

Communication

This part introduces the basics of natural language processing and the use of language for communication with humans.

Fascination of Language

- ▷ Even more so than thinking, language is a skill that only humans have.
- ▷ It is a miracle that we can express complex thoughts in a sentence in a matter of seconds.
- ▷ It is no less miraculous that a child can learn tens of thousands of words and a complex grammar in a matter of a few years.



©: Michael Kohlhase

922



Natural Language and AI

- ▷ Ca. 100.000 years ago, humans learned to speak, ca. 7.000 years ago, to write
- ▷ Alan Turing based his test on natural language (for good reason)
 - ▷ we want AI agents to be able to communicate with humans
 - ▷ we want AI agents to be able to acquire knowledge from written documents
- ▷ In this Part, we analyze the problem with specific information-seeking tasks
 - ▷ Language Models (Which strings are English/Spanish/etc.)
 - ▷ Text Classification (E.g. spam detection)
 - ▷ Information Retrieval (aka. Search Engines)
 - ▷ Information Extraction (finding objects and their relations in texts)



©: Michael Kohlhase

923



Chapter 29

Natural Language Processing

29.1 Introduction to NLP

Even though this course concentrates on computational aspects of natural language semantics, it is useful to see it in the context of the field of [natural language processing](#).

The general context of AI-1 is [natural language processing \(NLP\)](#), and in particular [natural language understanding \(NLU\)](#). The dual side of [NLU](#): [natural language generation \(NLG\)](#) requires similar foundations, but different techniques is less relevant for the purposes of this course.

What is Natural Language Processing?

- ▷ [Generally](#): Studying of natural languages and development of systems that can use/generate these.
- ▷ **Definition 29.1.1** [Natural language processing \(NLP\)](#) is an engineering field at the intersection of computer science, artificial intelligence, and linguistics which is concerned with the interactions between computers and human (natural) languages. Many challenges in NLP involve
 - ▷ [natural language understanding \(NLU\)](#) – that is, enabling computers to derive meaning (representations) from human or natural language input
 - ▷ [natural language generation](#) which aims at generating natural language or speech from meaning representation.



©: Michael Kohlhase

924



Language Technology

- ▷ Language Assistance
 - ▷ written language: Spell-/grammar-/style-checking
 - ▷ spoken language: dictation systems and screen readers
 - ▷ multilingual text: machine-supported text and dialog translation, eLearning
- ▷ Dialog Systems
 - ▷ Information Systems: at airport, tele-banking, e-commerce, call centers

- ▷ Dialog interfaces for computers, robots, cars (e.g. Siri/Alexa)
- ▷ Information management:
 - ▷ search and classification of documents (e.g.. Google/Bing)
 - ▷ information extraction, question answering. (e.g. <http://ask.com>)



©: Michael Kohlhase

925



What is the State of the Art In NLU?

- ▷ Two avenues of attack for the problem: knowledge-based and statistical techniques (they are complementary)

Deep	Knowledge-based We are here	Not there yet cooperation?
Shallow	no-one wants this	Statistical Methods applications
Analysis ↑ vs. Coverage →	narrow	wide

- ▷ We will cover foundational methods of deep processing in the course and a mixture of deep and shallow ones in the lab.



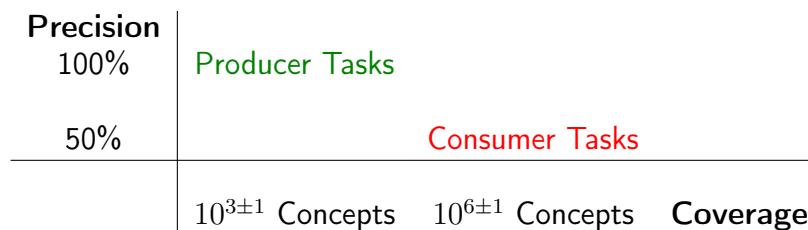
©: Michael Kohlhase

926



Environmental Niches for both Approaches to NLU

- ▷ There are two kinds of applications/tasks in NLU
 - ▷ consumer-grade applications have tasks that must be fully generic, and wide coverage (e.g. machine translation ~ Google Translate)
 - ▷ producer-grade applications must be high-precision, but domain-adapted (multilingual documentation, voice-control, ambulance translation)



- ▷ A producer domain I am interested in: Mathematical/Technical documents



29.2 Natural Language and its Meaning

A good probe into the issues involved in natural language understanding is to look at translations between natural language utterances – a task that arguably involves understanding the utterances first.

Meaning of Natural Language; e.g. Machine Translation

- ▷ **Idea:** Machine Translation is very simple! (we have good lexica)
- ▷ **Example 29.2.1** *Peter liebt Maria.* ~ *Peter loves Mary.*
- ▷ **⚠** this only works for simple examples
- ▷ **Example 29.2.2** *Wirf der Kuh das Heu über den Zaun.* ↗ *Throw the cow the hay over the fence.* (differing grammar; Google Translate)
- ▷ **Example 29.2.3 ⚠** Grammar is not the only problem
 - ▷ *Der Geist ist willig, aber das Fleisch ist schwach!*
 - ▷ *Der Schnaps ist gut, aber der Braten ist verkocht!*
- ▷ **We have to understand the meaning!**



If it is indeed the meaning of natural language, we should look further into how the form of the utterances and their meaning interact.

Language and Information

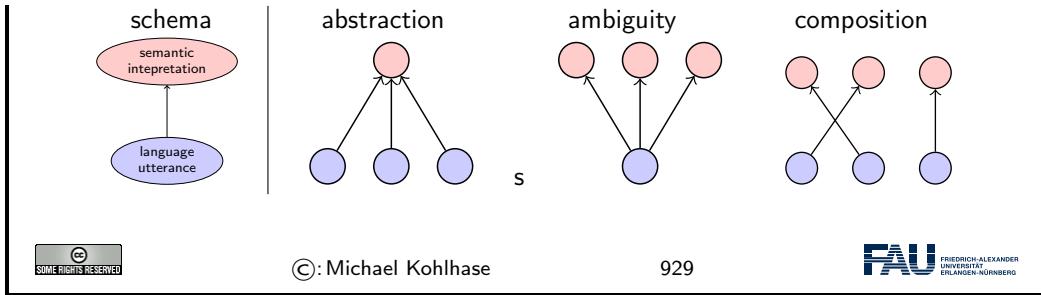
- ▷ **Observation:** Humans use words (sentences, texts) in natural languages to represent and communicate information.
- ▷ **But:** what really counts is not the **words** themselves, but the **meaning information** they carry.



- ▷ **Example 29.2.4**

Newspaper ~

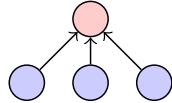
- ▷ for questions/answers, it would be very useful to find out what words (sentences/texts) mean.
- ▷ Interpretation of natural language utterances: three problems



Let us support the last claim a couple of initial examples. We will come back to these phenomena again and again over the course of the course and study them in detail.

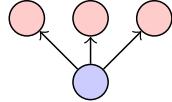
Language and Information (Examples)

▷ Example 29.2.5 (Abstraction)



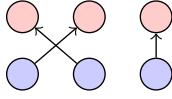
car and automobile have the same meaning

▷ Example 29.2.6 (Ambiguity)



a bank can be a financial institution or a geographical feature

▷ Example 29.2.7 (Composition)



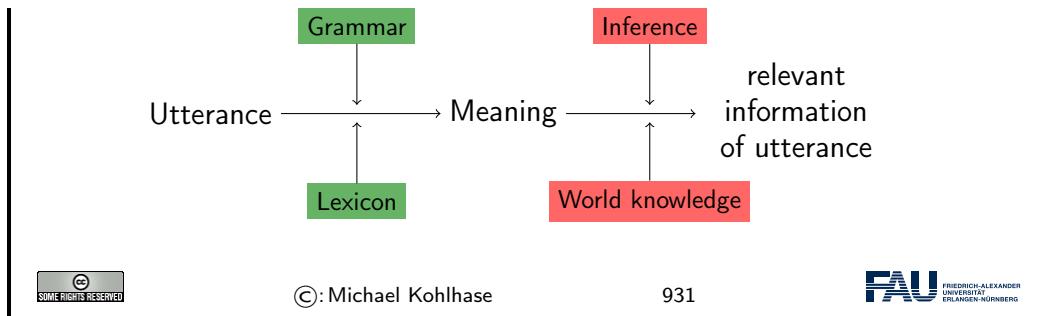
Every student sleeps $\sim \forall x.\text{student}(x) \Rightarrow \text{sleep}(x)$



But there are other phenomena that we need to take into account when compute the meaning of NL utterances

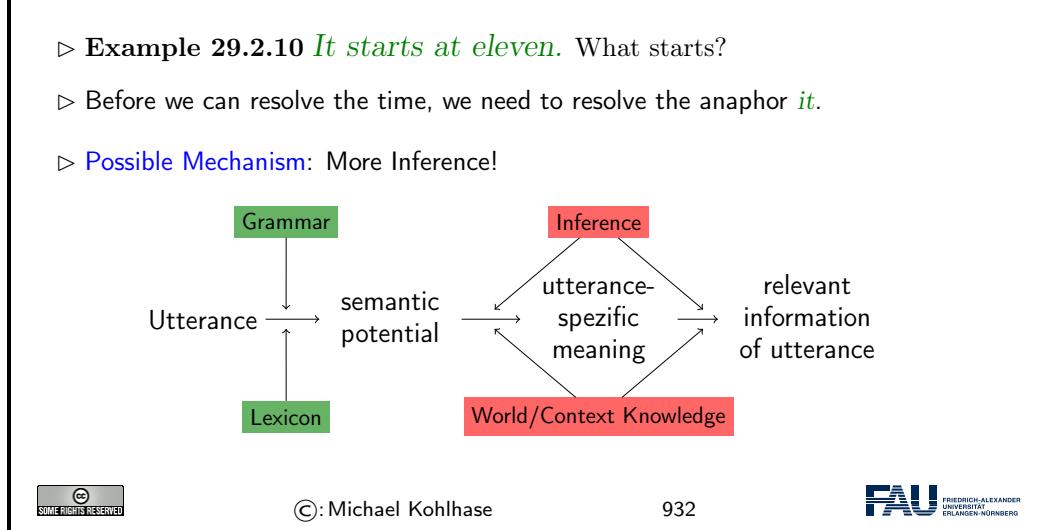
Context Contributes to the Meaning of NL Utterances

- ▷ **Observation:** Not all information conveyed is **linguistically realized** in an utterance.
- ▷ **Example 29.2.8** *The lecture begins at 11:00 am.* What lecture? Today?
- ▷ **Definition 29.2.9** We call a piece i of information **linguistically realized** in an utterance U , iff, we can trace i to a fragment of U .
- ▷ **Possible Mechanism:** Inference



Context Contributes to the Meaning of NL Utterances

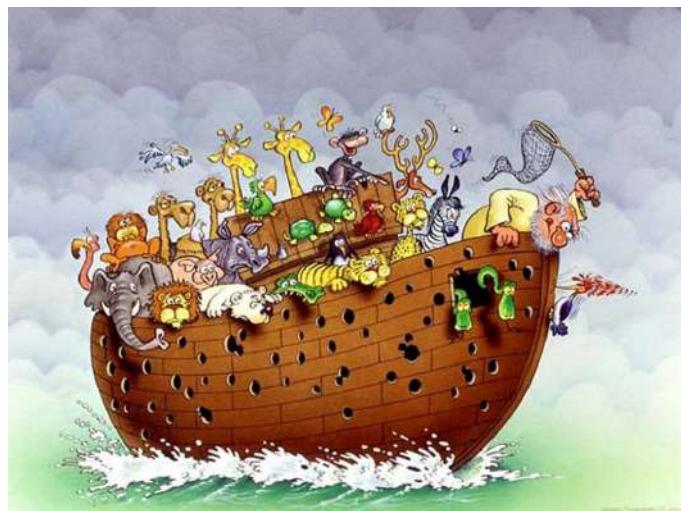
- ▷ Example 29.2.10 *It starts at eleven*. What starts?
- ▷ Before we can resolve the time, we need to resolve the anaphor *it*.
- ▷ Possible Mechanism: More Inference!



We end this very high-level introduction with a caveat.

Semantics is not a Cure-It-All!

How many animals of each species did Moses take onto the ark?



▷ Actually, it was Noah

(But you understood the question anyways)



©: Michael Kohlhase

933



But Semantics works in some cases

▷ The only thing that currently really helps is a restricted domain

▷ restricted vocabulary and world model

Demo: DBpedia <http://dbpedia.org/snorql/>

Query: Soccer players, who are born in a country with more than 10 million inhabitants, who played as goalkeeper for a club that has a stadium with more than 30.000 seats and the club country is different from the birth country



