Undergraduate Topics in Computer Science

Wolfgang Ertel

Introduction to Artificial Intelligence

UTiCS

Springer

Wolfgang Ertel

Introduction to Artificial Intelligence

Translated by Nathanael Black
With illustrations by Florian Mast

Springer

Prof. Dr. Wolfgang Ertel
FB Elektrotechnik und Informatik
Hochschule Ravensburg-Weingarten
University of Applied Sciences
Weingarten
Germany
ertel@hs-weingarten.de

ISSN 1863-7310
ISBN 978-0-85729-298-8 e-ISBN 978-0-85729-299-5
DOI 10.1007/978-0-85729-299-5
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2011923216

Originally published in the German language by Vieweg+Teubner, 65189 Wiesbaden, Germany, as "Ertel, Wolfgang: Grundkurs Künstliche Intelligenz. 2. rev. edition".
© Vieweg+Teubner | GWV Fachverlage GmbH, Wiesbaden 2009

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Contents

Introduction 1

1.1 What Is Artificial Intelligence?

The term *artificial intelligence* stirs emotions. For one thing there is our fascination with *intelligence*, which seemingly imparts to us humans a special place among life forms. Questions arise such as "*What is intelligence*?", "*How can one measure intelligence*?" or "*How does the brain work*?". All these questions are meaningful when trying to understand artificial intelligence. However, the central question for the engineer, especially for the computer scientist, is the question of the intelligent machine that behaves like a person, showing intelligent behavior.

The attribute *artificial* might awaken much different associations. It brings up fears of intelligent cyborgs. It recalls images from science fiction novels. It raises the question of whether our highest good, the soul, is something we should try to understand, model, or even reconstruct.

With such different offhand interpretations, it becomes difficult to define the term *artificial intelligence* or *AI* simply and robustly. Nevertheless I would like to try, using examples and historical definitions, to characterize the field of AI. In 1955, John McCarthy, one of the pioneers of AI, was the first to define the term *artificial intelligence*, roughly as follows:

> The goal of AI is to develop machines that behave as though they were intelligent.

To test this definition, the reader might imagine the following scenario. Fifteen or so small robotic vehicles are moving on an enclosed four by four meter square surface. One can observe various behavior patterns. Some vehicles form small groups with relatively little movement. Others move peacefully through the space and gracefully avoid any collision. Still others appear to follow a leader. Aggressive behaviors are also observable. Is what we are seeing intelligent behavior?

According to McCarthy's definition the aforementioned robots can be described as intelligent. The psychologist Valentin Braitenberg has shown that this seemingly complex behavior can be produced by very simple electrical circuits [Bra84]. So-called Braitenberg vehicles have two wheels, each of which is driven by an independent electric motor. The speed of each motor is influenced by a light sensor on

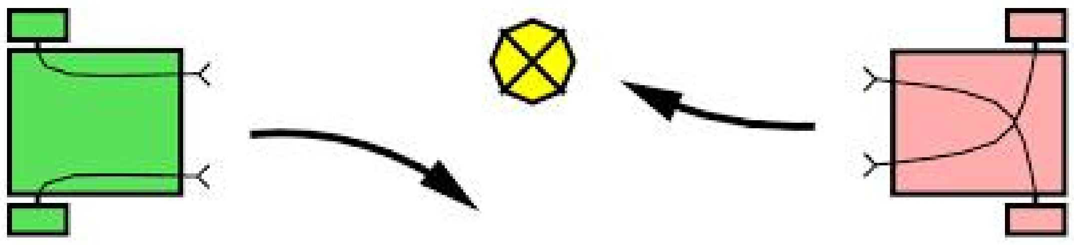


Fig. 1.1 Two very simple Braitenberg vehicles and their reactions to a light source

the front of the vehicle as shown in Fig. 1.1. The more light that hits the sensor, the faster the motor runs. Vehicle 1 in the left part of the figure, according to its configuration, moves away from a point light source. Vehicle 2 on the other hand moves toward the light source. Further small modifications can create other behavior patterns, such that with these very simple vehicles we can realize the impressive behavior described above.

Clearly the above definition is insufficient because AI has the goal of solving difficult practical problems which are surely too demanding for the Braitenberg vehicle. In the Encyclopedia Britannica [Bri91] one finds a Definition that goes like:

> AI is the ability of digital computers or computer controlled robots to solve problems that are normally associated with the higher intellectual processing capabilities of humans ...

But this definition also has weaknesses. It would admit for example that a computer with large memory that can save a long text and retrieve it on demand displays intelligent capabilities, for memorization of long texts can certainly be considered a *higher intellectual processing capability* of humans, as can for example the quick multiplication of two 20-digit numbers. According to this definition, then, every computer is an AI system. This dilemma is solved elegantly by the following definition by Elaine Rich [Ric83]:

> Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.

Rich, tersely and concisely, characterizes what AI researchers have been doing for the last 50 years. Even in the year 2050, this definition will be up to date.

Tasks such as the execution of many computations in a short amount of time are the strong points of digital computers. In this regard they outperform humans by many multiples. In many other areas, however, humans are far superior to machines. For instance, a person entering an unfamiliar room will recognize the surroundings within fractions of a second and, if necessary, just as swiftly make decisions and plan actions. To date, this task is too demanding for autonomous[1] robots. According to Rich's definition, this is therefore a task for AI. In fact, research on autonomous robots is an important, current theme in AI. Construction of chess computers, on the other hand, has lost relevance because they already play at or above the level of grandmasters.

It would be dangerous, however, to conclude from Rich's definition that AI is only concerned with the pragmatic implementation of intelligent processes. Intelligent systems, in the sense of Rich's definition, cannot be built without a deep un-

[1] An autonomous robot works independently, without manual support, in particular without remote control.

derstanding of human reasoning and intelligent action in general, because of which neuroscience (see Sect. 1.1.1) is of great importance to AI. This also shows that the other cited definitions reflect important aspects of AI.

A particular strength of human intelligence is adaptivity. We are capable of adjusting to various environmental conditions and change our behavior accordingly through *learning*. Precisely because our learning ability is so vastly superior to that of computers, *machine learning* is, according to Rich's definition, a central subfield of AI.

1.1.1 Brain Science and Problem Solving

Through research of intelligent systems we can try to understand how the human brain works and then model or simulate it on the computer. Many ideas and principles in the field of neural networks (see Chap. 9) stem from brain science with the related field of neuroscience.

A very different approach results from taking a goal-oriented line of action, starting from a problem and trying to find the most optimal solution. How humans solve the problem is treated as unimportant here. The method, in this approach, is secondary. First and foremost is the optimal intelligent solution to the problem. Rather than employing a fixed method (such as, for example, predicate logic) AI has as its constant goal the creation of intelligent agents for as many different tasks as possible. Because the tasks may be very different, it is unsurprising that the methods currently employed in AI are often also quite different. Similar to medicine, which encompasses many different, often life-saving diagnostic and therapy procedures, AI also offers a broad palette of effective solutions for widely varying applications. For mental inspiration, consider Fig. 1.2 on page 4. Just as in medicine, there is no universal method for all application areas of AI, rather a great number of possible solutions for the great number of various everyday problems, big and small.

Cognitive science is devoted to research into human thinking at a somewhat higher level. Similarly to brain science, this field furnishes practical AI with many important ideas. On the other hand, algorithms and implementations lead to further important conclusions about how human reasoning functions. Thus these three fields benefit from a fruitful interdisciplinary exchange. The subject of this book, however, is primarily problem-oriented AI as a subdiscipline of computer science.

There are many interesting philosophical questions surrounding intelligence and artificial intelligence. We humans have consciousness; that is, we can think about ourselves and even ponder that we are able to think about ourselves. How does consciousness come to be? Many philosophers and neurologists now believe that the mind and consciousness are linked with matter, that is, with the brain. The question of whether machines could one day have a mind or consciousness could at some point in the future become relevant. The mind-body problem in particular concerns whether or not the mind is bound to the body. We will not discuss these questions here. The interested reader may consult [Spe98, Spe97] and is invited, in the course of AI technology studies, to form a personal opinion about these questions.

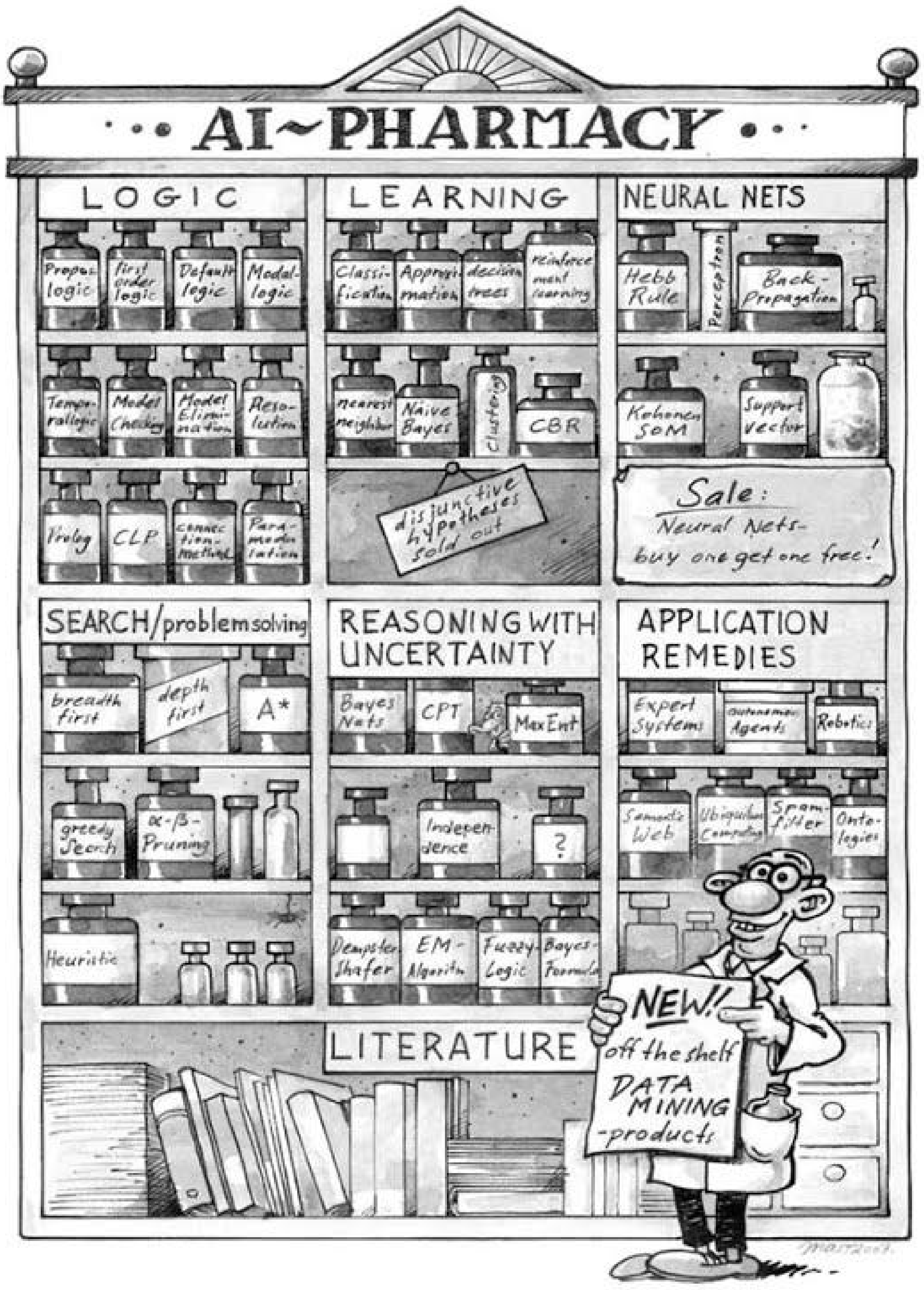


Fig. 1.2 A small sample of the solutions offered by AI

1.1.2 The Turing Test and Chatterbots

Alan Turing made a name for himself as an early pioneer of AI with his definition of an intelligent machine, in which the machine in question must pass the following test. The test person Alice sits in a locked room with two computer terminals. One terminal is connected to a machine, the other with a non-malicious person Bob. Alice can type questions into both terminals. She is given the task of deciding, after five minutes, which terminal belongs to the machine. The machine passes the test if it can trick Alice at least 30% of the time [Tur50].

While the test is very interesting philosophically, for practical AI, which deals with problem solving, it is not a very relevant test. The reasons for this are similar to those mentioned above related to Braitenberg vehicles (see Exercise 1.3 on page 14).

The AI pioneer and social critic Joseph Weizenbaum developed a program named *Eliza*, which is meant to answer a test subject's questions like a human psychologist [Wei66]. He was in fact able to demonstrate success in many cases. Supposedly his secretary often had long discussions with the program. Today in the internet there are many so-called *chatterbots*, some of whose initial responses are quite impressive. After a certain amount of time, however, their artificial nature becomes apparent. Some of these programs are actually capable of learning, while others possess extraordinary knowledge of various subjects, for example geography or software development. There are already commercial applications for chatterbots in online customer support and there may be others in the field of e-learning. It is conceivable that the learner and the e-learning system could communicate through a chatterbot. The reader may wish to compare several chatterbots and evaluate their intelligence in Exercise 1.1 on page 14.