©: Michael Kohlhase

934



But Semantics works in some cases

▷ Answer: is computed by DBpedia from a SPARQL Query

```
SELECT distinct ?soccerplayer ?countryOfBirth ?team ?countryOfTeam ?stadiumcapacity
{
?soccerplayer a dbo:SoccerPlayer ;
  dbo:position|dpb: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 ;
?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_Benabbelih	:Algeria	:Wydad_Casablanca	:Morocco	67000
:Ailton_Moraes_Michelon	:Brazil	:FC_Red_Bull_Salzburg	:Austria	31000
:Alain_Gouméné	: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
:Breiner_Castillo	:Colombia	:Deportivo_Táchira	:Venezuela	38755
:Carlos_Luis_Morales	:Ecuador	:Club_Atlético_Independiente	:Argentina	48069
:Carlos_Navarro_Montoya	:Colombia	:Club_Colo_Colo	:Argentina	48069
:Cristián_Muñoz	:Argentina	:FBC_Melgar	:Chile	47000
:Daniel_Ferreira	:Argentina	:Karşıyaka_S.K.	:Turkey	60000
:David_Bílek	:Czech_Republic	:Karşıyaka_S.K.	:Peru	51295
:David_Loria	:Kazakhstan	:Karşıyaka_S.K.	:Turkey	51295
:Denys_Boyko	:Ukraine	:Beşiktaş_J.K.	:Turkey	41903
:Eddie_Gustafsson	:United_States	:FC_Red_Bull_Salzburg	:Austria	31000
:Emilian_Dolha	:Romania	:Lech_Poznań	:Poland	43269
:Eusebio_Acasuso	:Peru	:Club_Bolívar	:Bolivia	42000
:Faryd_Mondragón	:Colombia	:Real_Zaragoza	:Spain	34596
:Faryd_Mondragón	:Colombia	:Club_Atlético_Independiente	:Argentina	48069
:Federico_Vilar	:Argentina	:Club_Atlas	:Mexico	54500
:Fernando_Martínez	:Argentina	:Real_Garcilaso	:Peru	45000
:Fábio_André_da_Silva	:Portugal	:Servette_FC	:Switzerland	30084
:Gerhard_Tremmel	:Germany	:FC_Red_Bull_Salzburg	:Austria	31000
:Gilt_Muzatzi	:United_Kingdom	:Lech_Poznań	:Poland	43269
:Günay_Güvenç	:Germany	:Beşiktaş_J.K.	:Turkey	41903
:Hugo_Marques	:Portugal	:C.D._Primeiro_de_Agosto	:Angola	48500
:Héctor_Landazuri	:Colombia	:La_Paz_F.C.	:Bolivia	42000



©: Michael Kohlhase

935



Even if we can get a perfect grasp of the semantics (aka. meaning) of NL utterances, their structure and context dependency – we will try this in this lecture, but of course fail, since the issues are

much too involved and complex for just one lecture – then we still cannot account for all the human mind does with language.

But there is hope, for limited and well-understood domains, we can do amazing things. This is what this course tries to show, both in theory as well as in practice.

29.3 Looking at Natural Language

The next step will be to make some observations about natural language and its meaning, so that we get an intuition of what problems we will have to overcome on the way to modeling natural language.

Fun with Diamonds (are they real?) [Dav67]

- ▷ **Example 29.3.1** We study the [truth conditions](#) of adjectival complexes

<ul style="list-style-type: none"> ▷ <i>This is a blue diamond</i> ▷ <i>This is a big diamond</i> ▷ <i>This is a fake diamond</i> ▷ <i>This is a fake blue diamond</i> ▷ <i>Mary knows that this is a diamond</i> ▷ <i>Mary believes that this is a diamond</i> 	$(\models \text{diamond}, \models \text{blue})$ $(\models \text{diamond}, \not\models \text{big})$ $(\not\models \text{diamond})$ $(\models \text{blue?}, \models \text{diamond?})$ $(\models \text{diamond})$ $(\not\models \text{diamond})$
---	--



©: Michael Kohlhase

936



Logical analysis vs. conceptual analysis: These examples — Mostly borrowed from [Dav67] — help us to see the difference between logical analysis and conceptual analysis. We observed that from *This is a big diamond*. we cannot conclude *This is big*. Now consider the sentence *Jane is a beautiful dancer*. Similarly, it does not follow from this that Jane is beautiful, but only that she dances beautifully. Now, what it is to be beautiful or to be a beautiful dancer is a complicated matter. To say what these things are is a problem of conceptual analysis. The job of semantics is to uncover the logical form of these sentences. Semantics should tell us that the two sentences have the same logical forms; and ensure that these logical forms make the right predictions about the entailments and truth conditions of the sentences, specifically, that they don't entail that the object is big or that Jane is beautiful. But our semantics should provide a distinct logical form for sentences of the type: *This is a fake diamond*. From which it follows that the thing is fake, but not that it is a diamond.

Ambiguity: The dark side of Meaning

- ▷ **Definition 29.3.2** We call an utterance [ambiguous](#), iff it has multiple meanings, which we call [readings](#).
- ▷ **Example 29.3.3** All of the following sentences are ambiguous:

<ul style="list-style-type: none"> ▷ <i>John went to the bank</i> ▷ <i>You should have seen the bull we got from the pope</i> ▷ <i>I saw her duck</i> ▷ <i>John chased the gangster in the red sports car</i> 	$(\text{river or financial?})$ (three readings!) $(\text{animal or action?})$ (three-way too!)
---	---



One way to think about the examples of ambiguity on the previous slide is that they illustrate a certain kind of indeterminacy in sentence meaning. But really what is indeterminate here is what sentence is represented by the physical realization (the written sentence or the phonetic string). The symbol *duck* just happens to be associated with two different things, the noun and the verb. Figuring out how to interpret the sentence is a matter of deciding which item to select. Similarly for the syntactic ambiguity represented by PP attachment. Once you, as interpreter, have selected one of the options, the interpretation is actually fixed. (This doesn't mean, by the way, that as an interpreter you necessarily do select a particular one of the options, just that you can.)

A brief digression: Notice that this discussion is in part a discussion about compositionality, and gives us an idea of what a non-compositional account of meaning could look like. The Radical Pragmatic View is a non-compositional view: it allows the information content of a sentence to be fixed by something that has no linguistic reflex.

To help clarify what is meant by compositionality, let me just mention a couple of other ways in which a semantic account could fail to be compositional.

- Suppose your syntactic theory tells you that S has the structure $[a[bc]]$ but your semantics computes the meaning of S by first combining the meanings of a and b and then combining the result with the meaning of c . This is non-compositional.
- Recall the difference between:
 1. Jane knows that George was late.
 2. Jane believes that George was late.

Sentence 1. entails that George was late; sentence 2. doesn't. We might try to account for this by saying that in the environment of the verb *believe*, a clause doesn't mean what it usually means, but something else instead. Then the clause *that George was late* is assumed to contribute different things to the informational content of different sentences. This is a non-compositional account.

Quantifiers, Scope and Context

- ▷ *Every man loves a woman* (Keira Knightley or his mother!)
- ▷ *Every car has a radio* (only one reading!)
- ▷ **Example 29.3.4** *Some student in every course sleeps in every class at least some of the time* (how many readings?)
- ▷ **Example 29.3.5** *The president of the US is having an affair with an intern* (2002 or 2000?)
- ▷ **Example 29.3.6** *Everyone is here* (who is everyone?)



Observation: If we look at the first sentence, then we see that it has two readings:

1. there is one woman who is loved by every man.
2. for each man there is one woman whom that man loves.

These correspond to distinct situations (or possible worlds) that make the sentence true.

Observation: For the second example we only get one reading: the analogue of 2. The reason for this lies not in the logical structure of the sentence, but in concepts involved. We interpret the meaning of the word *has*¹² as the relation “has as physical part”, which in our world carries a certain uniqueness condition: If *a* is a physical part of *b*, then it cannot be a physical part of *c*, unless *b* is a physical part of *c* or vice versa. This makes the structurally possible analogue to 1. impossible in our world and we discard it. EdN:12

Observation: In the examples above, we have seen that (in the worst case), we can have one reading for every ordering of the quantificational phrases in the sentence. So, in the third example, we have four of them, we would get $4! = 12$ readings. It should be clear from introspection that we (humans) do not entertain 12 readings when we understand and process this sentence. Our models should account for such effects as well.

Context and Interpretation: It appears that the last two sentences have different informational content on different occasions of use. Suppose I say *Everyone is here*. at the beginning of class. Then I mean that everyone who is meant to be in the class is here. Suppose I say it later in the day at a meeting; then I mean that everyone who is meant to be at the meeting is here. What shall we say about this? Here are three different kinds of solution:

Radical Semantic View On every occasion of use, the sentence literally means that everyone in the world is here, and so is strictly speaking false. An interpreter recognizes that the speaker has said something false, and uses general principles to figure out what the speaker actually meant.

Radical Pragmatic View What the semantics provides is in some sense incomplete. What the sentence means is determined in part by the context of utterance and the speaker’s intentions. The differences in meaning are entirely due to extra-linguistic facts which have no linguistic reflex.

The Intermediate View The logical form of sentences with the quantifier *every* contains a slot for information which is contributed by the context. So extra-linguistic information is required to fix the meaning; but the contribution of this information is mediated by linguistic form.

More Context: Anaphora

- ▷ *John is a bachelor. His wife* is very nice. (Uh, what?, who?)
- ▷ *John likes his dog Spiff even though he bites him sometimes.* (who bites?)
- ▷ *John likes Spiff. Peter does too.* (what to does Peter do?)
- ▷ *John loves his wife. Peter does too.* (whom does Peter love?)
- ▷ *John loves golf, and Mary too.* (who does what?)



©: Michael Kohlhase

939



Context is Personal and keeps changing

- ▷ *The king of America is rich.* (true or false?)
- ▷ *The king of America isn't rich.* (false or true?)

¹²EDNOTE: fix the nlex macro, so that it can be used to specify which example a fragment has been taken from.

- ▷ *If America had a king*, the king of America would be rich. (true or false!)
 - ▷ *The king of Buganda is rich.* (Where is Buganda?)
 - ▷ *... Joe Smith... The CEO of Westinghouse announced budget cuts.* (CEO=J.S.!) (True or False?)



29.4 Language Models

Natural Languages vs. Formal Language

- ▷ **Recap:** A formal language is a set of strings.
 - ▷ **Example 29.4.1** Programming languages like Java or C++ are formal languages.
 - ▷ **Remark 29.4.2** Natural languages like English, German, or Spanish are not.
 - ▷ **Example 29.4.3** Let us look at concrete examples
 - ▷ *Not to be invited is sad* (definitely English)
 - ▷ *To not be invited is sad* (controversial)
 - ▷ **Idea:** Let's be lenient, instead of a hard set, use a probability distribution.
 - ▷ **Definition 29.4.4** A (statistical) language model is a probability distribution over sequences of characters or words.
 - ▷ **Idea:** Try to learn/derive language models from text corpora.
 - ▷ **Definition 29.4.5** In linguistics, a text corpus (or simply corpus; plural corpora) is a large and structured set of texts. In corpus linguistics, corpora are used to do statistical analysis and hypothesis testing, checking occurrences or validating linguistic rules within a specific language territory.



N-gram Character Models

- ▷ Written text is composed of **characters** – letters, digits, punctuation, and spaces.
 - ▷ **Idea:** Let's study language models for sequences of **characters**.
 - ▷ As for Markov processes, we write $P(\mathbf{c}_{1:N})$ for the **probability** of a **character sequence** $c_1 \dots c_N$ of length N .
 - ▷ **Definition 29.4.6** We call an n **characters** sequence of length n an **n -gram** (**unigram**, **bigram**, **trigram** for $n = 1, 2, 3$).
 - ▷ **Definition 29.4.7** An **n -gram model** is a Markov chain of order $n - 1$.

- ▷ **Remark 29.4.8** For a [trigram model](#), we have $P(c_i \mid \mathbf{c}_{1:i-1}) = P(c_i \mid \mathbf{c}_{i-2:i-1})$ factoring with the chain rule and then using the Markov property, we obtain

$$P(\mathbf{c}_{1:N}) = \prod_{i=1}^N P(c_i \mid \mathbf{c}_{1:i-1}) = \prod_{i=1}^N P(c_i \mid \mathbf{c}_{i-2:i-1})$$

- ▷ A [trigram model](#) for a language with 100 [characters](#), $\mathbf{P}(c_i \mid \mathbf{c}_{i-2:i-1})$ has 1.000.000 entries. It can be estimated from a [corpus](#) with 10^7 [characters](#).



Applications of N -Gram Models of Character Sequences

- ▷ What can we do with N -gram models?
- ▷ **Definition 29.4.9** The problem of [language identification](#) is given a text, determine the natural language it is written in.
- ▷ **Remark 29.4.10** Current technology can classify even short texts like [Hello, world](#), or [Wie geht es Dir](#) correctly with more than 99% accuracy.
- ▷ **One approach:** build a trigram language model $\mathbf{P}(c_i \mid \mathbf{c}_{i-2:i-1}, \ell)$ for each candidate language ℓ by counting trigrams in a ℓ -corpus.

Apply Bayes' rule and the Markov property to get the most likely language:

$$\begin{aligned}\ell^* &= \underset{\ell}{\operatorname{argmax}}(P(\ell \mid \mathbf{c}_{1:N})) \\ &= \underset{\ell}{\operatorname{argmax}}(P(\ell) \cdot P(\mathbf{c}_{1:N} \mid \ell)) \\ &= \underset{\ell}{\operatorname{argmax}}(P(\ell) \cdot \prod_{i=1}^N P(c_i \mid \mathbf{c}_{i-2:i-1}, \ell))\end{aligned}$$

The prior probability $P(\ell)$ can be estimated, it is not a critical factor, since the trigram language models are extremely sensitive.



Other Applications of Character N -Gram Models

- ▷ Spelling correction is a direct application of a single-language language model: Estimate the probability of a word and all off-by-one variants.
- ▷ **Definition 29.4.11** [Genre classification](#) means deciding whether a text is a news story, a legal document, a scientific article, etc.
- ▷ **Remark 29.4.12** While many features help make this classification, counts of punctuation and other character n -gram features go a long way [KNS97].
- ▷ **Definition 29.4.13** [Named entity recognition](#) is the task of finding names of things in a document and deciding what class they belong to.

- ▷ **Example 29.4.14** In *Mr. Sopersteen was prescribed aciphex.* we should recognize that *Mr. Sopersteen* is the name of a person and *aciphex* is the name of a drug.
- ▷ **Remark 29.4.15** Character-level language models are good for this task because they can associate the character sequence *ex* with a drug name and *steen* with a person name, and thereby identify words that they have never seen before.



N-Grams over Word Sequences

- ▷ **Idea:** N-gram models apply to word sequences as well.
- ▷ **Problems:** The method works identically, but
 1. There are many more words than characters. (100 vs. 10^6 in English)
 2. And what is a word anyways? (space/punctuation-delimited substrings?)
 3. Most training corpora do not have all words.
- ▷ **Definition 29.4.16** Out-of-vocabulary (OOV) words are unknown words that appear in the test corpus but not training corpus.