1.2 The History of AI

AI draws upon many past scientific achievements which are not mentioned here, for AI as a science in its own right has only existed since the middle of the Twentieth Century. Table 1.1 on page 10, with the most important AI milestones, and a graphical representation of the main movements of AI in Fig. 1.3 on page 6 complement the following text.

1.2.1 The First Beginnings

In the 1930s Kurt Gödel, Alonso Church, and Alan Turing laid important foundations for logic and theoretical computer science. Of particular interest for AI are Gödel's theorems. The completeness theorem states that first-order predicate logic is complete. This means that every true statement that can be formulated in predicate logic is provable using the rules of a formal calculus. On this basis, automatic theorem provers could later be constructed as implementations of formal calculi. With the incompleteness theorem, Gödel showed that in higher-order logics there exist true statements that are unprovable.[2] With this he uncovered painful limits of formal systems.

Alan Turing's proof of the undecidability of the halting problem also falls into this time period. He showed that there is no program that can decide whether a given arbitrary program (and its respective input) will run in an infinite loop. With

[2]Higher-order logics are extensions of predicate logic, in which not only variables, but also function symbols or predicates can appear as terms in a quantification. Indeed, Gödel only showed that any system that is based on predicate logic and can formulate Peano arithmetic is incomplete.

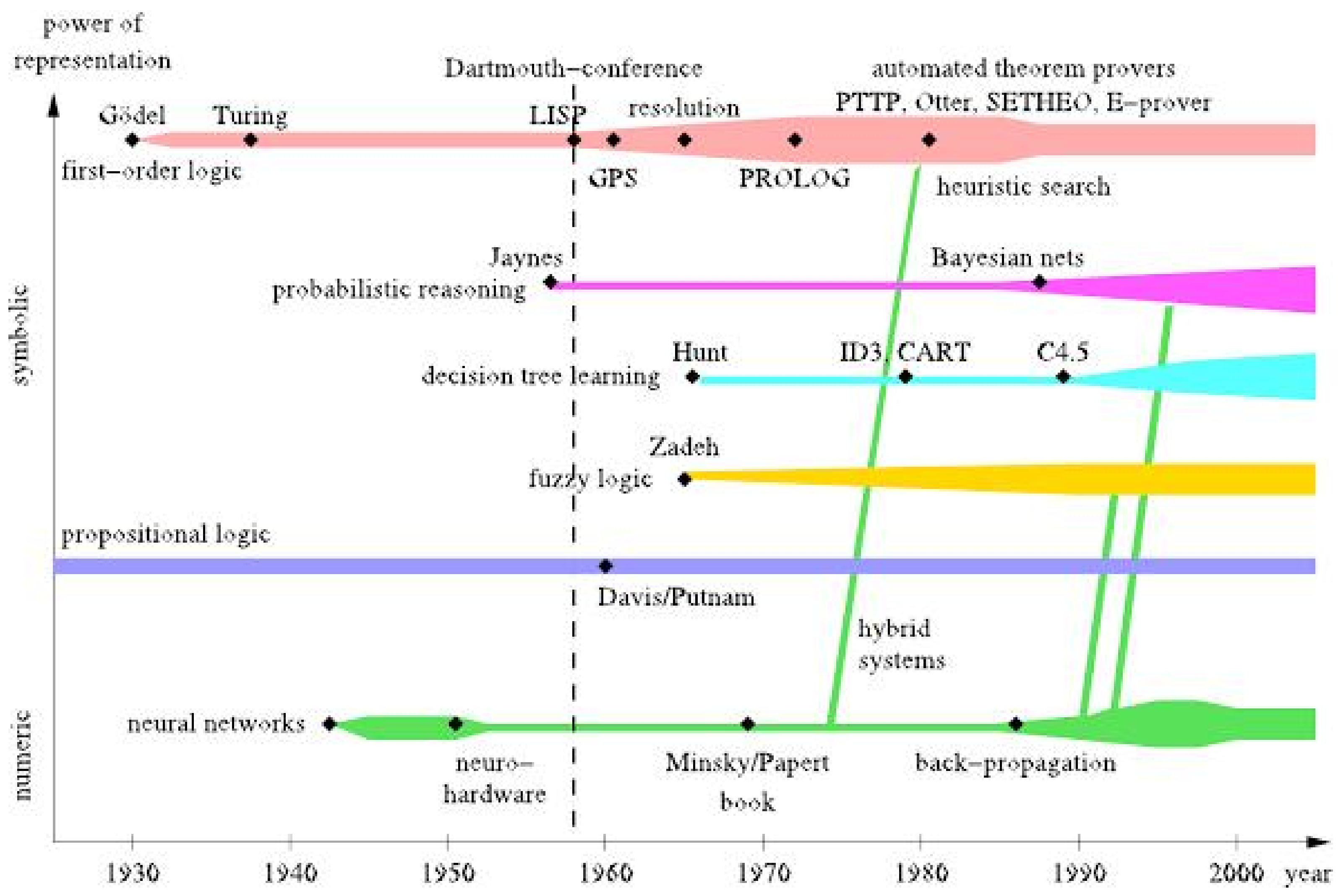


Fig. 1.3 History of the various AI areas. The width of the *bars* indicates prevalence of the method's use

this Turing also identified a limit for intelligent programs. It follows, for example, that there will never be a universal program verification system.[3]

In the 1940s, based on results from neuroscience, McCulloch, Pitts and Hebb designed the first mathematical models of neural networks. However, computers at that time lacked sufficient power to simulate simple brains.

1.2.2 Logic Solves (Almost) All Problems

AI as a practical science of thought mechanization could of course only begin once there were programmable computers. This was the case in the 1950s. Newell and Simon introduced Logic Theorist, the first automatic theorem prover, and thus also showed that with computers, which actually only work with numbers, one can also process symbols. At the same time McCarthy introduced, with the language LISP, a programming language specially created for the processing of symbolic structures. Both of these systems were introduced in 1956 at the historic Dartmouth Conference, which is considered the birthday of AI.

In the US, LISP developed into the most important tool for the implementation of symbol-processing AI systems. Thereafter the logical inference rule known as resolution developed into a complete calculus for predicate logic.

[3]This statement applies to "total correctness", which implies a proof of correct execution as well as a proof of termination for every valid input.

In the 1970s the logic programming language PROLOG was introduced as the European counterpart to LISP. PROLOG offers the advantage of allowing direct programming using Horn clauses, a subset of predicate logic. Like LISP, PROLOG has data types for convenient processing of lists.

Until well into the 1980s, a breakthrough spirit dominated AI, especially among many logicians. The reason for this was the string of impressive achievements in symbol processing. With the Fifth Generation Computer Systems project in Japan and the ESPRIT program in Europe, heavy investment went into the construction of intelligent computers.

For small problems, automatic provers and other symbol-processing systems sometimes worked very well. The combinatorial explosion of the search space, however, defined a very narrow window for these successes. This phase of AI was described in [RN10] as the "Look, Ma, no hands!" era.

Because the economic success of AI systems fell short of expectations, funding for logic-based AI research in the United States fell dramatically during the 1980s.

1.2.3 The New Connectionism

During this phase of disillusionment, computer scientists, physicists, and Cognitive scientists were able to show, using computers which were now sufficiently powerful, that mathematically modeled neural networks are capable of learning using training examples, to perform tasks which previously required costly programming. Because of the fault-tolerance of such systems and their ability to recognize patterns, considerable successes became possible, especially in pattern recognition. Facial recognition in photos and handwriting recognition are two example applications. The system Nettalk was able to learn speech from example texts [SR86]. Under the name *connectionism*, a new subdiscipline of AI was born.

Connectionism boomed and the subsidies flowed. But soon even here feasibility limits became obvious. The neural networks could acquire impressive capabilities, but it was usually not possible to capture the learned concept in simple formulas or logical rules. Attempts to combine neural nets with logical rules or the knowledge of human experts met with great difficulties. Additionally, no satisfactory solution to the structuring and modularization of the networks was found.

1.2.4 Reasoning Under Uncertainty

AI as a practical, goal-driven science searched for a way out of this crisis. One wished to unite logic's ability to explicitly represent knowledge with neural networks' strength in handling uncertainty. Several alternatives were suggested.

The most promising, *probabilistic reasoning*, works with conditional probabilities for propositional calculus formulas. Since then many diagnostic and expert systems have been built for problems of everyday reasoning using *Bayesian networks*. The success of Bayesian networks stems from their intuitive comprehensibility, the clean semantics of conditional probability, and from the centuries-old, mathematically grounded probability theory.

The weaknesses of logic, which can only work with two truth values, can be solved by *fuzzy logic*, which pragmatically introduces infinitely many values between zero and one. Though even today its theoretical foundation is not totally firm, it is being successfully utilized, especially in control engineering.

A much different path led to the successful synthesis of logic and neural networks under the name *hybrid systems*. For example, neural networks were employed to learn heuristics for reduction of the huge combinatorial search space in proof discovery [SE90].

Methods of decision tree learning from data also work with probabilities. Systems like CART, ID3 and C4.5 can quickly and automatically build very accurate decision trees which can represent propositional logic concepts and then be used as expert systems. Today they are a favorite among machine learning techniques (Sect. 8.4).

Since about 1990, *data mining* has developed as a subdiscipline of AI in the area of statistical data analysis for extraction of knowledge from large databases. Data mining brings no new techniques to AI, rather it introduces the requirement of using large databases to gain explicit knowledge. One application with great market potential is steering ad campaigns of big businesses based on analysis of many millions of purchases by their customers. Typically, machine learning techniques such as decision tree learning come into play here.

1.2.5 Distributed, Autonomous and Learning Agents

Distributed artificial intelligence, DAI, has been an active area research since about 1985. One of its goals is the use of parallel computers to increase the efficiency of problem solvers. It turned out, however, that because of the high computational complexity of most problems, the use of "intelligent" systems is more beneficial than parallelization itself.

A very different conceptual approach results from the development of autonomous software agents and robots that are meant to cooperate like human teams. As with the aforementioned Braitenberg vehicles, there are many cases in which an individual agent is not capable of solving a problem, even with unlimited resources. Only the cooperation of many agents leads to the intelligent behavior or to the solution of a problem. An ant colony or a termite colony is capable of erecting buildings of very high architectural complexity, despite the fact that no single ant comprehends how the whole thing fits together. This is similar to the situation of provisioning bread for a large city like New York [RN10]. There is no central planning agency for bread, rather there are hundreds of bakers that know their respective areas of the city and bake the appropriate amount of bread at those locations.

Active skill acquisition by robots is an exciting area of current research. There are robots today, for example, that independently learn to walk or to perform various motorskills related to soccer (Chap. 10). Cooperative learning of multiple robots to solve problems together is still in its infancy.

1.2.6 AI Grows up

The above systems offered by AI today are not a universal recipe, but a workshop with a manageable number of tools for very different tasks. Most of these tools are well-developed and are available as finished software libraries, often with convenient user interfaces. The selection of the right tool and its sensible use in each individual case is left to the AI developer or knowledge engineer. Like any other artisanship, this requires a solid education, which this book is meant to promote.

More than nearly any other science, AI is interdisciplinary, for it draws upon interesting discoveries from such diverse fields as logic, operations research, statistics, control engineering, image processing, linguistics, philosophy, psychology, and neurobiology. On top of that, there is the subject area of the particular application. To successfully develop an AI project is therefore not always so simple, but almost always extremely exciting.

1.3 Agents

Although the term *intelligent agents* is not new to AI, only in recent years has it gained prominence through [RN10], among others. *Agent* denotes rather generally a system that processes information and produces an output from an input. These agents may be classified in many different ways.

In classical computer science, *software agents* are primarily employed (Fig. 1.4 on page 11). In this case the agent consists of a program that calculates a result from user input.

In robotics, on the other hand, *hardware agents* (also called robots) are employed, which additionally have sensors and actuators at their disposal (Fig. 1.5 on page 11). The agent can perceive its environment with the sensors. With the actuators it carries out actions and changes its environment.

With respect to the intelligence of the agent, there is a distinction between *reflex agents*, which only react to input, and *agents with memory*, which can also include the past in their decisions. For example, a driving robot that through its sensors knows its exact position (and the time) has no way, as a reflex agent, of determining its velocity. If, however, it saves the position, at short, discrete time steps, it can thus easily calculate its average velocity in the previous time interval.

If a reflex agent is controlled by a deterministic program, it represents a function of the set of all inputs to the set of all outputs. An agent with memory, on the other hand, is in general not a function. Why? (See Exercise 1.5 on page 14.) Reflex agents are sufficient in cases where the problem to be solved involves a Markov decision process. This is a process in which only the current state is needed to determine the optimal next action (see Chap. 10).

A mobile robot which should move from room 112 to room 179 in a building takes actions different from those of a robot that should move to room 105. In other words, the actions depend on the goal. Such agents are called *goal-based*.