- ▷ **Remark 29.4.17** OOV words are usually content words such as names and locations which contain information crucial to the success of NLP tasks.
- ▷ **Idea:** Model OOV words by
 - ▷ adding a new word token, e.g. <UNK> to the vocabulary.
 - ▷ in the training corpus, replace the respective first occurrence of a previously unknown word by <UNK>.
 - ▷ count n-grams as usual, treating <UNK> as a regular word.

This trick can be refined if we have a word classifier, then use a new token per class, e.g. <EMAIL> or <NUM>.



What can Word N-Gram Models do?

- ▷ **Example 29.4.18 (Test n-grams)** Build unigram, bigram, and trigram language models over the words [RN03], randomly sample sequences from the models.
 1. Unigram: *logical are as are confusion a may right tries agent goal the was ...*
 2. Bigram: *systems are very similar computational approach would be represented ...*
 3. Trigram: *planning and scheduling are integrated the success of naive bayes model ...*

There are differences, how can we measure them to evaluate the models?

▷ **Definition 29.4.19** The **perplexity** of a sequence $\mathbf{c}_{1:N}$ is defined as

$$\text{Perplexity}(\mathbf{c}_{1:N}) := P(\mathbf{c}_{1:N})^{-\left(\frac{1}{N}\right)}$$

▷ **Example 29.4.20** For a language with n **characters** or **words** and a language model that predicts that all are equally likely, the perplexity of any sequence is n .

If some **characters** or **words** are more likely than others, and the model reflects that, then the perplexity of correct sequences will less than n .

▷ The perplexity was 891 for the unigram model, 142 for the bigram model and 91 for the trigram model.



29.5 Information Retrieval

Information Retrieval

▷ **Definition 29.5.1** **Information retrieval (IR)** deals with the representation, organization, storage, and maintenance of information objects so that it provides the user with easy access to the relevant information and satisfies the user's various **information needs**.

▷ We normally come in contact with IR in the form of **web search**.

▷ **Definition 29.5.2** **Web search** is fully automatic process that responds to a **user query** by returning a sorted document list relevant to the user requirements expressed in the query.

▷ **Example 29.5.3** Google and Bing and web search engines, their query is a **bag** of words and documents are web pages, PDFs, images, videos, shopping portals.



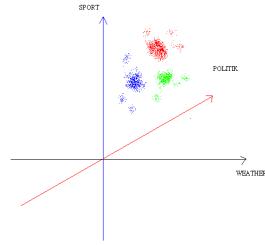
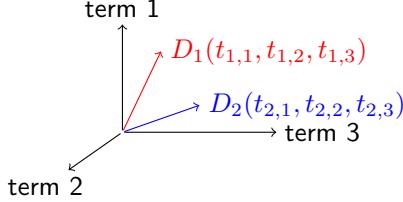
Vector Space Models for IR

▷ **Observation 29.5.4** *Documents and queries induce bags of words over a fixed vocabulary V , which can be represented as word frequency vectors in \mathbb{R}^V .*

▷ **Example 29.5.5** If we have two documents: $d_1 = \text{Haveagoodday!}$ and $d_2 = \text{Haveagreatday!}$, then we can use $V = \{\text{Have}, \text{a}, \text{good}, \text{great}, \text{day}\}$ and can represent **good** as $\langle 0, 0, 1, 0, 0 \rangle$, **great** as $\langle 0, 0, 0, 1, 0 \rangle$, and d_1 a $\langle 1, 1, 1, 0, 1 \rangle$.

▷ **Idea:** For **web search**, represent documents d_i and query q as word frequency vectors and return those d_i that are "similar" to q .

▷ **Idea:** Query and document are similar, iff the cosine of their vectors is small.



Vector Space Models for IR

- ▷ **Lemma 29.5.6 (Euclidean Dot Product Formula)** $A \cdot B = \|A\|_2 \|B\|_2 \cos \theta$, where θ is the angle between A and B
- ▷ **Definition 29.5.7** The cosine similarity of A and B is $\cos \theta = \frac{A \cdot B}{\|A\|_2 \|B\|_2}$.



TF-IDF: Term Frequency/Inverse Document Frequency

- ▷ **Definition 29.5.8 (Remark)** Given a document d and a word t , the term frequency $\text{tf}(t, d)$ is the number of times t occurs in d .
- ▷ **Observation 29.5.9** Word frequency vectors of a document d is just the vector of term frequencies $\langle \text{tf}(t_1, d), \dots, \text{tf}(t_n, d) \rangle$ for a vocabulary $V = \{t_1, \dots, t_n\}$.
- ▷ **Problem:** Word frequency vectors treat all the words equally.
- ▷ **Example 29.5.10** In an query *the brown cow*, the *the* is less important than *brown cow*. (because *the* is less specific)
- ▷ **Idea:** Introduce a weighting factor for the word frequency vector that de-emphasizes the dimension of the more (globally) frequent words.
- ▷ **Definition 29.5.11** Given a document collection $D = \{d_1, \dots, d_N\}$ and a word t the inverse document frequency is given by $\text{idf}(t, D) := \log_{10}(\frac{N}{\#\{d \in D \mid t \in d\}})$.
- ▷ **Definition 29.5.12** We define $\text{tfidf}(t, d, D) := \text{tf}(t, d) \cdot \text{idf}(t, D)$.
- ▷ **Idea:** Use the tfidf-vector with cosine similarity for information retrieval instead.



TF-IDF Example

- ▷ Let $D := \{d_1, d_2\}$ be a document corpus over the vocabulary

$$V = \{\text{this}, \text{is}, \text{a}, \text{sample}, \text{another}, \text{example}\}$$

with word frequency vectors $\langle 1, 1, 1, 2, 0, 0 \rangle$ and $\langle 1, 1, 0, 0, 2, 3 \rangle$.

▷ Then we compute for the word *this*

- ▷ $\text{tf}(\text{i}\text{this}, d_1) = \frac{1}{5} = 0.2$ and $\text{tf}(\text{i}\text{this}, d_2) = \frac{1}{7} \approx 0.14$.
- ▷ idf is constant over D , we have $\text{idf}(\text{i}\text{this}, D) = \log_{10}(\frac{2}{2}) = 0$,
- ▷ thus $\text{tfidf}(\text{i}\text{this}, d_1, D) = 0 = \text{tfidf}(\text{i}\text{this}, d_2, D)$. (*this occurs in both*)

▷ The word *example* is more interesting, since it occurs only in d_2 (thrice)

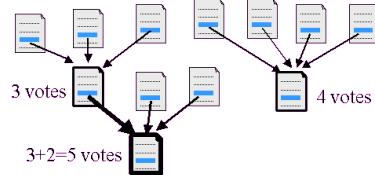
- ▷ $\text{tf}(\text{e}\text{x}\text{a}\text{m}\text{p}\text{l}\text{e}, d_1) = \frac{0}{5} = 0$ and $\text{tf}(\text{e}\text{x}\text{a}\text{m}\text{p}\text{l}\text{e}, d_2) = \frac{3}{7} \approx 0.429$.
- ▷ $\text{idf}(\text{e}\text{x}\text{a}\text{m}\text{p}\text{l}\text{e}, D) = \log_{10}(\frac{2}{1}) \approx 0.301$,
- ▷ thus $\text{tfidf}(\text{e}\text{x}\text{a}\text{m}\text{p}\text{l}\text{e}, d_1, D) = 0 \cdot 0.301 = 0$ and $\text{tfidf}(\text{e}\text{x}\text{a}\text{m}\text{p}\text{l}\text{e}, d_2, D) \approx 0.429 \cdot 0.301 = 0.129$.



Once an answer set has been determined, the results have to be sorted, so that they can be presented to the user. As the user has a limited attention span – users will look at most at three to eight results before refining a query, it is important to rank the results, so that the hits that contain information relevant to the user's information need early. This is a very difficult problem, as it involves guessing the intentions and information context of users, to which the search engine has no access.

Ranking Search Hits: e.g. Google's Page Rank

- ▷ **Problem:** There are many hits, need to sort them (e.g. by importance)
- ▷ **Idea:** A web site is important, ... if many other hyperlink to it.



▷ **Refinement:** ..., if many important web pages hyperlink to it.

▷ **Definition 29.5.13** Let A be a web page that is hyperlinked from web pages S_1, \dots, S_n , then the **page rank** PR of A is defined as

$$\text{PR}(A) = 1 - d + d \left(\frac{\text{PR}(S_1)}{C(S_1)} + \dots + \frac{\text{PR}(S_n)}{C(S_n)} \right)$$

where $C(W)$ is the number of links in a page W and $d = 0.85$.

▷ **Remark 29.5.14** $\text{PR}(A)$ is the probability of reaching A by random browsing.



Getting the ranking right is a determining factor for success of a search engine. In fact, the early

of Google was based on the pagerank algorithm discussed above (and the fact that they figured out a revenue stream using text ads to monetize searches).

29.6 Word Embeddings

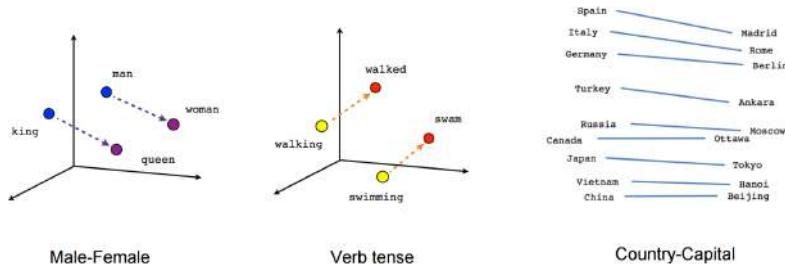
Word Embeddings

- ▷ **Problem:** For ML methods in NLP, we need numerical data. (not words)
- ▷ **Idea:** Embed words or word sequences into real-valued vector spaces.
- ▷ **Definition 29.6.1** A **word embedding** is a mapping from words in context into a real-valued vector space \mathbb{R}^n used for natural language processing.
- ▷ **Definition 29.6.2** A vector is called **one hot**, iff all components are 0 except for one 1. We call a word embedding one hot, iff all of its vectors are.
- ▷ **Example 29.6.3 (Vector Space Methods in Information Retrieval)**
Word frequency vectors are induced by adding up one-hot word embeddings.
- ▷ **Example 29.6.4** Given a document corpus D – the context – the **tf-idf word embedding** is given by $e: t \mapsto \langle \text{tfidf}(t, d_1, D), \dots, \text{tfidf}(t, d_{\#(D)}, D) \rangle$.
- ▷ **Intuition behind these two:** Words that occur in similar documents are similar.



Word2Vec: A Popular, Semantic Word Embedding

- ▷ **Distributional Semantics:** “a word is characterized by the company it keeps”.
- ▷ **Idea:** make word embeddings that takes context into account.
- ▷ **Result Preview:** semantic word embeddings



- ▷ **as before:** words that occur in similar documents are similar.

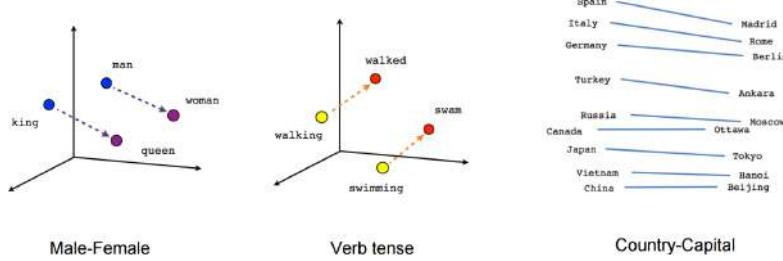
- ▷ **also in Word2Vec:** vector differences encode word relations

- ▷ **Algorithm Preview:** use a neural network to predict the word corresponding to an input context.



The Common Bag Of Words (CBOW) Algorithm I

▷ **Idea:** For the intended behavior



we need to maintain linear regularities, i.e. additive vector properties like:

$$V(\text{King}) - V(\text{Man}) + V(\text{Queen}) \quad \text{is close to} \quad V(\text{Woman})$$

▷ **Example 29.6.5** For the text *watch movies rather than read books*
context size: 2, target *rather*, we have

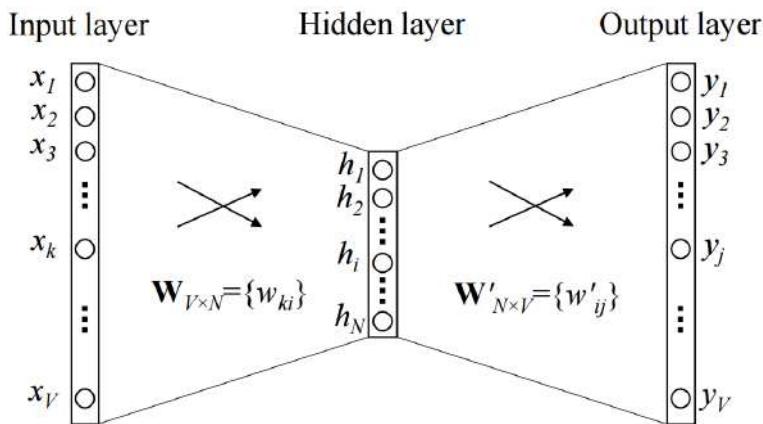
- ▷ context: $C := \{\text{watch}, \text{movies}, \text{than}, \text{read}\}$
- ▷ Vocabulary: $V := \{\text{watch}, \text{movies}, \text{rather}, \text{than}, \text{read}, \text{books}\}$

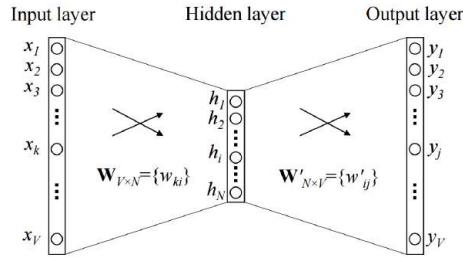
So in CBOW, build a neural network that
given the input $\{\text{watch}, \text{movies}, \text{than}, \text{read}\}$ produces *rather*.



The Common Bag Of Words (CBOW) Algorithm I

▷ A CBOW network for a single word





- ▷ The hidden layer neurons just copy the weighted sum of inputs to the next layer
(no threshold)
- ▷ The output layer computes the **softmax** of the hidden nodes

Weighted sums and **softmax** maintain linear regularities **(as intended)**

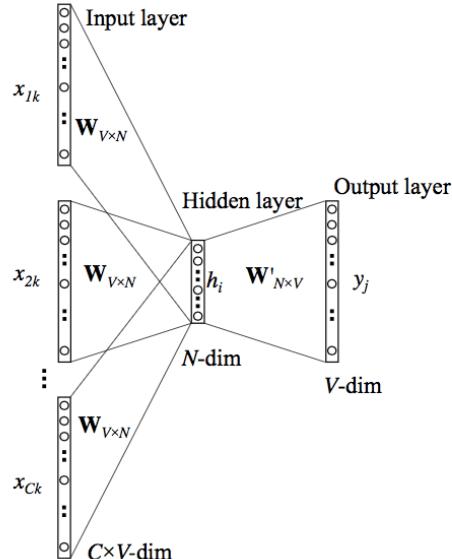
- ▷ **Definition 29.6.6** The **softmax** function $\sigma: \mathbb{R}^K \rightarrow \mathbb{R}^K$ is defined by

$$\sigma(z) := \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$



The Common Bag Of Words (CBOW) Algorithm II

- ▷ A neural network for a multiple words



Chapter 30

Natural Language for Communication

Outline

- ▷ Communication
- ▷ Grammar
- ▷ Syntactic analysis
- ▷ Problems (real Language Phenomena)



©: Michael Kohlhase

958



Communication

- ▷ “Classical” view (pre-1953):
 - ▷ language consists of sentences that are true/false (cf. logic)
- ▷ “Modern” view (post-1953):
 - ▷ language is a form of action
- ▷ Wittgenstein (1953) *Philosophical Investigations*
Austin (1962) *How to Do Things with Words*
Searle (1969) *Speech Acts*
- ▷ Why? *To change the actions of other agents*



©: Michael Kohlhase

959



Speech acts

SITUATION

Speaker → Utterance → Hearer



- ▷ Speech acts achieve the speaker's goals:

<i>Inform</i>	"There's a pit in front of you"
<i>Query</i>	"Can you see the gold?"
<i>Command</i>	"Pick it up"
<i>Promise</i>	"I'll share the gold with you"
<i>Acknowledge</i>	"OK"

- ▷ Speech act planning requires knowledge of

- ▷ Situation
- ▷ Semantic and syntactic conventions
- ▷ Hearer's goals, knowledge base, and rationality



©: Michael Kohlhase

960

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Stages in Communication (Informing)

Intention
Generation
Synthesis

- ▷ **Stages in communication (informing):** even here, the situation is complex

Perception
Analysis
Disambiguation
Incorporation

- ▷ **Question:** How could this go wrong?

1. Insincerity (S doesn't believe P)
(W ≠ W')
2. Speech wreck ignition failure (n > 1 and Pi ≠ P)
(C ≠ C')
3. Ambiguous utterance (n > 1 and Pi ≠ P)
(C ≠ C')
4. Differing understanding of current context (C ≠ C')



©: Michael Kohlhase

961

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

S wants to inform H the
S selects words W to express
S utters words W'

H perceives W' in context
H infers possible meanings
H infers intended meaning
H incorporates P_i into C'

Advertisement: Logic-Based Natural Language Semantics

- ▷ **Advanced Course:** "Logic-Based Natural Language Semantics" ([next semester](#))
- ▷ Wed. 10:15-11:50 and Thu 12:15-13:50 (expected: ≤ 10 Students)

- ▷ **Contents:** (Alternating Lectures and hands-on Lab Sessions)
 - ▷ Foundations of Natural Language Semantics (NLS)
 - ▷ Montague's Method of Fragments (Grammar, Semantics Constr., Logic)
 - ▷ Implementing Fragments in GLF (Grammatical Framework and MMT)
 - ▷ Inference Systems for Natural Language Pragmatics (tableau machine)
 - ▷ Advanced logical systems for NLS (modal, higher-order, dynamic Logics)
- ▷ **Grading:** Attendance & Wakefulness, Project/Homework, Oral Exam.
- ▷ **Course Intent:** Groom students for Bachelor/Master Theses and as KWARC research assistants.



©: Michael Kohlhase

962



Chapter 31

What did we learn in AI 1/2?

Topics of AI-1 (Winter Semester)

- ▷ Getting Started
 - ▷ What is Artificial Intelligence (situating ourselves)
 - ▷ Intelligent Agents (a unifying framework)
 - ▷ Logic Programming in Prolog (An influential paradigm)
- ▷ Problem Solving
 - ▷ Problem Solving and Search
 - ▷ Game playing (Adversarial Search)
 - ▷ Constraint Satisfaction Problems
- ▷ Knowledge and Reasoning
 - ▷ Formal Logic as the Mathematics of Meaning
 - ▷ Logic Programming
- ▷ Planning
 - ▷ Planning
 - ▷ Planning and Acting in the real world



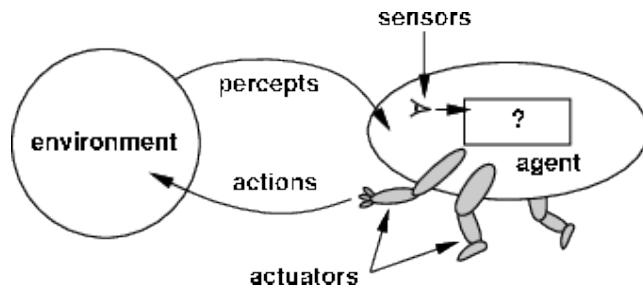
©: Michael Kohlhase

963

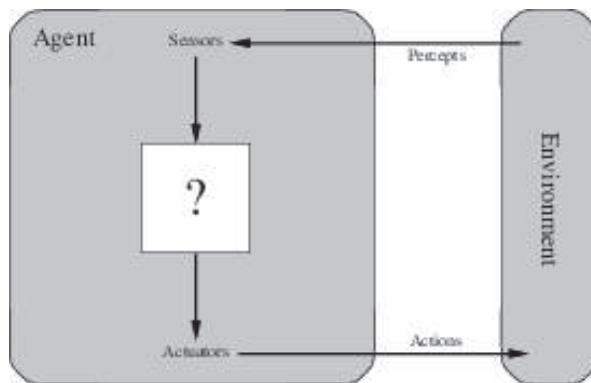


Rational Agents as an Evaluation Framework for AI

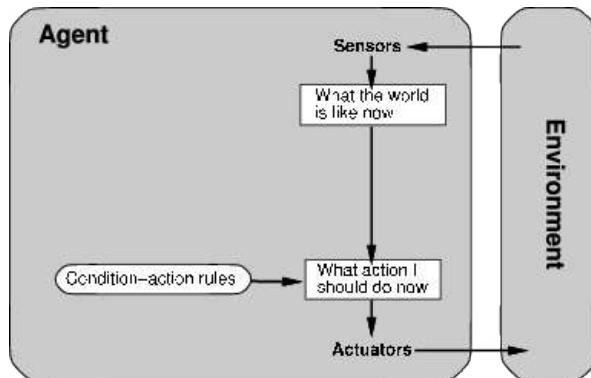
- ▷ Agents interact with the environment



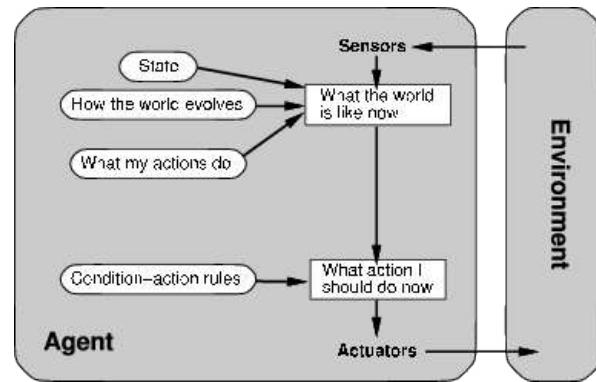
General agent schema



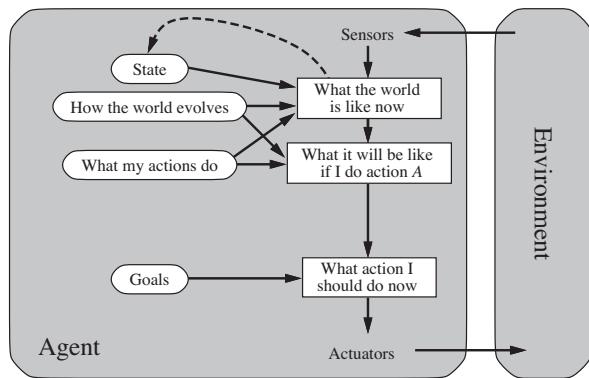
Simple Reflex Agents



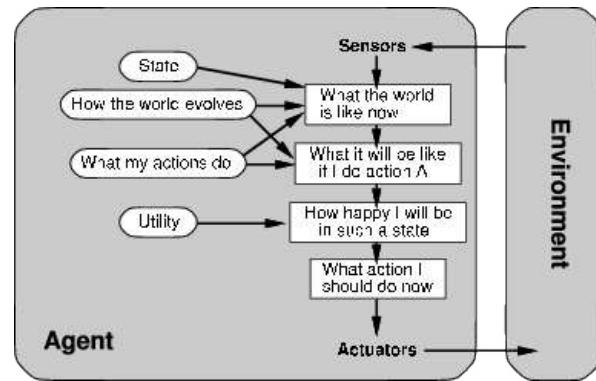
Reflex Agents with State



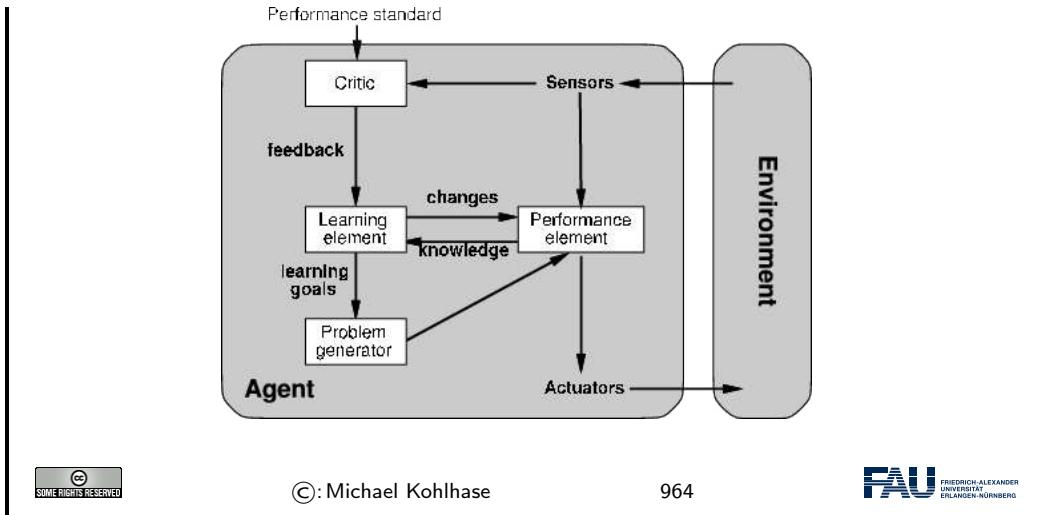
Goal-Based Agents



Utility-Based Agent



Learning Agents



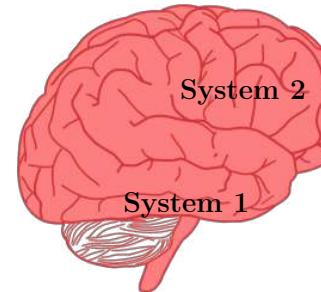
Rational Agent

- ▷ **Idea:** Try to design agents that are successful (do the right thing)
- ▷ **Definition 31.0.1** An agent is called **rational**, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date. This is called the **MEU principle**.
- Note:** a rational 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 \sim exploration, learning, autonomy

Thinking, Fast and Slow (two Brain systems)

- ▷ In his 2011 Bestseller **Thinking, fast and slow** [Kahnemann:tfss11], David Kahnemann posits a dichotomy between two modes of thought:
 - ▷ “System 1” is fast, instinctive and emotional;
 - ▷ “System 2” is slower, more deliberative, and more logical.

- ▷ System 1 can
 - ▷ see whether an object is near or far
 - ▷ complete the phrase *war and ...*
 - ▷ display disgust when seeing a gruesome image
 - ▷ solve $2+2=?$
 - ▷ read text on a billboard
- ▷ System 2 can
 - ▷ look out for the woman with the grey hair
 - ▷ sustain a higher than normal walking rate
 - ▷ count the number of A's in a certain text
 - ▷ give someone your phone number
 - ▷ park into a tight parking space
 - ▷ solve 17×24
 - ▷ determine the validity of a complex argument



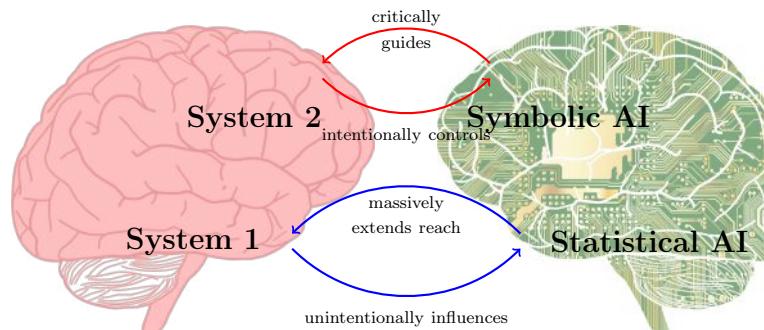
©: Michael Kohlhase

966



Thinking, Fast and Slow (two AI systems)

- ▷ System 1 and Statistical AI interface well. System 2 and Statistical AI interface well.



System 1