Table 1.1 Milestones in the development of AI from Gödel to today

1931	The Austrian Kurt Gödel shows that in first-order *predicate logic* all true statements are derivable [Göd31a]. In higher-order logics, on the other hand, there are true statements that are unprovable [Göd31b]. (In [Göd31b] Gödel showed that predicate logic extended with the axioms of arithmetic is incomplete.)
1937	Alan Turing points out the limits of intelligent machines with the halting problem [Tur37].
1943	McCulloch and Pitts model *neural networks* and make the connection to propositional logic.
1950	Alan Turing defines machine intelligence with the *Turing test* and writes about learning machines and genetic algorithms [Tur50].
1951	Marvin Minsky develops a neural network machine. With 3000 vacuum tubes he simulates 40 neurons.
1955	Arthur Samuel (IBM) builds a learning chess program that plays better than its developer [Sam59].
1956	McCarthy organizes a conference in Dartmouth College. Here the name *Artificial Intelligence* was first introduced.
	Newell and Simon of Carnegie Mellon University (CMU) present the *Logic Theorist*, the first symbol-processing computer program [NSS83].
1958	McCarthy invents at MIT (Massachusetts Institute of Technology) the high-level language *LISP*. He writes programs that are capable of modifying themselves.
1959	Gelernter (IBM) builds the Geometry Theorem Prover.
1961	The General Problem Solver (GPS) by Newell and Simon imitates human thought [NS61].
1963	McCarthy founds the AI Lab at Stanford University.
1965	Robinson invents the *resolution calculus* for predicate logic [Rob65] (Sect. 3.5).
1966	Weizenbaum's program Eliza carries out dialog with people in natural language [Wei66] (Sect. 1.1.2).
1969	Minsky and Papert show in their book *Perceptrons* that the perceptron, a very simple neural network, can only represent linear functions [MP69] (Sect. 1.1.2).
1972	French scientist Alain Colmerauer invents the logic programming language *PROLOG* (Chap. 5).
	British physician de Dombal develops an *expert system* for diagnosis of acute abdominal pain [dDLS+72]. It goes unnoticed in the mainstream AI community of the time (Sect. 7.3).
1976	Shortliffe and Buchanan develop MYCIN, an expert system for diagnosis of infectious diseases, which is capable of dealing with uncertainty (Chap. 7).
1981	Japan begins, at great expense, the "Fifth Generation Project" with the goal of building a powerful PROLOG machine.
1982	R1, the expert system for configuring computers, saves Digital Equipment Corporation 40 million dollars per year [McD82].
1986	Renaissance of neural networks through, among others, Rumelhart, Hinton and Sejnowski [RM86]. The system Nettalk learns to read texts aloud [SR86] (Chap. 9).
1990	Pearl [Pea88], Cheeseman [Che85], Whittaker, Spiegelhalter bring probability theory into AI with *Bayesian networks* (Sect. 7.4). Multi-agent systems become popular.
1992	Tesauros TD-gammon program demonstrates the advantages of reinforcement learning.
1993	Worldwide *RoboCup* initiative to build soccer-playing autonomous robots [Roba].

Table 1.1 (continued)

1995	From statistical learning theory, Vapnik develops support vector machines, which are very important today.
1997	IBM's chess computer Deep Blue defeats the chess world champion Gary Kasparov.
	First international RoboCup competition in Japan.
2003	The robots in RoboCup demonstrate impressively what AI and robotics are capable of achieving.
2006	Service robotics becomes a major AI research area.
2010	Autonomous robots start learning their policies.
2011	IBM's natural language understanding and question answering program "Watson" defeats two human champions in the U.S. television quiz show "Jeopardy!" (Sect. 1.4).

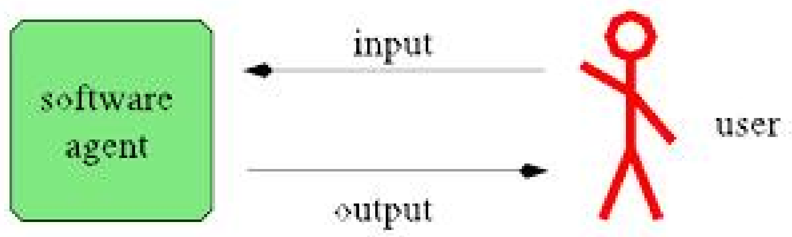


Fig. 1.4 A software agent with user interaction

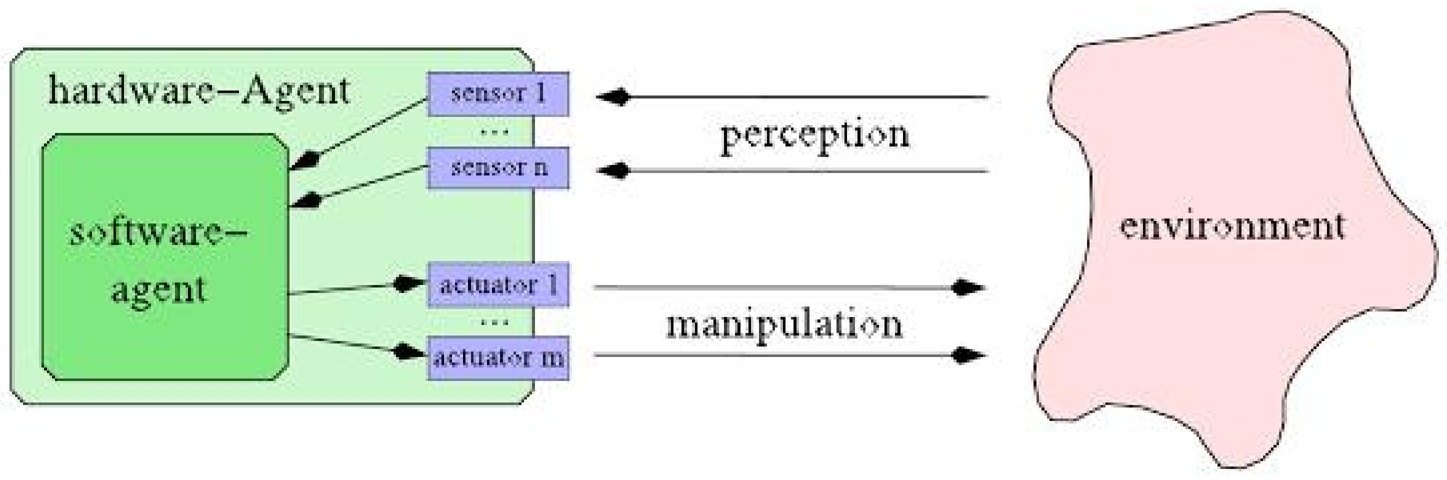


Fig. 1.5 A hardware agent

Example 1.1 A spam filter is an agent that puts incoming emails into wanted or unwanted (spam) categories, and deletes any unwanted emails. Its goal as a goal-based agent is to put all emails in the right category. In the course of this not-so-simple task, the agent can occasionally make mistakes. Because its goal is to classify all emails correctly, it will attempt to make as few errors as possible. However, that is not always what the user has in mind. Let us compare the following two agents. Out of 1,000 emails, Agent 1 makes only 12 errors. Agent 2 on the other hand makes 38 errors with the same 1,000 emails. Is it therefore worse than Agent 1? The errors of both agents are shown in more detail in the following table, the so-called "confusion matrix":

Agent 1:

		correct class	
		wanted	spam
spam filter decides	wanted	189	1
	spam	11	799

Agent 2:

		correct class	
		wanted	spam
spam filter decides	wanted	200	38
	spam	0	762

Agent 1 in fact makes fewer errors than Agent 2, but those few errors are severe because the user loses 11 potentially important emails. Because there are in this case two types of errors of differing severity, each error should be weighted with the appropriate cost factor (see Sect. 7.3.5 and Exercise 1.7 on page 14).

The sum of all weighted errors gives the total cost caused by erroneous decisions. The goal of a *cost-based agent* is to minimize the cost of erroneous decisions in the long term, that is, on average. In Sect. 7.3 we will become familiar with the diagnostic system LEXMED as an example of a cost-based agent.

Analogously, the goal of a utility-based agent is to maximize the utility derived from correct decisions in the long term, that is, on average. The sum of all decisions weighted by their respective utility factors gives the total utility.

Of particular interest in AI are *Learning agents*, which are capable of changing themselves given training examples or through positive or negative feedback, such that the average utility of their actions grows over time (see Chap. 8).

As mentioned in Sect. 1.2.5, *distributed agents* are increasingly coming into use, whose intelligence are not localized in one agent, but rather can only be seen through cooperation of many agents.

The design of an agent is oriented, along with its objective, strongly toward its *environment*, or alternately its picture of the environment, which strongly depends on it sensors. The environment is *observable* if the agent always knows the complete state of the world. Otherwise the environment is only *partially observable*. If an action always leads to the same result, then the environment is *deterministic*. Otherwise it is *nondeterministic*. In a *discrete environment* only finitely many states and actions occur, whereas a *continuous environment* boasts infinitely many states or actions.

1.4 Knowledge-Based Systems

An agent is a program that implements a mapping from perceptions to actions. For simple agents this way of looking at the problem is sufficient. For complex applications in which the agent must be able to rely on a large amount of information and is meant to do a difficult task, programming the agent can be very costly and unclear how to proceed. Here AI provides a clear path to follow that will greatly simplify the work.

First we separate *knowledge* from the system or program, which uses the knowledge to, for example, reach conclusions, answer queries, or come up with a plan. This system is called the *inference mechanism*. The knowledge is stored in a *knowledge base* (*KB*). Acquisition of knowledge in the knowledge base is denoted *Knowledge Engineering* and is based on various knowledge sources such as human experts, the knowledge engineer, and databases. Active learning systems can also acquire knowledge though active exploration of the world (see Chap. 10). In Fig. 1.6 on page 13 the general architecture of knowledge-based systems is presented.

Moving toward a separation of knowledge and inference has several crucial advantages. The separation of knowledge and inference can allow inference systems to be implemented in a largely application-independent way. For example, it is much

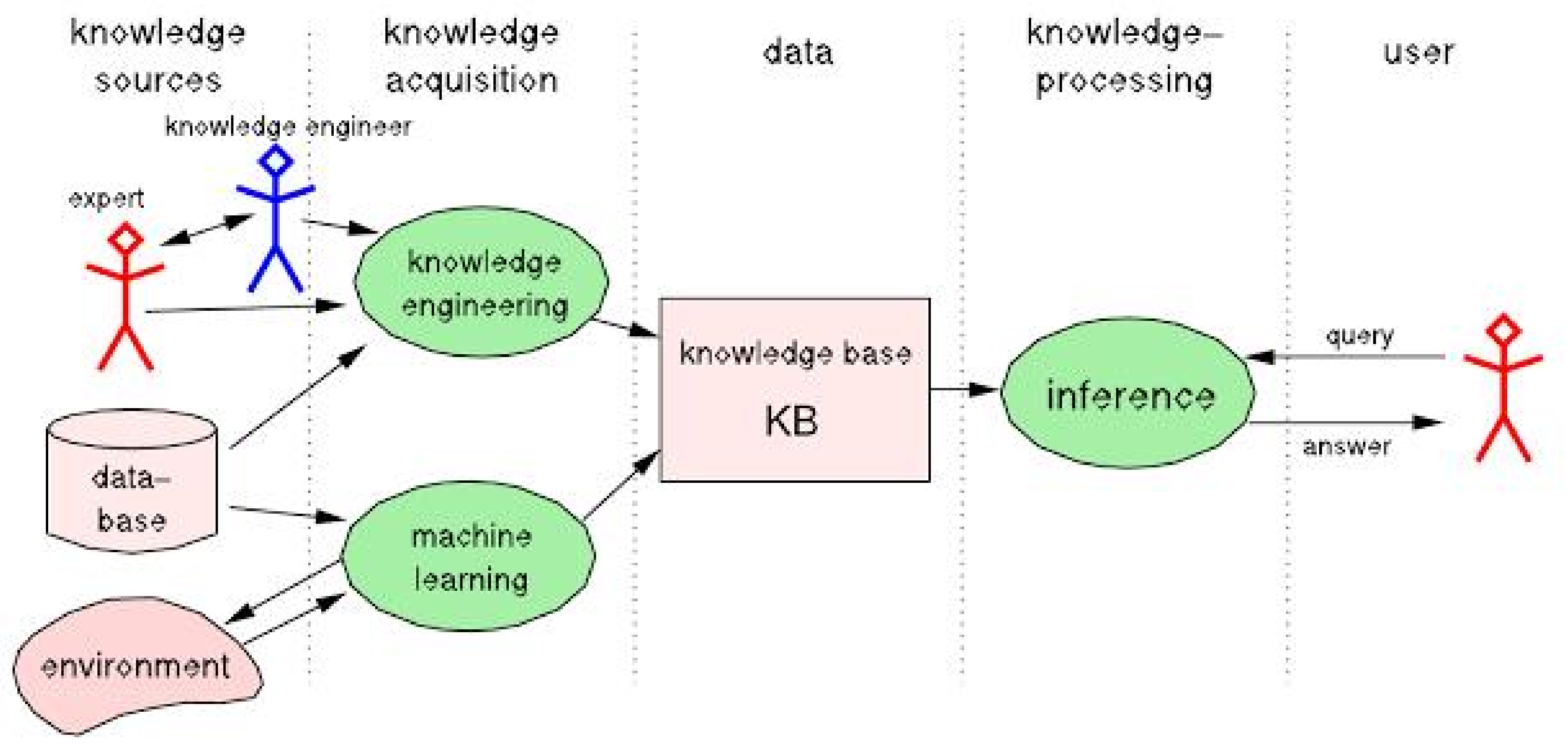


Fig. 1.6 Structure of a classic knowledge-processing system

easier to replace the knowledge base of a medical expert system than to program a whole new system.