Statistical AI

- | | |
|--|---|
| <ul style="list-style-type: none"> ▷ low attention level ▷ short term desires ▷ little to no reflection ▷ microdecisions ▷ unintended influence | <ul style="list-style-type: none"> ▷ low transparency ▷ low interactivity ▷ low accountability ▷ rudimentary theory of mind |
|--|---|

System 2

- ▷ high attention level
- ▷ stable convictions
- ▷ high level of reflection
- ▷ macro-decisions

Symbolic AI

- ▷ high transparency
- ▷ high interactivity
- ▷ high accountability
- ▷ advanced theory of mind



©: Michael Kohlhase

967



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)
- ▷ 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



©: Michael Kohlhase

968



Topics of AI-2 (Summer Semester)

- ▷ Uncertain Knowledge and Reasoning
 - ▷ Uncertainty
 - ▷ Probabilistic Reasoning
 - ▷ Making Decisions in Episodic Environments
 - ▷ Problem Solving in Sequential Environments
- ▷ Foundations of Machine Learning

- ▷ Learning from Observations
- ▷ Knowledge in Learning
- ▷ Statistical Learning Methods

- ▷ Communication (If there is time)
- ▷ Natural Language Processing
- ▷ Natural Language for Communication



Statistical AI: Adding Uncertainty and Learning

- ▷ Problem Solving under Uncertainty (non-observable environment, stochastic states)
 - ▷ **Framework:** Probabilistic Inference: Conditional Probabilities/Independence
 - ▷ **Intuition:** Reasoning in Belief Space instead of State Space!
 - ▷ **Implementation:** Bayesian Networks (exploit conditional independence)
 - ▷ **Extension:** Utilities and Decision Theory (for static/episodic environments)

- ▷ Problem Solving in Sequential Worlds:
 - ▷ **Framework:** Markov Processes, Transition Models
 - ▷ **Extension:** MDPs, POMDPs (+ utilities/decisions)
 - ▷ **Implementation:** Dynamic Bayesian Networks

- ▷ Machine Learning: adding optimization in changing environments (unsupervised)
 - ▷ **Framework:** Learning from Observations (positive/negative examples)
 - ▷ **Intuitions:** finding consistent/optimal hypotheses in a hypothesis space
 - ▷ **Problems:** consistency, expressivity, under/overfitting, computational/data resources.
 - ▷ Extensions
 - ▷ knowledge in learning (based on logical methods)
 - ▷ statistical learning(optimizing the probability distribution over hyperspace, learning BNs)

- ▷ Communication



Topics of AI-3 – not taught at FAU 😞

- ▷ Learning
- ▷ More Reinforcement Learning

- ▷ Communicating, Perceiving, and Acting
 - ▷ Probabilistic Language Processing
 - ▷ Natural Language for Communication
 - ▷ Perception
 - ▷ Robotics
 - ▷ Emotions, Sentiment Analysisaks



Part VIII

Excursions

Appendix A

Completeness of Calculi for Propositional Logic

The next step is to analyze the two calculi for completeness. For that we will first give ourselves a very powerful tool: the model “existence theorem”, which encapsulates the model-theoretic part of completeness theorems. With that, completeness proofs – which are quite tedious otherwise – become a breeze.¹³

EdN:13

A.1 Abstract Consistency and Model Existence

We will now come to an important tool in the theoretical study of reasoning calculi: the “abstract consistency”/“model existence” method. This method for analyzing calculi was developed by Jaakko Hintikka, Raymond Smullyan, and Peter Andrews in 1950-1970 as an encapsulation of similar constructions that were used in completeness arguments in the decades before. The basis for this method is Smullyan’s Observation [Smu63] that completeness proofs based on Hintikka sets only certain properties of consistency and that with little effort one can obtain a generalization “Smullyan’s Unifying Principle”.

The basic intuition for this method is the following: typically, a logical system $\mathcal{S} = \langle \mathcal{L}, \mathcal{K}, \models \rangle$ has multiple calculi, human-oriented ones like the natural deduction calculi and machine-oriented ones like the automated theorem proving calculi. All of these need to be analyzed for completeness (as a basic quality assurance measure).

A completeness proof for a calculus \mathcal{C} for \mathcal{S} typically comes in two parts: one analyzes \mathcal{C} -consistency (sets that cannot be refuted in \mathcal{C}), and the other construct \mathcal{K} -models for \mathcal{C} -consistent sets.

In this situation the “abstract consistency”/“model existence” method encapsulates the model construction process into a meta-theorem: the “model existence” theorem. This provides a set of syntactic (“abstract consistency”) conditions for calculi that are sufficient to construct models.

With the model existence theorem it suffices to show that \mathcal{C} -consistency is an abstract consistency property (a purely syntactic task that can be done by a \mathcal{C} -proof transformation argument) to obtain a completeness result for \mathcal{C} .

Model Existence (Overview)

▷ **Definition:** Abstract consistency

¹³EDNOTE: reference the theorems.

- ▷ **Definition:** Hintikka set (maximally abstract consistent)
- ▷ **Theorem:** Hintikka sets are satisfiable
- ▷ **Theorem:** If Φ is abstract consistent, then Φ can be extended to a Hintikka set.
- ▷ **Corollary:** If Φ is abstract consistent, then Φ is satisfiable
- ▷ **Application:** Let \mathcal{C} be a calculus, if Φ is \mathcal{C} -consistent, then Φ is abstract consistent.
- ▷ **Corollary:** \mathcal{C} is complete.



The proof of the model existence theorem goes via the notion of a Hintikka set, a set of formulae with very strong syntactic closure properties, which allow to read off models. Jaako Hintikka's original idea for completeness proofs was that for every complete calculus \mathcal{C} and every \mathcal{C} -consistent set one can induce a Hintikka set, from which a model can be constructed. This can be considered as a first model existence theorem. However, the process of obtaining a Hintikka set for a set \mathcal{C} -consistent set Φ of sentences usually involves complicated calculus-dependent constructions.

In this situation, Raymond Smullyan was able to formulate the sufficient conditions for the existence of Hintikka sets in the form of "abstract consistency properties" by isolating the calculus-independent parts of the Hintikka set construction. His technique allows to reformulate Hintikka sets as maximal elements of abstract consistency classes and interpret the Hintikka set construction as a maximizing limit process.

To carry out the "model-existence"/"abstract consistency" method, we will first have to look at the notion of consistency.

Consistency and refutability are very important notions when studying the completeness for calculi; they form syntactic counterparts of satisfiability.

Consistency

- ▷ Let \mathcal{C} be a calculus
- ▷ **Definition A.1.1** Φ is called \mathcal{C} -refutable, if there is a formula \mathbf{B} , such that $\Phi \vdash_{\mathcal{C}} \mathbf{B}$ and $\Phi \vdash_{\mathcal{C}} \neg \mathbf{B}$.
- ▷ **Definition A.1.2** We call a pair \mathbf{A} and $\neg \mathbf{A}$ a **contradiction**.
- ▷ So a set Φ is \mathcal{C} -refutable, if \mathcal{C} can derive a contradiction from it.
- ▷ **Definition A.1.3** Φ is called \mathcal{C} -consistent, iff there is a formula \mathbf{B} , that is not derivable from Φ in \mathcal{C} .
- ▷ **Definition A.1.4** We call a calculus \mathcal{C} **reasonable**, iff implication elimination and conjunction introduction are admissible in \mathcal{C} and $\mathbf{A} \wedge \neg \mathbf{A} \Rightarrow \mathbf{B}$ is a \mathcal{C} -theorem.
- ▷ **Theorem A.1.5** \mathcal{C} -inconsistency and \mathcal{C} -refutability coincide for reasonable calculi.



It is very important to distinguish the syntactic \mathcal{C} -refutability and \mathcal{C} -consistency from satisfiability,

which is a property of formulae that is at the heart of semantics. Note that the former specify the calculus (a syntactic device) while the latter does not. In fact we should actually say \mathcal{S} -satisfiability, where $\mathcal{S} = \langle \mathcal{L}, \mathcal{K}, \models \rangle$ is the current logical system.

Even the word “contradiction” has a syntactical flavor to it, it translates to “saying against each other” from its latin root.

Abstract Consistency

- ▷ **Definition A.1.6** Let ∇ be a family of sets. We call ∇ **closed under subset**, iff for each $\Phi \in \nabla$, all subsets $\Psi \subseteq \Phi$ are elements of ∇ .
- ▷ **Notation A.1.7** We will use $\Phi * \mathbf{A}$ for $\Phi \cup \{\mathbf{A}\}$.
- ▷ **Definition A.1.8** A family ∇ of sets of propositional formulae is called an **abstract consistency class**, iff it is closed under subsets, and for each $\Phi \in \nabla$
 - $\nabla_c)$ $P \notin \Phi$ or $\neg P \notin \Phi$ for $P \in \mathcal{V}_o$
 - $\nabla_{\neg})$ $\neg \neg \mathbf{A} \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$
 - $\nabla_V)$ $(\mathbf{A} \vee \mathbf{B}) \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$ or $\Phi * \mathbf{B} \in \nabla$
 - $\nabla_{\wedge})$ $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$ implies $(\Phi \cup \{\neg \mathbf{A}, \neg \mathbf{B}\}) \in \nabla$
- ▷ **Example A.1.9** The empty set is an abstract consistency class
- ▷ **Example A.1.10** The set $\{\emptyset, \{Q\}, \{P \vee Q\}, \{P \vee Q, Q\}\}$ is an abstract consistency class
- ▷ **Example A.1.11** The family of satisfiable sets is an abstract consistency class.



So a family of sets (we call it a family, so that we do not have to say “set of sets” and we can distinguish the levels) is an abstract consistency class, iff it fulfills five simple conditions, of which the last three are closure conditions.

Think of an abstract consistency class as a family of “consistent” sets (e.g. \mathcal{C} -consistent for some calculus \mathcal{C}), then the properties make perfect sense: They are naturally closed under subsets — if we cannot derive a contradiction from a large set, we certainly cannot from a subset, furthermore,

- $\nabla_c)$ If both $P \in \Phi$ and $\neg P \in \Phi$, then Φ cannot be “consistent”.
- $\nabla_{\neg})$ If we cannot derive a contradiction from Φ with $\neg \neg \mathbf{A} \in \Phi$ then we cannot from $\Phi * \mathbf{A}$, since they are logically equivalent.

The other two conditions are motivated similarly.

Compact Collections

- ▷ **Definition A.1.12** We call a collection ∇ of sets **compact**, iff for any set Φ we have $\Phi \in \nabla$, iff $\Psi \in \nabla$ for every finite subset Ψ of Φ .
- ▷ **Lemma A.1.13** If ∇ is compact, then ∇ is closed under subsets.
- ▷ **Proof:**
- P.1** Suppose $S \subseteq T$ and $T \in \nabla$.

P.2 Every finite subset A of S is a finite subset of T .

P.3 As ∇ is compact, we know that $A \in \nabla$.

P.4 Thus $S \in \nabla$. □



The property of being closed under subsets is a “downwards-oriented” property: We go from large sets to small sets, compactness (the interesting direction anyways) is also an “upwards-oriented” property. We can go from small (finite) sets to large (infinite) sets. The main application for the compactness condition will be to show that infinite sets of formulae are in a family ∇ by testing all their finite subsets (which is much simpler).

We will carry out the proof here, since it gives us practice in dealing with the abstract consistency properties.

We now come to a very technical condition that will allow us to carry out a limit construction in the Hintikka set extension argument later.

Compact Collections

▷ **Definition A.1.14** We call a collection ∇ of sets **compact**, iff for any set Φ we have

$\Phi \in \nabla$, iff $\Psi \in \nabla$ for every finite subset Ψ of Φ .

▷ **Lemma A.1.15** *If ∇ is compact, then ∇ is closed under subsets.*

▷ **Proof:**

P.1 Suppose $S \subseteq T$ and $T \in \nabla$.

P.2 Every finite subset A of S is a finite subset of T .

P.3 As ∇ is compact, we know that $A \in \nabla$.

P.4 Thus $S \in \nabla$. □



The property of being closed under subsets is a “downwards-oriented” property: We go from large sets to small sets, compactness (the interesting direction anyways) is also an “upwards-oriented” property. We can go from small (finite) sets to large (infinite) sets. The main application for the compactness condition will be to show that infinite sets of formulae are in a family ∇ by testing all their finite subsets (which is much simpler).

The main result here is that abstract consistency classes can be extended to compact ones. The proof is quite tedious, but relatively straightforward. It allows us to assume that all abstract consistency classes are compact in the first place (otherwise we pass to the compact extension).

Compact Abstract Consistency Classes

▷ **Lemma A.1.16** *Any abstract consistency class can be extended to a compact one.*

▷ **Proof:**

P.1 We choose $\nabla' := \{\Phi \subseteq \text{wff}_o(\mathcal{V}_o) \mid \text{every finite subset of } \Phi \text{ is in } \nabla\}$.

P.2 Now suppose that $\Phi \in \nabla$. ∇ is closed under subsets, so every finite subset of Φ is in ∇ and thus $\Phi \in \nabla'$. Hence $\nabla \subseteq \nabla'$.

P.3 Next let us show that each ∇' is compact.

P.3.1 Suppose $\Phi \in \nabla'$ and Ψ is an arbitrary finite subset of Φ .

P.3.2 By definition of ∇' all finite subsets of Φ are in ∇ and therefore $\Psi \in \nabla$.

P.3.3 Thus all finite subsets of Φ are in ∇' whenever Φ is in ∇' .

P.3.4 On the other hand, suppose all finite subsets of Φ are in ∇' .

P.3.5 Then by the definition of ∇' the finite subsets of Φ are also in ∇ , so $\Phi \in \nabla$. Thus ∇' is compact.

P.4 Note that ∇' is closed under subsets by the Lemma above.

P.5 Now we show that if ∇ satisfies ∇_* , then ∇' satisfies ∇_* .

P.5.1 To show ∇_c , let $\Phi \in \nabla'$ and suppose there is an atom \mathbf{A} , such that $\{\mathbf{A}, \neg \mathbf{A}\} \subseteq \Phi$. Then $\{\mathbf{A}, \neg \mathbf{A}\} \in \nabla$ contradicting ∇_c .

P.5.2 To show ∇_{\neg} , let $\Phi \in \nabla'$ and $\neg \neg \mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \nabla'$.

P.5.2.1 Let Ψ be any finite subset of $\Phi * \mathbf{A}$, and $\Theta := (\Psi \setminus \{\mathbf{A}\}) * \neg \neg \mathbf{A}$.

P.5.2.2 Θ is a finite subset of Φ , so $\Theta \in \nabla$.

P.5.2.3 Since ∇ is an abstract consistency class and $\neg \neg \mathbf{A} \in \Theta$, we get $\Theta * \mathbf{A} \in \nabla$ by ∇_{\neg} .

P.5.2.4 We know that $\Psi \subseteq \Theta * \mathbf{A}$ and ∇ is closed under subsets, so $\Psi \in \nabla$.

P.5.2.5 Thus every finite subset Ψ of $\Phi * \mathbf{A}$ is in ∇ and therefore by definition $\Phi * \mathbf{A} \in \nabla'$.

P.5.3 the other cases are analogous to ∇_{\neg} . □



Hintikka sets are sets of sentences with very strong analytic closure conditions. These are motivated as maximally consistent sets i.e. sets that already contain everything that can be consistently added to them.

∇ -Hintikka Set

▷ **Definition A.1.17** Let ∇ be an abstract consistency class, then we call a set $\mathcal{H} \in \nabla$ a **Hintikka Set**, iff \mathcal{H} is maximal in ∇ , i.e. for all \mathbf{A} with $\mathcal{H} * \mathbf{A} \in \nabla$ we already have $\mathbf{A} \in \mathcal{H}$.

▷ **Theorem A.1.18 (Hintikka Properties)** *Let ∇ be an abstract consistency class and \mathcal{H} be a ∇ -Hintikka set, then*

- \mathcal{H}_c) For all $\mathbf{A} \in \text{wff}_o(\mathcal{V}_o)$ we have $\mathbf{A} \notin \mathcal{H}$ or $\neg \mathbf{A} \notin \mathcal{H}$
- \mathcal{H}_{\neg}) If $\neg \neg \mathbf{A} \in \mathcal{H}$ then $\mathbf{A} \in \mathcal{H}$