Through the decoupling of the knowledge base from inference, knowledge can be stored declaratively. In the knowledge base there is only a description of the knowledge, which is independent from the inference system in use. Without this clear separation, knowledge and processing of inference steps would be interwoven, and any changes to the knowledge would be very costly.

Formal language as a convenient interface between man and machine lends itself to the representation of knowledge in the knowledge base. In the following chapters we will get to know a whole series of such languages. First, in Chaps. 2 and 3 there are *propositional calculus* and *first-order predicate logic* (PL1). But other formalisms such as probabilistic logic, fuzzy logic or decision trees are also presented. We start with propositional calculus and the related inference systems. Building on that, we will present predicate logic, a powerful language that is accessible by machines and very important in AI.

As an example for a large scale knowledge based system we want to refer to the software agent "Watson". Developed at IBM together with a number of universities, Watson is a question answering program, that can be fed with clues given in natural language. It works on a knowledge base comprising four terabytes of hard disk storage, including the full text of Wikipedia [FNA+09]. Watson was developed within IBM's DeepQA project which is characterized in [Dee11] as follows:

> The DeepQA project at IBM shapes a grand challenge in Computer Science that aims to illustrate how the wide and growing accessibility of natural language content and the integration and advancement of Natural Language Processing, Information Retrieval, Machine Learning, Knowledge Representation and Reasoning, and massively parallel computation can drive open-domain automatic Question Answering technology to a point where it clearly and consistently rivals the best human performance.

In the U.S. television quiz show "Jeopardy!", in February 2011, Watson defeated the two human champions Brad Rutter and Ken Jennings in a two-game, combined-point match and won the one million dollar price. One of Watson's particular strengths was its very fast reaction to the questions with the result that Watson often hit the buzzer (using a solenoid) faster than its human competitors and then was able to give the first answer to the question.

The high performance and short reaction times of Watson were due to an implementation on 90 IBM Power 750 servers, each of which contains 32 processors, resulting in 2880 parallel processors.

1.5 Exercises

Exercise 1.1 Test some of the chatterbots available on the internet. Start for example with www.hs-weingarten.de/~ertel/aibook in the collection of links under Turingtest/Chatterbots, or at www.simonlaven.com or www.alicebot.org. Write down a starting question and measure the time it takes, for each of the various programs, until you know for certain that it is not a human.

❄❄ **Exercise 1.2** At www.pandorabots.com you will find a server on which you can build a chatterbot with the markup language AIML quite easily. Depending on your interest level, develop a simple or complex chatterbot, or change an existing one.

Exercise 1.3 Give reasons for the unsuitability of the Turing test as a definition of "*artificial intelligence*" in practical AI.

➼ **Exercise 1.4** Many well-known inference processes, learning processes, etc. are NP-complete or even undecidable. What does this mean for AI?

Exercise 1.5
(a) Why is a deterministic agent with memory not a function from the set of all inputs to the set of all outputs, in the mathematical sense?
(b) How can one change the agent with memory, or model it, such that it becomes equivalent to a function but does not lose its memory?

Exercise 1.6 Let there be an agent with memory that can move within a plane. From its sensors, it receives at clock ticks of a regular interval Δt its exact position (x, y) in Cartesian coordinates.
(a) Give a formula with which the agent can calculate its velocity from the current time t and the previous measurement of $t - \Delta t$.
(b) How must the agent be changed so that it can also calculate its acceleration? Provide a formula here as well.

❄ **Exercise 1.7**
(a) Determine for both agents in Example 1.1 on page 11 the costs created by the errors and compare the results. Assume here that having to manually delete a spam email costs one cent and retrieving a deleted email, or the loss of an email, costs one dollar.
(b) Determine for both agents the profit created by correct classifications and compare the results. Assume that for every desired email recognized, a profit of one dollar accrues and for every correctly deleted spam email, a profit of one cent.

Propositional Logic

2

In propositional logic, as the name suggests, propositions are connected by logical operators. The statement "*the street is wet*" is a proposition, as is "*it is raining*". These two propositions can be connected to form the new proposition

if it is raining the street is wet.

Written more formally

it is raining $\Rightarrow$ *the street is wet.*

This notation has the advantage that the elemental propositions appear again in unaltered form. So that we can work with propositional logic precisely, we will begin with a definition of the set of all propositional logic formulas.

2.1 Syntax

Definition 2.1 Let $Op = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$ be the set of logical operators and Σ a set of symbols. The sets Op, Σ and $\{t, f\}$ are pairwise disjoint. Σ is called the *signature* and its elements are the *proposition variables*. The set of propositional logic formulas is now recursively defined:

- t and f are (atomic) formulas.
- All proposition variables, that is all elements from Σ, are (atomic) formulas.
- If A and B are formulas, then $\neg A,\ (A),\ A \wedge B,\ A \vee B,\ A \Rightarrow B,\ A \Leftrightarrow B$ are also formulas.

This elegant recursive definition of the set of all formulas allows us to generate infinitely many formulas. For example, given $\Sigma = \{A, B, C\}$,

$$A \wedge B, \quad A \wedge B \wedge C, \quad A \wedge A \wedge A, \quad C \wedge B \vee A, \quad (\neg A \wedge B) \Rightarrow (\neg C \vee A)$$

are formulas. $(((A)) \vee B)$ is also a syntactically correct formula.

W. Ertel, *Introduction to Artificial Intelligence*,
Undergraduate Topics in Computer Science,
DOI 10.1007/978-0-85729-299-5_2, © Springer-Verlag London Limited 2011

Definition 2.2 We read the symbols and operators in the following way:

t:	"true"	
f:	"false"	
$\neg A$:	"not A"	(negation)
$A \wedge B$:	"A and B"	(conjunction)
$A \vee B$:	"A or B"	(disjunction)
$A \Rightarrow B$:	"if A then B"	(implication (also called *material implication*))
$A \Leftrightarrow B$:	"A if and only if B"	(equivalence)

The formulas defined in this way are so far purely syntactic constructions without meaning. We are still missing the semantics.