- \mathcal{H}_{\vee}) If $(\mathbf{A} \vee \mathbf{B}) \in \mathcal{H}$ then $\mathbf{A} \in \mathcal{H}$ or $\mathbf{B} \in \mathcal{H}$
- \mathcal{H}_{\wedge}) If $\neg(\mathbf{A} \vee \mathbf{B}) \in \mathcal{H}$ then $\neg \mathbf{A}, \neg \mathbf{B} \in \mathcal{H}$

Proof:

▷ **P.1** We prove the properties in turn

P.1.1 \mathcal{H}_c : by induction on the structure of \mathbf{A}

P.1.1.1.1 $\mathbf{A} \in \mathcal{V}_o$: Then $\mathbf{A} \notin \mathcal{H}$ or $\neg \mathbf{A} \notin \mathcal{H}$ by ∇_c .

P.1.1.1.2 $\mathbf{A} = \neg \mathbf{B}$:

P.1.1.1.2.1 Let us assume that $\neg \mathbf{B} \in \mathcal{H}$ and $\neg \neg \mathbf{B} \in \mathcal{H}$,

P.1.1.1.2.2 then $\mathcal{H} * \mathbf{B} \in \nabla$ by ∇_{\neg} , and therefore $\mathbf{B} \in \mathcal{H}$ by maximality.

P.1.1.1.2.3 So both \mathbf{B} and $\neg \mathbf{B}$ are in \mathcal{H} , which contradicts the inductive hypothesis. \square

P.1.1.1.3 $\mathbf{A} = \mathbf{B} \vee \mathbf{C}$: similar to the previous case:

\square

\square

P.1.2 We prove \mathcal{H}_{\neg} by maximality of \mathcal{H} in ∇ :

P.1.2.1 If $\neg \neg \mathbf{A} \in \mathcal{H}$, then $\mathcal{H} * \mathbf{A} \in \nabla$ by ∇_{\neg} .

P.1.2.2 The maximality of \mathcal{H} now gives us that $\mathbf{A} \in \mathcal{H}$. \square

P.1.3 other \mathcal{H}_* are similar:



The following theorem is one of the main results in the “abstract consistency”/“model existence” method. For any abstract consistent set Φ it allows us to construct a Hintikka set \mathcal{H} with $\Phi \in \mathcal{H}$.

Extension Theorem

▷ **Theorem A.1.19** If ∇ is an abstract consistency class and $\Phi \in \nabla$, then there is a ∇ -Hintikka set \mathcal{H} with $\Phi \subseteq \mathcal{H}$.

▷ **Proof:**

P.1 Wlog. we assume that ∇ is compact (otherwise pass to compact extension)

P.2 We choose an enumeration $\mathbf{A}^1, \mathbf{A}^2, \dots$ of the set $wff_o(\mathcal{V}_o)$

P.3 and construct a sequence of sets H^i with $H^0 := \Phi$ and

$$H^{n+1} := \begin{cases} H^n & \text{if } H^n * \mathbf{A}^n \notin \nabla \\ H^n * \mathbf{A}^n & \text{if } H^n * \mathbf{A}^n \in \nabla \end{cases}$$

P.4 Note that all $H^i \in \nabla$, choose $\mathcal{H} := \bigcup_{i \in \mathbb{N}} H^i$

P.5 $\Psi \subseteq \mathcal{H}$ finite implies there is a $j \in \mathbb{N}$ such that $\Psi \subseteq H^j$,

P.6 so $\Psi \in \nabla$ as ∇ closed under subsets and $\mathcal{H} \in \nabla$ as ∇ is compact.

P.7 Let $\mathcal{H} * \mathbf{B} \in \nabla$, then there is a $j \in \mathbb{N}$ with $\mathbf{B} = \mathbf{A}^j$, so that $\mathbf{B} \in H^{j+1}$ and $H^{j+1} \subseteq \mathcal{H}$

P.8 Thus \mathcal{H} is ∇ -maximal \square



Note that the construction in the proof above is non-trivial in two respects. First, the limit construction for \mathcal{H} is not executed in our original abstract consistency class ∇ , but in a suitably extended one to make it compact — the original would not have contained \mathcal{H} in general. Second, the set \mathcal{H} is not unique for Φ , but depends on the choice of the enumeration of $wff_o(\mathcal{V}_o)$. If we pick a different enumeration, we will end up with a different \mathcal{H} . Say if \mathbf{A} and $\neg \mathbf{A}$ are both

∇ -consistent¹⁴ with Φ , then depending on which one is first in the enumeration \mathcal{H} , will contain EdN:14 that one; with all the consequences for subsequent choices in the construction process.

Valuation

- ▷ **Definition A.1.20** A function $\nu: \text{wff}_o(\mathcal{V}_o) \rightarrow \mathcal{D}_o$ is called a **valuation**, iff
 - ▷ $\nu(\neg A) = \top$, iff $\nu(A) = \perp$
 - ▷ $\nu(A \vee B) = \top$, iff $\nu(A) = \top$ or $\nu(B) = \top$
- ▷ **Lemma A.1.21** If $\nu: \text{wff}_o(\mathcal{V}_o) \rightarrow \mathcal{D}_o$ is a valuation and $\Phi \subseteq \text{wff}_o(\mathcal{V}_o)$ with $\nu(\Phi) = \{\top\}$, then Φ is satisfiable.
- ▷ **Proof Sketch:** $\nu|_{\mathcal{V}_o}: \mathcal{V}_o \rightarrow \mathcal{D}_o$ is a satisfying variable assignment. \square
- ▷ **Lemma A.1.22** If $\varphi: \mathcal{V}_o \rightarrow \mathcal{D}_o$ is a variable assignment, then $\mathcal{I}_\varphi: \text{wff}_o(\mathcal{V}_o) \rightarrow \mathcal{D}_o$ is a valuation.



©: Michael Kohlhase

980



Now, we only have to put the pieces together to obtain the model existence theorem we are after.

Model Existence

- ▷ **Lemma A.1.23 (Hintikka-Lemma)** If ∇ is an abstract consistency class and \mathcal{H} a ∇ -Hintikka set, then \mathcal{H} is satisfiable.
- ▷ **Proof:**
 - P.1 We define $\nu(A) := \top$, iff $A \in \mathcal{H}$
 - P.2 then ν is a valuation by the Hintikka properties
 - P.3 and thus $\nu|_{\mathcal{V}_o}$ is a satisfying assignment. \square
- ▷ **Theorem A.1.24 (Model Existence)** If ∇ is an abstract consistency class and $\Phi \in \nabla$, then Φ is satisfiable.
- ▷ **Proof:**
 - ▷ P.1 There is a ∇ -Hintikka set \mathcal{H} with $\Phi \subseteq \mathcal{H}$ (Extension Theorem)
 - We know that \mathcal{H} is satisfiable. (Hintikka-Lemma)
 - In particular, $\Phi \subseteq \mathcal{H}$ is satisfiable. \square



©: Michael Kohlhase

981



A.2 A Completeness Proof for Propositional Tableaux

With the model existence proof we have introduced in the last section, the completeness proof for first-order natural deduction is rather simple, we only have to check that Tableaux-consistency is an abstract consistency property.

¹⁴EDNOTE: introduce this above

We encapsulate all of the technical difficulties of the problem in a technical Lemma. From that, the completeness proof is just an application of the high-level theorems we have just proven.

P.2 P.3 Abstract Completeness for \mathcal{T}_0

▷ **Lemma A.2.1** $\{\Phi \mid \Phi^T \text{ has no closed Tableau}\}$ is an abstract consistency class.

▷ **Proof:** Let's call the set above ∇

P.1 We have to convince ourselves of the abstract consistency properties

P.1.1 ∇_c : $P, \neg P \in \Phi$ implies $P^F, P^T \in \Phi^T$. \square

P.1.2 ∇_\perp : Let $\neg \neg A \in \Phi$.

P.1.2.1 For the proof of the contrapositive we assume that $\Phi * A$ has a closed tableau \mathcal{T} and show that already Φ has one:

P.1.2.2 applying $\mathcal{T}_0 \neg$ twice allows to extend any tableau with $\neg \neg B^\alpha$ by B^α .

P.1.2.3 any path in \mathcal{T} that is closed with $\neg \neg A^\alpha$, can be closed by A^α . \square

P.1.3 ∇_v : Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ have closed tableaux

P.1.3.1 consider the tableaux:

$$\begin{array}{ccc} \Phi^T & \Phi^T & \Psi^T \\ A^T & B^T & A \vee B^T \\ Rest^1 & Rest^2 & \begin{array}{c|c} A^T & B^T \\ \hline Rest^1 & Rest^2 \end{array} \end{array}$$

\square

P.1.4 ∇_\wedge : suppose, $\neg(A \vee B) \in \Phi$ and $\Phi\{\neg A, \neg B\}$ have closed tableau \mathcal{T} .

P.1.4.1 We consider

$$\begin{array}{ccc} \Phi^T & \Psi^T \\ A^F & A \vee B^F \\ B^F & A^F \\ Rest & B^F \\ Rest & Rest \end{array}$$

where $\Phi = \Psi * \neg(A \vee B)$. \square

\square

\square



Observation: If we look at the completeness proof below, we see that the Lemma above is the only place where we had to deal with specific properties of the tableau calculus.

So if we want to prove completeness of any other calculus with respect to propositional logic, then we only need to prove an analogon to this lemma and can use the rest of the machinery we have already established “off the shelf”.

This is one great advantage of the “abstract consistency method”; the other is that the method can be extended transparently to other logics.

Completeness of \mathcal{T}_0

▷ **Corollary A.2.2** \mathcal{T}_0 is complete.

▷ **Proof:** by contradiction

P.1 We assume that $\mathbf{A} \in wff_o(\mathcal{V}_o)$ is valid, but there is no closed tableau for \mathbf{A}^F .

P.2 We have $\{\neg \mathbf{A}\} \in \nabla$ as $\neg \mathbf{A}^T = \mathbf{A}^F$.

P.3 so $\neg \mathbf{A}$ is satisfiable by the model existence theorem (which is applicable as ∇ is an abstract consistency class by our Lemma above)

P.4 this contradicts our assumption that \mathbf{A} is valid. \square



Appendix B

Completeness of Calculi for First-Order Logic

We will now analyze the first-order calculi for completeness. Just as in the case of the propositional calculi, we prove a model existence theorem for the first-order model theory and then use that for the completeness proofs¹⁵. The proof of the first-order model existence theorem is completely analogous to the propositional one; indeed, apart from the model construction itself, it is just an extension by a treatment for the first-order quantifiers.¹⁶

EdN:15

EdN:16

B.1 Abstract Consistency and Model Existence

We will now come to an important tool in the theoretical study of reasoning calculi: the “abstract consistency”/“model existence” method. This method for analyzing calculi was developed by Jaako Hintikka, Raymond Smullyan, and Peter Andrews in 1950-1970 as an encapsulation of similar constructions that were used in completeness arguments in the decades before. The basis for this method is Smullyan’s Observation [Smu63] that completeness proofs based on Hintikka sets only certain properties of consistency and that with little effort one can obtain a generalization “Smullyan’s Unifying Principle”.

The basic intuition for this method is the following: typically, a logical system $\mathcal{S} = \langle \mathcal{L}, \mathcal{K}, \models \rangle$ has multiple calculi, human-oriented ones like the natural deduction calculi and machine-oriented ones like the automated theorem proving calculi. All of these need to be analyzed for completeness (as a basic quality assurance measure).

A completeness proof for a calculus \mathcal{C} for \mathcal{S} typically comes in two parts: one analyzes \mathcal{C} -consistency (sets that cannot be refuted in \mathcal{C}), and the other construct \mathcal{K} -models for \mathcal{C} -consistent sets.

In this situation the “abstract consistency”/“model existence” method encapsulates the model construction process into a meta-theorem: the “model existence” theorem. This provides a set of syntactic (“abstract consistency”) conditions for calculi that are sufficient to construct models.

With the model existence theorem it suffices to show that \mathcal{C} -consistency is an abstract consistency property (a purely syntactic task that can be done by a \mathcal{C} -proof transformation argument) to obtain a completeness result for \mathcal{C} .

Model Existence (Overview)

▷ **Definition:** Abstract consistency

¹⁵EDNOTE: reference the theorems

¹⁶EDNOTE: MK: what about equality?

- ▷ **Definition:** Hintikka set (maximally abstract consistent)
- ▷ **Theorem:** Hintikka sets are satisfiable
- ▷ **Theorem:** If Φ is abstract consistent, then Φ can be extended to a Hintikka set.
- ▷ **Corollary:** If Φ is abstract consistent, then Φ is satisfiable
- ▷ **Application:** Let \mathcal{C} be a calculus, if Φ is \mathcal{C} -consistent, then Φ is abstract consistent.
- ▷ **Corollary:** \mathcal{C} is complete.



The proof of the model existence theorem goes via the notion of a Hintikka set, a set of formulae with very strong syntactic closure properties, which allow to read off models. Jaako Hintikka's original idea for completeness proofs was that for every complete calculus \mathcal{C} and every \mathcal{C} -consistent set one can induce a Hintikka set, from which a model can be constructed. This can be considered as a first model existence theorem. However, the process of obtaining a Hintikka set for a set \mathcal{C} -consistent set Φ of sentences usually involves complicated calculus-dependent constructions.

In this situation, Raymond Smullyan was able to formulate the sufficient conditions for the existence of Hintikka sets in the form of "abstract consistency properties" by isolating the calculus-independent parts of the Hintikka set construction. His technique allows to reformulate Hintikka sets as maximal elements of abstract consistency classes and interpret the Hintikka set construction as a maximizing limit process.

To carry out the "model-existence"/"abstract consistency" method, we will first have to look at the notion of consistency.

Consistency and refutability are very important notions when studying the completeness for calculi; they form syntactic counterparts of satisfiability.

Consistency

- ▷ Let \mathcal{C} be a calculus
- ▷ **Definition B.1.1** Φ is called \mathcal{C} -refutable, if there is a formula \mathbf{B} , such that $\Phi \vdash_{\mathcal{C}} \mathbf{B}$ and $\Phi \vdash_{\mathcal{C}} \neg \mathbf{B}$.
- ▷ **Definition B.1.2** We call a pair \mathbf{A} and $\neg \mathbf{A}$ a contradiction.
- ▷ So a set Φ is \mathcal{C} -refutable, if \mathcal{C} can derive a contradiction from it.
- ▷ **Definition B.1.3** Φ is called \mathcal{C} -consistent, iff there is a formula \mathbf{B} , that is not derivable from Φ in \mathcal{C} .
- ▷ **Definition B.1.4** We call a calculus \mathcal{C} reasonable, iff implication elimination and conjunction introduction are admissible in \mathcal{C} and $\mathbf{A} \wedge \neg \mathbf{A} \Rightarrow \mathbf{B}$ is a \mathcal{C} -theorem.
- ▷ **Theorem B.1.5** \mathcal{C} -inconsistency and \mathcal{C} -refutability coincide for reasonable calculi.



It is very important to distinguish the syntactic \mathcal{C} -refutability and \mathcal{C} -consistency from satisfiability,

which is a property of formulae that is at the heart of semantics. Note that the former specify the calculus (a syntactic device) while the latter does not. In fact we should actually say \mathcal{S} -satisfiability, where $\mathcal{S} = \langle \mathcal{L}, \mathcal{K}, \models \rangle$ is the current logical system.

Even the word “contradiction” has a syntactical flavor to it, it translates to “saying against each other” from its latin root.

The notion of an “abstract consistency class” provides the a calculus-independent notion of “consistency”: A set Φ of sentences is considered “consistent in an abstract sense”, iff it is a member of an abstract consistency class ∇ .

Abstract Consistency

- ▷ **Definition B.1.6** Let ∇ be a family of sets. We call ∇ closed under subsets, iff for each $\Phi \in \nabla$, all subsets $\Psi \subseteq \Phi$ are elements of ∇ .
- ▷ **Notation B.1.7** We will use $\Phi * \mathbf{A}$ for $\Phi \cup \{\mathbf{A}\}$.
- ▷ **Definition B.1.8** A family $\nabla \subseteq wff_o(\Sigma)$ of sets of formulae is called a (first-order) abstract consistency class, iff it is closed under subsets, and for each $\Phi \in \nabla$
 - ∇_c) $\mathbf{A} \notin \Phi$ or $\neg \mathbf{A} \notin \Phi$ for atomic $\mathbf{A} \in wff_o(\Sigma)$.
 - ∇_{\neg}) $\neg \neg \mathbf{A} \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$
 - ∇_{\wedge}) $(\mathbf{A} \wedge \mathbf{B}) \in \Phi$ implies $(\Phi \cup \{\mathbf{A}, \mathbf{B}\}) \in \nabla$
 - ∇_{\vee}) $\neg (\mathbf{A} \wedge \mathbf{B}) \in \Phi$ implies $\Phi * \neg \mathbf{A} \in \nabla$ or $\Phi * \neg \mathbf{B} \in \nabla$
 - ∇_{\forall}) If $(\forall X . \mathbf{A}) \in \Phi$, then $\Phi * [\mathbf{B}/X](\mathbf{A}) \in \nabla$ for each closed term \mathbf{B} .
 - ∇_{\exists}) If $\neg (\forall X . \mathbf{A}) \in \Phi$ and c is an individual constant that does not occur in Φ , then $\Phi * \neg [c/X](\mathbf{A}) \in \nabla$



The conditions are very natural: Take for instance ∇_c , it would be foolish to call a set Φ of sentences “consistent under a complete calculus”, if it contains an elementary contradiction. The next condition ∇_{\neg} says that if a set Φ that contains a sentence $\neg \neg \mathbf{A}$ is “consistent”, then we should be able to extend it by \mathbf{A} without losing this property; in other words, a complete calculus should be able to recognize \mathbf{A} and $\neg \neg \mathbf{A}$ to be equivalent.

We will carry out the proof here, since it gives us practice in dealing with the abstract consistency properties.

Actually we are after abstract consistency classes that have an even stronger property than just being closed under subsets. This will allow us to carry out a limit construction in the Hintikka set extension argument later.

Compact Collections

- ▷ **Definition B.1.9** We call a collection ∇ of sets compact, iff for any set Φ we have $\Phi \in \nabla$, iff $\Psi \in \nabla$ for every finite subset Ψ of Φ .
- ▷ **Lemma B.1.10** If ∇ is compact, then ∇ is closed under subsets.
- ▷ **Proof:**
- P.1 Suppose $S \subseteq T$ and $T \in \nabla$.

P.2 Every finite subset A of S is a finite subset of T .

P.3 As ∇ is compact, we know that $A \in \nabla$.

P.4 Thus $S \in \nabla$. □



The property of being closed under subsets is a “downwards-oriented” property: We go from large sets to small sets, compactness (the interesting direction anyways) is also an “upwards-oriented” property. We can go from small (finite) sets to large (infinite) sets. The main application for the compactness condition will be to show that infinite sets of formulae are in a family ∇ by testing all their finite subsets (which is much simpler).

The main result here is that abstract consistency classes can be extended to compact ones. The proof is quite tedious, but relatively straightforward. It allows us to assume that all abstract consistency classes are compact in the first place (otherwise we pass to the compact extension).

Compact Abstract Consistency Classes

▷ **Lemma B.1.11** *Any first-order abstract consistency class can be extended to a compact one.*

▷ **Proof:**

P.1 We choose $\nabla' := \{\Phi \subseteq \text{cwff}_o(\Sigma_\ell) \mid \text{every finite subset of } \Phi \text{ is in } \nabla\}$.

P.2 Now suppose that $\Phi \in \nabla$. ∇ is closed under subsets, so every finite subset of Φ is in ∇ and thus $\Phi \in \nabla'$. Hence $\nabla \subseteq \nabla'$.

P.3 Let us now show that each ∇' is compact.

P.3.1 Suppose $\Phi \in \nabla'$ and Ψ is an arbitrary finite subset of Φ .

P.3.2 By definition of ∇' all finite subsets of Φ are in ∇ and therefore $\Psi \in \nabla$.

P.3.3 Thus all finite subsets of Φ are in ∇' whenever Φ is in ∇' .

P.3.4 On the other hand, suppose all finite subsets of Φ are in ∇' .

P.3.5 Then by the definition of ∇' the finite subsets of Φ are also in ∇ , so $\Phi \in \nabla$. Thus ∇' is compact.

P.4 Note that ∇' is closed under subsets by the Lemma above.

P.5 Next we show that if ∇ satisfies ∇_* , then ∇' satisfies ∇_* .

P.5.1 To show ∇_c , let $\Phi \in \nabla'$ and suppose there is an atom \mathbf{A} , such that $\{\mathbf{A}, \neg \mathbf{A}\} \subseteq \Phi$. Then $\{\mathbf{A}, \neg \mathbf{A}\} \in \nabla$ contradicting ∇_c .

P.5.2 To show ∇_{\neg} , let $\Phi \in \nabla'$ and $\neg \neg \mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \nabla'$.

P.5.2.1 Let Ψ be any finite subset of $\Phi * \mathbf{A}$, and $\Theta := (\Psi \setminus \{\mathbf{A}\}) * \neg \neg \mathbf{A}$.

P.5.2.2 Θ is a finite subset of Φ , so $\Theta \in \nabla$.

P.5.2.3 Since ∇ is an abstract consistency class and $\neg \neg \mathbf{A} \in \Theta$, we get $\Theta * \mathbf{A} \in \nabla$ by ∇_{\neg} .

P.5.2.4 We know that $\Psi \subseteq \Theta * \mathbf{A}$ and ∇ is closed under subsets, so $\Psi \in \nabla$.

P.5.2.5 Thus every finite subset Ψ of $\Phi * \mathbf{A}$ is in ∇ and therefore by definition $\Phi * \mathbf{A} \in \nabla'$.

P.5.3 the other cases are analogous to ∇_{\neg} . □



Hintikka sets are sets of sentences with very strong analytic closure conditions. These are motivated as maximally consistent sets i.e. sets that already contain everything that can be consistently added to them.

∇ -Hintikka Set

▷ **Definition B.1.12** Let ∇ be an abstract consistency class, then we call a set $\mathcal{H} \in \nabla$ a **∇ -Hintikka Set**, iff \mathcal{H} is maximal in ∇ , i.e. for all \mathbf{A} with $\mathcal{H} * \mathbf{A} \in \nabla$ we already have $\mathbf{A} \in \mathcal{H}$.

▷ **Theorem B.1.13 (Hintikka Properties)** *Let ∇ be an abstract consistency class and \mathcal{H} be a ∇ -Hintikka set, then*

- \mathcal{H}_c) For all $\mathbf{A} \in wff_o(\Sigma)$ we have $\mathbf{A} \notin \mathcal{H}$ or $\neg \mathbf{A} \notin \mathcal{H}$.
- \mathcal{H}_{\neg}) If $\neg \neg \mathbf{A} \in \mathcal{H}$ then $\mathbf{A} \in \mathcal{H}$.
- \mathcal{H}_{\wedge}) If $(\mathbf{A} \wedge \mathbf{B}) \in \mathcal{H}$ then $\mathbf{A}, \mathbf{B} \in \mathcal{H}$.
- \mathcal{H}_{\vee}) If $\neg(\mathbf{A} \wedge \mathbf{B}) \in \mathcal{H}$ then $\neg \mathbf{A} \in \mathcal{H}$ or $\neg \mathbf{B} \in \mathcal{H}$.
- \mathcal{H}_{\forall}) If $(\forall X . \mathbf{A}) \in \mathcal{H}$, then $[\mathbf{B}/X](\mathbf{A}) \in \mathcal{H}$ for each closed term \mathbf{B} .
- \mathcal{H}_{\exists}) If $\neg(\forall X . \mathbf{A}) \in \mathcal{H}$ then $\neg [\mathbf{B}/X](\mathbf{A}) \in \mathcal{H}$ for some term closed term \mathbf{B} .

Proof:

▷ **P.1** We prove the properties in turn

\mathcal{H}_c goes by induction on the structure of \mathbf{A}

IP21 A atomic: Then $\mathbf{A} \notin \mathcal{H}$ or $\neg \mathbf{A} \notin \mathcal{H}$ by ∇_c .

P.2.2 $\mathbf{A} = \neg \mathbf{B}$:

P.2.2.1 Let us assume that $\neg \mathbf{B} \in \mathcal{H}$ and $\neg \neg \mathbf{B} \in \mathcal{H}$,

P.2.2.2 then $\mathcal{H} * \mathbf{B} \in \nabla$ by ∇_{\neg} , and therefore $\mathbf{B} \in \mathcal{H}$ by maximality.

P.2.2.3 So $\{\mathbf{B}, \neg \mathbf{B}\} \subseteq \mathcal{H}$, which contradicts the inductive hypothesis. □

P.2.3 $\mathbf{A} = \mathbf{B} \vee \mathbf{C}$: similar to the previous case

We prove \mathcal{H}_{\neg} by maximality of \mathcal{H} in ∇ .

IP31 If $\neg \neg \mathbf{A} \in \mathcal{H}$, then $\mathcal{H} * \mathbf{A} \in \nabla$ by ∇_{\neg} .

P.3.2 The maximality of \mathcal{H} now gives us that $\mathbf{A} \in \mathcal{H}$.

The other \mathcal{H}_* are similar □



The following theorem is one of the main results in the “abstract consistency”/“model existence” method. For any abstract consistent set Φ it allows us to construct a Hintikka set \mathcal{H} with $\Phi \in \mathcal{H}$.

P.4 Extension Theorem

▷ **Theorem B.1.14** *If ∇ is an abstract consistency class and $\Phi \in \nabla$ finite, then*

there is a ∇ -Hintikka set \mathcal{H} with $\Phi \subseteq \mathcal{H}$.

▷ Proof: Wlog. assume that ∇ compact (else use compact extension)

P.1 Choose an enumeration $\mathbf{A}^1, \mathbf{A}^2, \dots$ of $cwff_o(\Sigma_\iota)$ and c^1, c^2, \dots of Σ_0^{sk} .

P.2 and construct a sequence of sets H^i with $H^0 := \Phi$ and

$$H^{n+1} := \begin{cases} H^n & \text{if } H^n * \mathbf{A}^n \notin \nabla \\ H^n \cup \{\mathbf{A}^n, \neg[c^n/X](\mathbf{B})\} & \text{if } H^n * \mathbf{A}^n \in \nabla \text{ and } \mathbf{A}^n = \neg(\forall X . \mathbf{B}) \\ H^n * \mathbf{A}^n & \text{else} \end{cases}$$

P.3 Note that all $H^i \in \nabla$, choose $\mathcal{H} := \bigcup_{i \in \mathbb{N}} H^i$

P.4 $\Psi \subseteq \mathcal{H}$ finite implies there is a $j \in \mathbb{N}$ such that $\Psi \subseteq H^j$,

P.5 so $\Psi \in \nabla$ as ∇ closed under subsets and $\mathcal{H} \in \nabla$ as ∇ is compact.

P.6 Let $\mathcal{H} * \mathbf{B} \in \nabla$, then there is a $j \in \mathbb{N}$ with $\mathbf{B} = \mathbf{A}^j$, so that $\mathbf{B} \in H^{j+1}$ and $H^{j+1} \subseteq \mathcal{H}$

P.7 Thus \mathcal{H} is ∇ -maximal \square



Note that the construction in the proof above is non-trivial in two respects. First, the limit construction for \mathcal{H} is not executed in our original abstract consistency class ∇ , but in a suitably extended one to make it compact — the original would not have contained \mathcal{H} in general. Second, the set \mathcal{H} is not unique for Φ , but depends on the choice of the enumeration of $cwff_o(\Sigma_\iota)$. If we pick a different enumeration, we will end up with a different \mathcal{H} . Say if \mathbf{A} and $\neg\mathbf{A}$ are both ∇ -consistent¹⁷ with Φ , then depending on which one is first in the enumeration \mathcal{H} , will contain that one; with all the consequences for subsequent choices in the construction process.

EdN:17

Valuation

▷ **Definition B.1.15** A function $\nu: cwff_o(\Sigma_\iota) \rightarrow \mathcal{D}_o$ is called a (first-order) valuation, iff

- ▷ $\nu(\neg \mathbf{A}) = \top$, iff $\nu(\mathbf{A}) = \perp$
- ▷ $\nu(\mathbf{A} \wedge \mathbf{B}) = \top$, iff $\nu(\mathbf{A}) = \top$ and $\nu(\mathbf{B}) = \top$
- ▷ $\nu(\forall X . \mathbf{A}) = \top$, iff $\nu([\mathbf{B}/X](\mathbf{A})) = \top$ for all closed terms \mathbf{B} .

▷ **Lemma B.1.16** If $\varphi: \mathcal{V}_\iota \rightarrow \mathcal{D}$ is a variable assignment, then $\mathcal{I}_\varphi: cwff_o(\Sigma_\iota) \rightarrow \mathcal{D}_o$ is a valuation.

▷ **Proof Sketch:** Immediate from the definitions \square



Thus a valuation is a weaker notion of evaluation in first-order logic; the other direction is also true, even though the proof of this result is much more involved: The existence of a first-order valuation that makes a set of sentences true entails the existence of a model that satisfies it.¹⁸

¹⁷ EdNote: introduce this above

¹⁸ EdNote: I think that we only get a semivaluation, look it up in Andrews.

Valuation and Satisfiability

▷ **Lemma B.1.17** If $\nu: \text{cwoff}_o(\Sigma_\ell) \rightarrow \mathcal{D}_o$ is a valuation and $\Phi \subseteq \text{cwoff}_o(\Sigma_\ell)$ with $\nu(\Phi) = \{\top\}$, then Φ is satisfiable.

▷ **Proof:** We construct a model for Φ .

P.1 Let $\mathcal{D}_\ell := \text{cwoff}_\ell(\Sigma_\ell)$, and

- ▷ $\mathcal{I}(f): \mathcal{D}_\ell^k \rightarrow \mathcal{D}_\ell; \langle \mathbf{A}_1, \dots, \mathbf{A}_k \rangle \mapsto f(\mathbf{A}_1, \dots, \mathbf{A}_k)$ for $f \in \Sigma^f$
- ▷ $\mathcal{I}(p): \mathcal{D}_\ell^k \rightarrow \mathcal{D}_o; \langle \mathbf{A}_1, \dots, \mathbf{A}_k \rangle \mapsto \nu(p(\mathbf{A}_1, \dots, \mathbf{A}_n))$ for $p \in \Sigma^p$.

P.2 Then variable assignments into \mathcal{D}_ℓ are ground substitutions.

P.3 We show $\mathcal{I}_\varphi(\mathbf{A}) = \varphi(\mathbf{A})$ for $\mathbf{A} \in \text{wff}_\ell(\Sigma_\ell)$ by induction on \mathbf{A}

P.3.1 $\mathbf{A} = X$: then $\mathcal{I}_\varphi(\mathbf{A}) = \varphi(X)$ by definition.

P.3.2 $\mathbf{A} = f(\mathbf{A}_1, \dots, \mathbf{A}_n)$: then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1), \dots, \mathcal{I}_\varphi(\mathbf{A}_n)) = \mathcal{I}(f)(\varphi(\mathbf{A}_1), \dots, \varphi(\mathbf{A}_n)) = f(\varphi(\mathbf{A}_1), \dots, \varphi(\mathbf{A}_n)) = \varphi(f(\mathbf{A}_1, \dots, \mathbf{A}_n)) = \varphi(\mathbf{A})$

P.4 We show $\mathcal{I}_\varphi(\mathbf{A}) = \nu(\varphi(\mathbf{A}))$ for $\mathbf{A} \in \text{wff}_o(\Sigma)$ by induction on \mathbf{A}

P.4.1 $\mathbf{A} = p(\mathbf{A}_1, \dots, \mathbf{A}_n)$: then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}(p)(\mathcal{I}_\varphi(\mathbf{A}_1), \dots, \mathcal{I}_\varphi(\mathbf{A}_n)) = \mathcal{I}(p)(\varphi(\mathbf{A}_1), \dots, \varphi(\mathbf{A}_n)) = \nu(p(\varphi(\mathbf{A}_1), \dots, \varphi(\mathbf{A}_n))) = \nu(\varphi(p(\mathbf{A}_1, \dots, \mathbf{A}_n))) = \nu(\varphi(\mathbf{A}))$

P.4.2 $\mathbf{A} = \neg \mathbf{B}$: then $\mathcal{I}_\varphi(\mathbf{A}) = \top$, iff $\mathcal{I}_\varphi(\mathbf{B}) = \nu(\varphi(\mathbf{B})) = \mathbb{F}$, iff $\nu(\varphi(\mathbf{A})) = \top$.

P.4.3 $\mathbf{A} = \mathbf{B} \wedge \mathbf{C}$: similar

P.4.4 $\mathbf{A} = \forall X. \mathbf{B}$: then $\mathcal{I}_\varphi(\mathbf{A}) = \top$, iff $\mathcal{I}_\psi(\mathbf{B}) = \nu(\psi(\mathbf{B})) = \top$, for all $\mathbf{C} \in \mathcal{D}_\ell$, where $\psi = \varphi, [\mathbf{C}/X]$. This is the case, iff $\nu(\varphi(\mathbf{A})) = \top$.

P.5 Thus $\mathcal{I}_\varphi(\mathbf{A}) = \nu(\varphi(\mathbf{A})) = \nu(\mathbf{A}) = \top$ for all $\mathbf{A} \in \Phi$.

P.6 Hence $\mathcal{M} \models \mathbf{A}$ for $\mathcal{M} := \langle \mathcal{D}_\ell, \mathcal{I} \rangle$. \square



Now, we only have to put the pieces together to obtain the model existence theorem we are after.

Model Existence

▷ **Theorem B.1.18 (Hintikka-Lemma)** If ∇ is an abstract consistency class and \mathcal{H} a ∇ -Hintikka set, then \mathcal{H} is satisfiable.

▷ **Proof:**

P.1 we define $\nu(\mathbf{A}) := \top$, iff $\mathbf{A} \in \mathcal{H}$,

P.2 then ν is a valuation by the Hintikka set properties.

P.3 We have $\nu(\mathcal{H}) = \{\top\}$, so \mathcal{H} is satisfiable. \square

▷ **Theorem B.1.19 (Model Existence)** If ∇ is an abstract consistency class and $\Phi \in \nabla$, then Φ is satisfiable.

Proof:

▷ **P.1** There is a ∇ -Hintikka set \mathcal{H} with $\Phi \subseteq \mathcal{H}$

(Extension Theorem)

We know that \mathcal{H} is satisfiable.

(Hintikka-Lemma)

In particular, $\Phi \subseteq \mathcal{H}$ is satisfiable.

□



B.2 A Completeness Proof for First-Order ND

With the model existence proof we have introduced in the last section, the completeness proof for first-order natural deduction is rather simple, we only have to check that ND-consistency is an abstract consistency property.

P.2 P.3 Consistency, Refutability and Abstract Consistency

▷ **Theorem B.2.1 (Non-Refutability is an Abstract Consistency Property)**
 $\Gamma := \{\Phi \subseteq \text{wff}_o(\Sigma) \mid \Phi \text{ not } \mathcal{ND}^1\text{-refutable}\}$ is an abstract consistency class.

▷ **Proof:** We check the properties of an ACC

P.1 If Φ is non-refutable, then any subset is as well, so Γ is closed under subsets.

P.2 We show the abstract consistency conditions ∇_* for $\Phi \in \Gamma$.

P.2.1 ∇_c : We have to show that $\mathbf{A} \notin \Phi$ or $\neg \mathbf{A} \notin \Phi$ for atomic $\mathbf{A} \in \text{wff}_o(\Sigma)$.

P.2.1.2 Equivalently, we show the contrapositive: If $\{\mathbf{A}, \neg \mathbf{A}\} \subseteq \Phi$, then $\Phi \notin \Gamma$.

P.2.1.3 So let $\{\mathbf{A}, \neg \mathbf{A}\} \subseteq \Phi$, then Φ is \mathcal{ND}^1 -refutable by construction.

P.2.1.4 So $\Phi \notin \Gamma$. □

P.2.2 ∇_{\neg} : We show the contrapositive again

P.2.2.2 Let $\neg \neg \mathbf{A} \in \Phi$ and $\Phi * \mathbf{A} \notin \Gamma$

P.2.2.3 Then we have a refutation $\mathcal{D}: \Phi * \mathbf{A} \vdash_{\mathcal{ND}^1} F$

P.2.2.4 By prepending an application of $\neg E$ for $\neg \neg \mathbf{A}$ to \mathcal{D} , we obtain a refutation $\mathcal{D}': \Phi \vdash_{\mathcal{ND}^1} F$.

P.2.2.5 Thus $\Phi \notin \Gamma$. □

P.2.3 other ∇_* similar:



This directly yields two important results that we will use for the completeness analysis.

Henkin's Theorem

▷ **Corollary B.2.2 (Henkin's Theorem)** Every \mathcal{ND}^1 -consistent set of sentences has a model.

▷ **Proof:**

P.1 Let Φ be a \mathcal{ND}^1 -consistent set of sentences.

P.2 The class of sets of \mathcal{ND}^1 -consistent propositions constitute an abstract consistency class

P.3 Thus the model existence theorem guarantees a model for Φ . \square

▷ **Corollary B.2.3 (Löwenheim&Skolem Theorem)** *Satisfiable set Φ of first-order sentences has a countable model.*

▷ **Proof Sketch:** The model we constructed is countable, since the set of ground terms is. \square



Now, the completeness result for first-order natural deduction is just a simple argument away. We also get a compactness theorem (almost) for free: logical systems with a complete calculus are always compact.

Completeness and Compactness

▷ **Theorem B.2.4 (Completeness Theorem for \mathcal{ND}^1)** *If $\Phi \models A$, then $\Phi \vdash_{\mathcal{ND}^1} A$.*

▷ **Proof:** We prove the result by playing with negations.

P.1 If A is valid in all models of Φ , then $\Phi * \neg A$ has no model

P.2 Thus $\Phi * \neg A$ is inconsistent by (the contrapositive of) Henkens Theorem.

P.3 So $\Phi \vdash_{\mathcal{ND}^1} \neg \neg A$ by $\neg I$ and thus $\Phi \vdash_{\mathcal{ND}^1} A$ by $\neg E$. \square

▷ **Theorem B.2.5 (Compactness Theorem for first-order logic)** *If $\Phi \models A$, then there is already a finite set $\Psi \subseteq \Phi$ with $\Psi \models A$.*

Proof: This is a direct consequence of the completeness theorem

▷ **P.1** We have $\Phi \models A$, iff $\Phi \vdash_{\mathcal{ND}^1} A$.

As a proof is a finite object, only a finite subset $\Psi \subseteq \Phi$ can appear as leaves in the proof. \square



B.3 Soundness and Completeness of First-Order Tableaux

The soundness of the first-order free-variable tableaux calculus can be established a simple induction over the size of the tableau.

P.2 Soundness of \mathcal{T}_1^f

▷ **Lemma B.3.1** *Tableau rules transform satisfiable tableaux into satisfiable ones.*

▷ **Proof:**

P.1 we examine the tableau rules in turn

P.1.1 propositional rules: as in propositional tableaux

P.1.2 $\mathcal{T}_1^f : \exists$: by Lemma B.3.3

P.1.3 $\mathcal{T}_1^f : \perp$: by Lemma 13.2.25 (substitution value lemma)

P.1.4 $\mathcal{T}_1^f : \forall$:

P.1.4.1 $\mathcal{I}_\varphi(\forall X . \mathbf{A}) = \top$, iff $\mathcal{I}_\psi(\mathbf{A}) = \top$ for all $a \in \mathcal{D}_t$

P.1.4.2 so in particular for some $a \in \mathcal{D}_t \neq \emptyset$. \square

\square

\square

▷ **Corollary B.3.2** \mathcal{T}_1^f is correct.



The only interesting steps are the cut rule, which can be directly handled by the substitution value lemma, and the rule for the existential quantifier, which we do in a separate lemma.

Soundness of $\mathcal{T}_1^f : \exists$

▷ **Lemma B.3.3** $\mathcal{T}_1^f : \exists$ transforms satisfiable tableaux into satisfiable ones.

▷ **Proof:** Let \mathcal{T}' be obtained by applying $\mathcal{T}_1^f : \exists$ to $\forall X . \mathbf{A}^\mathsf{F}$ in \mathcal{T} , extending it with $[f(X^1, \dots, X^n)/X](\mathbf{A})^\mathsf{F}$, where $W := \text{free}(\forall X . \mathbf{A}) = \{X^1, \dots, X^n\}$

P.1 Let \mathcal{T} be satisfiable in $\mathcal{M} := \langle \mathcal{D}, \mathcal{I} \rangle$, then $\mathcal{I}_\varphi(\forall X . \mathbf{A}) = \mathsf{F}$.

P.2 We need to find a model \mathcal{M}' that satisfies \mathcal{T}' (find interpretation for f)

P.3 By definition $\mathcal{I}_{\varphi, [a/X]}(\mathbf{A}) = \mathsf{F}$ for some $a \in \mathcal{D}$ (depends on $\varphi|_W$)

P.4 Let $g: \mathcal{D}^k \rightarrow \mathcal{D}$ be defined by $g(a_1, \dots, a_k) := a$, if $\varphi(X^i) = a_i$

P.5 choose $\mathcal{M}' = \langle \mathcal{D}, \mathcal{I}' \rangle$ with $\mathcal{I}' := \mathcal{I}, [g/f]$, then by subst. value lemma

$$\begin{aligned} \mathcal{I}'_\varphi([f(X^1, \dots, X^n)/X](\mathbf{A})) &= \mathcal{I}'_{\varphi, [\mathcal{I}'_\varphi(f(X^1, \dots, X^n))/X]}(\mathbf{A}) \\ &= \mathcal{I}'_{\varphi, [a/X]}(\mathbf{A}) = \mathsf{F} \end{aligned}$$

P.6 So $[f(X^1, \dots, X^n)/X](\mathbf{A})^\mathsf{F}$ satisfiable in \mathcal{M}' \square



This proof is paradigmatic for soundness proofs for calculi with Skolemization. We use the axiom of choice at the meta-level to choose a meaning for the Skolem function symbol.

Armed with the Model Existence Theorem for first-order logic (Theorem B.1.19), the completeness of first-order tableaux is similarly straightforward. We just have to show that the collection of tableau-irrefutable sentences is an abstract consistency class, which is a simple proof-transformation exercise in all but the universal quantifier case, which we postpone to its own Lemma.

Completeness of (\mathcal{T}_1^f)

▷ **Theorem B.3.4** \mathcal{T}_1^f is refutation complete.

▷ **Proof:** We show that $\nabla := \{\Phi \mid \Phi^\mathsf{T}$ has no closed Tableau} is an abstract consistency class

P.1 (∇_c , ∇_{\neg} , ∇_V , and ∇_A) as for propositional case.

P.2 (∇_V) by the lifting lemma below

P.3 (∇_A) Let \mathcal{T} be a closed tableau for $\neg(\forall X . \mathbf{A}) \in \Phi$ and $\Phi^T * [c/X](\mathbf{A})^F \in \nabla$.

$$\begin{array}{ccc} \Psi^T & & \Psi^T \\ \forall X . \mathbf{A}^F & & \forall X . \mathbf{A}^F \\ [c/X](\mathbf{A})^F & & [f(X^1, \dots, X^k)/X](\mathbf{A})^F \\ Rest & & [f(X^1, \dots, X^k)/c](Rest) \end{array}$$

□



©: Michael Kohlhase

999



So we only have to treat the case for the universal quantifier. This is what we usually call a “lifting argument”, since we have to transform (“lift”) a proof for a formula $\theta(\mathbf{A})$ to one for \mathbf{A} . In the case of tableaux we do that by an induction on the tableau refutation for $\theta(\mathbf{A})$ which creates a tableau-isomorphism to a tableau refutation for \mathbf{A} .

Tableau-Lifting

▷ **Theorem B.3.5** If \mathcal{T}_θ is a closed tableau for a st $\theta(\Phi)$ of formulae, then there is a closed tableau \mathcal{T} for Φ .

▷ **Proof:** by induction over the structure of \mathcal{T}_θ we build an isomorphic tableau \mathcal{T} , and a tableau-isomorphism $\omega: \mathcal{T} \rightarrow \mathcal{T}_\theta$, such that $\omega(\mathbf{A}) = \theta(\mathbf{A})$.

P.1 only the tableau-substitution rule is interesting.

P.2 Let $\theta(\mathbf{A}^i)^T$ and $\theta(\mathbf{B}^i)^F$ cut formulae in the branch Θ_θ^i of \mathcal{T}_θ

P.3 there is a joint unifier σ of $\theta(\mathbf{A}^1) =? \theta(\mathbf{B}^1) \wedge \dots \wedge \theta(\mathbf{A}^n) =? \theta(\mathbf{B}^n)$

P.4 thus $\sigma \circ \theta$ is a unifier of \mathbf{A} and \mathbf{B}

P.5 hence there is a most general unifier ρ of $\mathbf{A}^1 =? \mathbf{B}^1 \wedge \dots \wedge \mathbf{A}^n =? \mathbf{B}^n$

P.6 so Θ is closed

□



©: Michael Kohlhase

1000



Again, the “lifting lemma for tableaux” is paradigmatic for lifting lemmata for other refutation calculi.

B.4 Soundness and Completeness of First-Order Resolution

Correctness (CNF)

▷ **Lemma B.4.1** A set Φ of sentences is satisfiable, iff $CNF^1(\Phi)$ is.

▷ **Proof:** propositional rules and \vee -rule are trivial; do the \exists -rule

P.1 Let $\forall X . \mathbf{A}^F$ satisfiable in $\mathcal{M} := \langle \mathcal{D}, \mathcal{I} \rangle$ and $free(\mathbf{A}) = \{X^1, \dots, X^n\}$

P.2 $\mathcal{I}_\varphi(\forall X . \mathbf{A}) = F$, so there is an $a \in \mathcal{D}$ with $\mathcal{I}_{\varphi, [a/X]}(\mathbf{A}) = F$ (only depends on $\varphi|_{free(\mathbf{A})}$)

P.3 let $g: \mathcal{D}^n \rightarrow \mathcal{D}$ be defined by $g(a_2, \dots, a_n) := a$, iff $\varphi(X^i) = a_i$.

P.4 choose $\mathcal{M}' := \langle \mathcal{D}, \mathcal{I}' \rangle$ with $\mathcal{I}'(f) := g$, then $\mathcal{I}'_\varphi([f(X^1, \dots, X^k)/X](\mathbf{A})) = \mathbf{F}$

P.5 Thus $[f(X^1, \dots, X^k)/X](\mathbf{A})^\mathbf{F}$ is satisfiable in \mathcal{M}' \square



Resolution (Correctness)

- ▷ **Definition B.4.2** A clause is called **satisfiable**, iff $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$ for one of its literals \mathbf{A}^α .
- ▷ **Lemma B.4.3** \square is *unsatisfiable*
- ▷ **Lemma B.4.4** CNF-transformations preserve satisfiability (see above)
- ▷ **Lemma B.4.5** Resolution and Factorization too



Completeness (\mathcal{R}^1)

- ▷ **Theorem B.4.6** \mathcal{R}^1 is refutation complete.
- ▷ **Proof:** $\nabla := \{\Phi \mid \Phi^\top \text{ has no closed tableau}\}$ is an abstract consistency class
- P.1** (∇_c , ∇_\neg , ∇_V , and ∇_\wedge) as for propositional case.
- P.2** (∇_V) by the lifting lemma below
- P.3** (∇_\exists) Let \mathcal{T} be a closed tableau for $\neg(\forall X . \mathbf{A}) \in \Phi$ and $\Phi^\top * [c/X](\mathbf{A})^\mathbf{F} \in \nabla$.
- P.4** $CNF^1(\Phi^\top) = CNF^1(\Psi^\top) \cup CNF^1([f(X^1, \dots, X^k)/X](\mathbf{A})^\mathbf{F})$
- P.5** $[f(X^1, \dots, X^k)/c](CNF^1(\Phi^\top)) * [c/X](\mathbf{A})^\top \mathbf{F} = CNF^1(\Phi^\top)$
- P.6** so $\mathcal{R}^1: CNF^1(\Phi^\top) \vdash_{\mathcal{D}'} \square$, where $\mathcal{D}' = [f(X^1, \dots, X^k)/c](\mathcal{D})$. \square



Clause Set Isomorphism

- ▷ **Definition B.4.7** Let \mathbf{B} and \mathbf{C} be clauses, then a **clause isomorphism** $\omega: \mathbf{C} \rightarrow \mathbf{D}$ is a bijection of the literals of \mathbf{C} and \mathbf{D} , such that $\omega(\mathbf{L})^\alpha = \mathbf{M}^\alpha$ (conserves labels)

We call ω **θ -compatible**, iff $\omega(\mathbf{L}^\alpha) = \theta(\mathbf{L})^\alpha$

- ▷ **Definition B.4.8** Let Φ and Ψ be clause sets, then we call a bijection $\Omega: \Phi \rightarrow \Psi$ a **clause set isomorphism**, iff there is a clause isomorphism $\omega: \mathbf{C} \rightarrow \Omega(\mathbf{C})$ for each $\mathbf{C} \in \Phi$.

- ▷ **Lemma B.4.9** If $\theta(\Phi)$ is set of formulae, then there is a θ -compatible clause set isomorphism $\Omega: CNF^1((\theta)\Phi) \rightarrow CNF^1(\theta(\Phi))$.
- ▷ **Proof Sketch:** by induction on the CNF derivation of $CNF^1(\Phi)$. □



Lifting for \mathcal{R}^1

- ▷ **Theorem B.4.10** If $\mathcal{R}^1: \theta(\Phi) \vdash_{\mathcal{D}_\theta} \square$ for a set $\theta(\Phi)$ of formulae, then there is a \mathcal{R}^1 -refutation for Φ .
- ▷ **Proof:** by induction over \mathcal{D}_θ we construct a \mathcal{R}^1 -derivation $\mathcal{R}^1: \Phi \vdash_{\mathcal{D}} \mathbf{C}$ and a θ -compatible clause set isomorphism $\Omega: \mathcal{D} \rightarrow \mathcal{D}_\theta$

$$\text{P.1 If } \mathcal{D}_\theta \text{ ends in } \frac{\mathcal{D}'_\theta}{\theta(\mathbf{A}) \vee \theta(\mathbf{C})^\top} \quad \frac{\mathcal{D}''_\theta}{\theta(\mathbf{B})^F \vee \theta(\mathbf{D})} \quad res \\ \sigma(\theta(\mathbf{C})) \vee \sigma(\theta(\mathbf{B}))$$

then we have (IH) clause isomorphisms $\omega': \mathbf{A}^\top \vee \mathbf{C} \rightarrow \theta(\mathbf{A})^\top \vee \theta(\mathbf{C})$ and $\omega': \mathbf{B}^F \vee \mathbf{D} \rightarrow \theta(\mathbf{B})^F \vee \theta(\mathbf{D})$

$$\text{P.2 thus } \frac{\mathbf{A}^\top \vee \mathbf{C} \quad \mathbf{B}^F \vee \mathbf{D}}{\rho(\mathbf{C}) \vee \rho(\mathbf{B})} Res \quad \text{where } \rho = \text{mgu}(\mathbf{A}, \mathbf{B}) \quad (\text{exists, as } \sigma \circ \theta \text{ unifier})$$

□


Bibliography

- [Bac00] Fahiem Bacchus. *Subset of PDDL for the AIPS2000 Planning Competition*. The AIPS-00 Planning Competition Comitee. 2000.
- [BF95] Avrim L. Blum and Merrick L. Furst. “Fast planning through planning graph analysis”. In: *Proceedings of the 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.
- [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.
- [Dav67] Donald Davidson. “Truth and Meaning”. In: *Synthese* 17 (1967).
- [DF31] B. De Finetti. “Sul significato soggettivo della probabilità”. In: *Fundamenta Mathematicae* 17 (1931), pp. 298–329.
- [DHK15] Carmel Domshlak, Jörg Hoffmann, and Michael Katz. “Red-Black Planning: A New Systematic Approach to Partial Delete Relaxation”. In: *Artificial Intelligence* 221 (2015), pp. 73–114.
- [Ede01] Stefan Edelkamp. “Planning with Pattern Databases”. In: *Proceedings of the 6th European Conference on Planning (ECP’01)*. Ed. by A. Cesta and D. Borrajo. Springer-Verlag, 2001, pp. 13–24.
- [FD14] Zohar Feldman and Carmel Domshlak. “Simple Regret Optimization in Online Planning for Markov Decision Processes”. In: *Journal of Artificial Intelligence Research* 51 (2014), pp. 165–205.
- [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.

- [FN71] Richard E. Fikes and Nils Nilsson. “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”. In: *Artificial Intelligence* 2 (1971), pp. 189–208.
- [Gen34] Gerhard Gentzen. “Untersuchungen über das logische Schließen I”. In: *Mathematische Zeitschrift* 39.2 (1934), pp. 176–210.
- [Ger+09] Alfonso Gerevini et al. “Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners”. In: *Artificial Intelligence* 173.5-6 (2009), pp. 619–668.
- [GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. BN book: Freeman, 1979.
- [Glo] *Grundlagen der Logik in der Informatik*. Course notes at https://www8.cs.fau.de/_media/ws16:gloin:skript.pdf. URL: https://www8.cs.fau.de/_media/ws16:gloin:skript.pdf (visited on 10/13/2017).
- [GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.
- [GS05] Carla Gomes and Bart Selman. “Can get satisfaction”. In: *Nature* 435 (2005), pp. 751–752.
- [GSS03] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. “Planning through Stochastic Local Search and Temporal Action Graphs”. In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 239–290.
- [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.
- [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.
- [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.

- [Hof11] Jörg Hoffmann. “Every806thing You Always Wanted to Know about Planning (But Were Afraid to Ask)”. In: *Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI’11)*. Ed. by Joscha Bach and Stefan Edelkamp. Vol. 7006. Lecture Notes in Computer Science. Springer, 2011, pp. 1–13. URL: <http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf>.
- [How60] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
- [JN33] E. S. Pearson J. Neyman. “IX. On the problem of the most efficient tests of statistical hypotheses”. In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 231.694-706 (1933), pp. 289–337. DOI: [10.1098/rsta.1933.0009](https://doi.org/10.1098/rsta.1933.0009).
- [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.
- [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.
- [KNS97] B. Kessler, G. Nurnberg, and H. Schütze. “Automatic detection of text genre”. In: *CoRR* cmp-lg/9707002 (1997).
- [Koe+97] Jana Koehler et al. “Extending Planning Graphs to an ADL Subset”. In: *Proceedings of the 4th European Conference on Planning (ECP’97)*. Ed. by S. Steel and R. Alami. Springer-Verlag, 1997, pp. 273–285. URL: <ftp://ftp.informatik.uni-freiburg.de/papers/ki/koehler-etal-ecp-97.ps.gz>.
- [Koh08] Michael Kohlhase. “Using L^AT_EX as a Semantic Markup Format”. In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: <https://kwarc.info/kohlhase/papers/mcs08-stex.pdf>.
- [Koh18] Michael Kohlhase. *sTeX: Semantic Markup in T_EX/L^AT_EX*. Tech. rep. Comprehensive T_EX Archive Network (CTAN), 2018. URL: <http://www.ctan.org/get/macros/latex/contrib/stex/sty/stex.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.

- [KS99] Henry Kautz and Bart Selman. “Unifying SAT-based and Graph-based Planning”. In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Thomas Dean. Stockholm: Morgan Kaufmann, 1999, pp. 318–325.
- [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/>.
- [MR91] John Mylopoulos and Ray Reiter, eds. Sydney, Australia: Morgan Kaufmann, San Mateo, CA, 1991.
- [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.
- [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.
- [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.
- [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>.
- [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-k92.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.
- [RN95] Stuart J. Russell and Peter Norvig. *Artificial Intelligence — A Modern Approach*. Upper Saddle River, NJ: Prentice Hall, 1995.

- [RW10] Silvia Richter and Matthias Westphal. “The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks”. In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 127–177.
- [Sil+16] David Silver et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529 (2016), pp. 484–503. URL: <http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>.
- [Smu63] Raymond M. Smullyan. “A Unifying Principle for Quantification Theory”. In: *Proc. Nat. Acad. Sciences* 49 (1963), pp. 828–832.
- [SR91] C. Samuelsson and M. Rayner. “Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system”. 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. 609–615.
- [Tur50] Alan Turing. “Computing Machinery and Intelligence”. In: *Mind* 59 (1950), pp. 433–460.
- [Wal75] David Waltz. “Understanding Line Drawings of Scenes with Shadows”. In: *The Psychology of Computer Vision*. Ed. by P. H. Winston. McGraw-Hill, 1975, pp. 1–19.
- [WHI] *Human intelligence — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Human_intelligence (visited on 04/09/2018).

Index

- χ^2
 - pruning, 467
- \mathcal{C} -consistent, 582, 592
- \mathcal{C} -derivation, 183
- \mathcal{C} -refutable, 582, 592
- ∇ -Hintikka Set, 585, 595
- θ -compatible, 602
- k -fold
 - cross-validation, 468
- n -ary, 480
- n -gram, 552
 - model, 552
- n -layer, 487
- A^* search, 95
- \mathcal{U} -reducible, 256
- 0/1
 - loss, 471
- abduction, 262
- absolute
 - value
 - loss, 471
- absorbing
 - state, 442
- abstract
 - consistency
 - class, 583, 593
- action, 49, 68, 109, 281, 440
 - node, 416
- activation, 486
 - function, 486
- actuator, 48, 345, 352
- add
 - list, 281
- additive, 415
 - rewards, 442
- admissible, 94, 184, 295
- admits
 - weakening, 183
- agent, 48, 345, 352
 - architecture, 49, 55
 - function, 49
 - program, 49, 55
- AGI, 18
- AI, 10, 336
 - complete, 18
- aleatory
 - variable, 364
- algorithm
 - search, 67
- alphabeta
 - search, 120
- alphabetical
 - variant, 233
- ambiguous, 549
- analysis
 - conceptual, 549
 - logical, 549
- answer
 - substitution, 30
- applicable, 69, 283
- applied
 - AI, 18
- approximatively
 - correct, 474
- arc
 - consistent, 156
- Artificial
 - Intelligence, 10, 336
- assignment, 139, 175
- assumption, 183
- atom, 178, 191
- atomic, 60, 178, 191
 - event, 365
 - formula, 232
- attribute, 61, 460
 - value, 460
- attribute-based
 - representation, 460
- autonomous, 51, 354
- axiom, 183
- back-propagation
 - learning
 - algorithm, 492
 - rule, 491
- backtrack
 - point, 32
- backtracking, 31
 - search, 142

Bayes'
rule, 374
Bayesian
classifiers, 379, 508
network, 389
behaviorism, 46
belief
network, 389
state, 319, 357
bias
weight, 487
bigram, 552
biimplication, 177
binary
constraint, 136
CSP, 137
biological
neural
network, 485
blackbox, 110
description, 69
body
literal, 28
Boltzmann
machine, 487
Boolean, 364
bound
variable
occurrence, 233
brain, 485
calculus, 175, 183
canonical
distribution, 395
chain
rule, 371
choice
literal, 213
classification, 461
problem, 477
clause, 199
isomorphism, 602
set, 199
clause set
isomorphism, 602
cliff, 484
closed, 233
branch, 193, 248
closed under
subset, 583
subsets, 593
CNF, 191
CNF transformation calculus, 199
Cognitive
Neuroscience, 47
Science, 47
cognitive
revolution, 46
commutative, 142
compact, 583, 584, 593
complete, 185, 254
completeness, 76, 175
complex, 178, 191
formula, 232
complexity
theory, 38
conceptual
analysis, 549
conclusion, 183
conditional
probability, 368
distribution, 368
table, 389
conditionally
independent, 377
conflict
graph, 216
vertex, 214
conformant, 320
conjunction, 177
conjunctive
normal
form, 191, 199
Conjunctive Normal Form
Calculus, 257
connectionism, 486
connective, 177, 232
consistent, 94, 139, 391, 457
with, 458
constraint, 138
network, 138
satisfaction
problem, 129
continuity, 407
continuous, 54, 355, 477
contradiction, 582, 592
corpus, 552
linguistics, 552
correct, 185, 254
cosine
similarity, 556
cost, 75
function, 69
counterexamples, 179
critic, 58
CSP, 129
cummulative
distribution, 411

current
 state, 102
 cut
 operator, 35
 cut-off
 states, 115
 data
 matrix, 481
 DBN, 436
 DDN, 449
 decision
 boundary, 482
 list, 475
 learning, 477
 theory, 405
 tree
 learning, 462
 pruning, 466
 declarative
 description, 69
 decomposability, 407
 deduction, 175, 262
 degree
 heuristic, 145
 degree of
 belief, 363
 delete
 list, 281
 delete-relaxation, 304
 depth
 first
 search, 81
 depth-limited
 search, 85
 derivation
 relation, 182
 derived, 175
 inference
 rule, 196, 200
 rule, 196
 derives, 193, 248
 description
 length, 472
 determination, 519
 determines, 519
 deterministic, 54, 355, 395
 direct
 utility
 estimation, 534
 discharge, 236
 discharged, 187, 238
 discount
 factor, 442
 discounted
 rewards, 442
 discrete, 54, 355
 disjunction, 177
 disjunctive
 normal
 form, 191, 199
 DNF, 191, 199
 domain, 129, 138
 dominates, 100
 DPLL, 208
 duplicate
 pruning, 88
 dynamic, 54, 355, 436
 decision
 network, 449
 empirical
 loss, 471
 empty
 assignment, 139
 clause, 199
 end
 state, 109
 entailment, 175
 entails, 179
 entropy, 463
 environment, 48, 345, 352
 episode, 54, 355
 episodic, 54, 355
 decision
 theory, 405
 epoch, 493
 equational
 system, 253
 equivalence, 177
 equivalent, 151
 error
 rate, 433, 468, 474
 estimated
 best
 hypothesis, 471
 evaluation
 function, 95, 115
 event, 364, 365, 398
 evidence
 variable, 398, 425
 example, 457
 exhaustive, 363
 expected
 utility, 406, 442
 explicit, 110
 exploitation, 538
 exploration, 538

extends, 139
 extension, 235
 of, 139
 fact, 281
 clause, 261
 factored, 61
 false
 under, 179
 falsifiable, 179, 182
 falsified by \mathcal{M} , 182
 falsifies, 179
 feed-forward
 network, 487
 filtering, 426
 finite-domain, 364
 First-order
 Markov
 process, 424
 first-order
 logic, 231
 natural deduction
 calculus, 242
 signature, 232
 first-order logic with equality, 243
 formal
 system, 183, 184
 formula, 175, 182
 position, 244
 Forward-backward
 algorithm, 429
 free
 variable, 233
 occurrence, 233
 fringe, 76
 full
 AI, 18
 joint
 probability distribution, 365
 fully
 observable, 54, 355
 function
 constant, 232
 game
 state
 space, 109
 generalization
 loss, 471
 genre
 classification, 553
 goal, 28, 281
 distance
 function, 93
 state, 68, 283
 test, 69
 goal-based
 agent, 57
 gradient
 descent, 479
 graph
 search
 algorithm, 88
 graphical
 model, 389
 greedy, 537
 search, 89
 ground, 233
 GSAT
 algorithm, 207
 head, 28
 literal, 28
 heuristic, 89, 92, 295
 function, 92, 295
 hidden, 487
 Markov
 model, 432
 unit, 490
 variable, 398
 higher-order
 constraint, 137
 hindsight, 426
 HMM, 432
 filtering
 equation, 432
 smoothing
 equation, 432
 holdout
 cross-validation, 468
 Hopfield
 network, 487
 horizon, 115
 Horn
 clause, 261
 logic, 262
 hypotheses, 183
 hypothesis, 457
 complexity, 472
 prior, 503
 space, 457
 hypothetical
 reasoning, 187, 238
 ideal
 delete-relaxation
 heuristic, 309
 identically

distributed, 467
 idiot
 Bayes
 model, 379, 508
 IID, 467
 Imitation
 Game, 46
 implication, 177
 graph, 214
 implied
 literal, 213
 inconsistent, 139
 independent, 369, 467
 independent and identically distributed, 467
 indifference, 407
 individual, 231, 234
 variable, 232
 induction, 262
 inductive
 learning
 problem, 457
 Inference, 150
 inference
 rule, 183, 199
 information, 463
 gain, 464
 gathering
 agent, 421
 need, 555
 retrieval, 555
 value
 theory, 418
 inhibited, 396
 initial
 state, 68, 109, 281
 input
 layer, 487
 sample, 457
 unit, 487
 instrumental
 AI, 18
 intercept, 487
 Interpretation, 178
 interpretation, 234
 introduced, 235
 inverse
 document
 frequency, 556
 IR, 555
 irrelevant, 466
 Iterative
 deepening
 search, 85
 joint
 probability
 distribution, 365
 judgment, 189, 240
 Kernel
 function, 498
 knowledge
 base, 28
 labeled
 formula, 191
 language
 identification, 553
 model, 552
 layer, 487
 leak
 node, 396
 learning
 agent, 58
 curve, 465
 element, 58
 rate, 479
 least
 constraining
 value, 146
 leave-one-out
 cross-validation, 468
 likelihood, 503
 linear
 regression, 478
 separator, 482, 489
 linearly
 separable, 482
 linguistically
 realized, 546
 link, 486
 literal, 191
 literals, 195
 local
 search, 102
 log-likelihood, 505
 logic, 47
 program, 261
 logic-based
 inductive
 learning, 511
 logical
 analysis, 549
 system, 182
 logistic
 regression, 484
 LOOCV, 468
 loss

function, 470
lottery, 406

MAP
learning, 504

maximum
a posteriori
learning, 504

margin, 496

Markov
chain, 424
decision
process, 440
process, 424
property, 424

mating, 251

maximal
margin
separator, 496

maximizes, 409

maximum
likelihood
learning, 505

McCulloch-Pitts
unit, 486

McCullorch-Pitts
network, 486
unit, 486

MDL
hypothesis, 472
learning, 505

MDP, 440

MEU, 409
principle, 330, 576

micromort, 409

min-conflicts, 165

minimax, 113
algorithm, 113

Minimum
remaining
values, 145

minimum
description
length, 472

mixing
time, 428

ML
learning, 505

Model, 234

model, 178, 182
selection, 469

model-based
agent, 56, 356

monitoring, 426

monomial, 199

monotonic, 183

monotonicity, 407

Monte-Carlo
sampling, 122
tree search, 122

more
general, 252

most
likely
explanation, 426

most general
unifier, 252

move, 109

MRV, 145

multiplicity, 250

multiset
ordering, 255

multivariate, 480
linear
regression, 480

mutual
preferential
independence, 415

mutually
exclusive, 363
utility-independent, 416

myopic, 421

naive
Bayes
model, 379, 508

name, 281

named
entity
recognition, 553

narrow
AI, 18

natural
language
generation, 543
processing, 543
understanding, 543

negation, 177

negative, 193, 248, 461

neural
computation, 486
network, 486
networks, 486

neuron, 485

NLP, 543

NLU, 543

node, 75

noisy

disjunction
 node, 396
 normal
 conjunctive (form), 199
 disjunctive (form), 199
 normalization
 constant, 373
 normalized
 utilities, 409
 Normative, 47
 null
 hypothesis, 466
 number of
 decisions, 210
 observation
 model, 447
 Ockham's
 razor, 459
 offline
 problem
 solving, 67
 one
 hot, 558
 online
 problem
 solving, 67
 OOV, 554
 open
 branch, 193, 248
 opposite
 literal, 191
 optimal, 110
 planning, 273, 288
 policy, 443
 relaxed
 plan, 308
 optimality, 76
 optimization, 469
 orderability, 407
 ordinal
 utility
 function, 408
 out-of-vocabulary, 554
 outcome, 364, 457
 output
 layer, 487
 unit, 487
 overfitting, 466
 PAC
 learning, 474
 page
 rank, 557
 parallel
 distributed
 processing, 486
 part
 physical, 551
 partial
 assignment, 139
 partially
 observable, 54, 355
 MDP, 447
 partner
 literal, 191
 passive
 ADP
 algorithm, 535
 path
 cost, 75
 PEAS, 52, 354
 peeking, 468
 percept, 49
 perceptron, 488
 learning
 rule, 482
 network, 488
 unit, 486
 perfect
 heuristic, 295
 performance
 element, 58
 measure, 51, 353
 standard, 58
 perplexity, 555
 physical
 part, 551
 plan, 283
 for, 283
 planner, 274, 288
 planning
 system, 274, 288
 task, 281
 tool, 274, 288
 policy, 440
 evaluation, 445
 improvement, 445
 iteration
 algorithm, 445
 polynomial
 kernel, 498
 polytree, 401
 POMDP, 447
 position, 109
 positive, 461
 positively
 influences, 412

possible
 world, 363
posterior
 probability, 368
precondition, 281
predicate
 constant, 232
prediction, 426
Preference, 137
preference, 406
preferentially
 independent, 415
preferred, 407
prescriptive, 47
prior
 probability, 364
prize, 406
probabilistic
 inference
 task, 398
 network, 389
probability
 distribution, 365
 function, 364
 measure, 366
 model, 364
 theory, 363
probably
 approximately
 correct, 474
problem
 description
 language, 69
 generator, 59
 offline (solving), 67
Prolog
 clause, 28
 constant, 28
 fact, 28
 function, 28
 predicate, 28
 program, 28
 rule, 28
 term, 28
 variable, 28
proof, 184
proof-reflexive, 182
proof-transitive, 183
Proposition, 232
proposition, 281, 366
propositional
 formula, 366
 logic, 177
 natural deduction

calculus, 187, 212
variable, 177
prune, 466
QALY, 410
quadratic
 programming, 497
qualification
 problem, 318
quality-adjusted
 life
 years, 410
query, 28, 261, 555
 variable, 398
random
 quantity, 364
 variable, 364
Rational, 47
rational, 51, 330, 353,
reading, 549
realizable, 460
reasonable, 582, 592
recurrent, 487
recursive
 state
 estimation, 426
reflex
 agent
 with state, 56, 330
regression
 problem, 477
regularization, 472
 function, 472
reinforcement, 531
 learning, 531
relaxation, 100
relaxed
 plan, 305
 problem, 100
replacing, 244
resolution
 calculus, 199
 proof, 199
 refutation, 199
result
 relation, 69
reward, 440, 531
 function, 440
reward-to-go, 534
Ridge, 104
risk-averse, 410
risk-prone, 410
Rollup

filtering, 437
 rule
 clause, 261
 derived, 196
 inference, 199
 sample
 complexity, 474
 space, 364
 SAT
 Problem, 205
 solver, 205
 satisfaction
 relation, 182
 satisfiable, 179, 182, 602
 satisficing
 planning, 273, 288
 satisfied by \mathcal{M} , 182
 satisfies, 179
 saturated, 193, 248
 search
 algorithm, 67
 depth
 limit, 115
 greedy, 89
 problem, 68
 strategy, 76
 tree, 74
 Second-order
 Markov
 process, 424
 semantics, 175
 semidynamic, 54, 355
 Sensor
 matrix, 432
 sensor, 48, 345, 352
 Markov
 property, 425
 model, 320, 358, 425
 sensorless
 planning, 320
 sentence, 233
 sequent, 189, 240
 sequential, 54, 355
 sigmoid
 perceptron
 unit, 486
 simple
 reflex
 agent, 55, 355
 single-agent, 54, 355
 singly
 connected, 401
 Skolem
 constant, 232
 smoothing, 426
 softmax
 function, 560
 solution, 69, 140
 solvable, 140, 283
 solved, 253
 form, 253
 sound, 185
 soundness, 175
 space
 complexity, 76
 spanning
 mating, 251
 sparse
 model, 481
 squared
 error
 loss, 471
 standard
 lottery, 409
 state, 68, 109, 283, 440
 estimation, 426
 space, 74, 282
 variable, 425
 stateful
 reflex
 agent, 56, 356
 static, 54, 355
 stationary, 425, 442
 distribution, 428
 statistical
 significance, 466
 step
 cost, 69
 stochastic, 54, 355
 variable, 364
 stochastically
 dominates, 411
 strategy, 110
 strictly
 dominates, 411
 tighter, 152
 STRIPS, 280
 planning
 task, 281
 strong
 AI, 18
 structured, 61
 subformula at p , 244
 substitutability, 407
 substitution, 235
 succesor
 state, 68

support, 235
 vector, 497
 SVM
 equation, 497
 syntax, 175
 systematic, 101
 \mathcal{T}_0 -theorem, 193, 248
 tableau
 proof, 193, 248
 refutation, 193, 248
 target
 function, 457
 temporal
 probability
 model, 423
 term, 232
 frequency, 556
 terminal, 466
 state, 109
 test
 set, 465, 468
 test calculi, 193, 248
 text
 corpus, 552
 tf-idf
 word
 embedding, 558
 theorem, 184
 theory
 complexity, 38
 threshold
 function, 482
 tighter, 152
 time
 complexity, 76
 slice, 436
 sliced, 436
 structure, 423
 total
 cost, 472
 training
 curve, 493
 set, 457
 Transition
 matrix, 432
 transition
 function, 319, 357
 model, 68, 319, 357, 425, 440
 relation, 109, 320, 358
 transitive-reflexive
 closure, 391
 transitivity, 407
 tree

search
 algorithm, 74
 trigram, 552
 true
 under, 179
 truth
 value, 231, 234
 Turing
 test, 46
 unary, 478
 constraint, 136
 unconditional
 probability, 364
 underfitting, 466
 unifier, 252
 unigram, 552
 unit, 486
 resolution, 210
 unitary, 254
 univariate, 478
 Universe, 178, 234
 universe, 234
 unroll, 437
 unsatisfiable, 179, 182, 199
 unsupervised
 learning, 531
 UR, 210
 utility, 58, 361, 443, 533
 function, 58, 109, 361
 node, 416
 utility-based
 agent, 58, 361
 utility-independent, 415
 valid, 179, 182
 valuation, 587, 596
 value, 61
 function, 178, 234, 408
 iteration, 444
 node, 416
 value of
 perfect
 information, 419
 variable, 129, 138
 assignment, 178, 234
 elimination, 401
 Viterbi
 algorithm, 431
 VPI, 419
 weak
 AI, 18
 web

search, 555
weight
 space, 479
well-formed
 propositional
 formula, 177
whitebox
 description, 69
word
 embedding, 558
 frequency
 vector, 555
worked
 off, 198
world
 model, 56, 356
 state, 283, 319, 357