2.2 Semantics

In propositional logic there are two truth values: t for "true" and f for "false". We begin with an example and ask ourselves whether the formula $A \wedge B$ is true. The answer is: it depends on whether the variables A and B are true. For example, if A stands for "*It is raining today*" and B for "*It is cold today*" and these are both true, then $A \wedge B$ is true. If, however, B represents "*It is hot today*" (and this is false), then $A \wedge B$ is false.

We must obviously assign truth values that reflect the state of the world to proposition variables. Therefore we define

Definition 2.3 A mapping $I : \Sigma \rightarrow \{w, f\}$, which assigns a truth value to every proposition variable, is called an *interpretation*.

Because every proposition variable can take on two truth values, every propositional logic formula with n different variables has 2^n different interpretations. We define the truth values for the basic operations by showing all possible interpretations in a *truth table* (see Table 2.1 on page 17).

The empty formula is true for all interpretations. In order to determine the truth value for complex formulas, we must also define the order of operations for logical operators. If expressions are parenthesized, the term in the parentheses is evaluated

Table 2.1 Definition of the logical operators by truth table

A	B	(A)	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
t	t	t	f	t	t	t	t
t	f	t	f	f	t	f	f
f	t	f	t	f	t	t	f
f	f	f	t	f	f	t	t

first. For unparenthesized formulas, the priorities are ordered as follows, beginning with the strongest binding: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

To clearly differentiate between the equivalence of formulas and syntactic equivalence, we define

Definition 2.4 Two formulas F and G are called semantically equivalent if they take on the same truth value for all interpretations. We write $F \equiv G$.

Semantic equivalence serves above all to be able to use the meta-language, that is, natural language, to talk about the object language, namely logic. The statement "$A \equiv B$" conveys that the two formulas A and B are semantically equivalent. The statement "$A \Leftrightarrow B$" on the other hand is a syntactic object of the formal language of propositional logic.

According to how many interpretations in which a formula is true, we can divide formulas into the following classes:

Definition 2.5 A formula is called
- *Satisfiable* if it is true for at least one interpretation.
- *Logically valid* or simply *valid* if it is true for all interpretations. True formulas are also called *tautologies*.
- *Unsatisfiable* if it is not true for any interpretation.

Every interpretation that satisfies a formula is called a *model* of the formula.

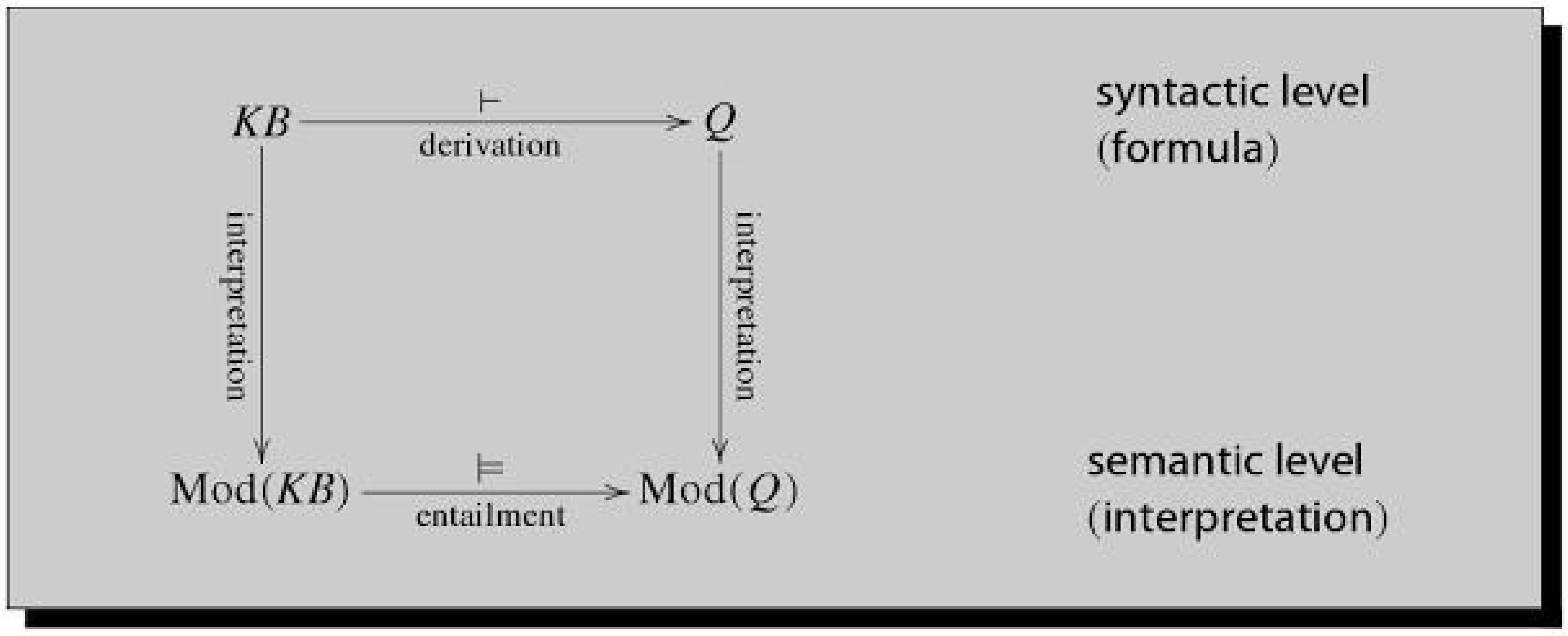
Clearly the negation of every generally valid formula is unsatisfiable. The negation of a satisfiable, but not generally valid formula F is satisfiable.

We are now able to create truth tables for complex formulas to ascertain their truth values. We put this into action immediately using equivalences of formulas which are important in practice.

image not available

image not available

image
not
available



$$\begin{tikzcd} KB \arrow[r, "\vdash", "\text{derivation}"'] \arrow[d, "\text{interpretation}"'] & Q \arrow[d, "\text{interpretation}"] \\ \mathrm{Mod}(KB) \arrow[r, "\models", "\text{entailment}"'] & \mathrm{Mod}(Q) \end{tikzcd}$$

Fig. 2.1 Syntactic derivation and semantic entailment. $\mathrm{Mod}(X)$ represents the set of models of a formula X

and complete, then syntactic derivation and semantic entailment are two equivalent relations (see Fig. 2.1).

To keep automatic proof systems as simple as possible, these are usually made to operate on formulas in conjunctive normal form.

Definition 2.8 A formula is in *conjunctive normal form* (*CNF*) if and only if it consists of a *conjunction*

$$K_1 \wedge K_2 \wedge \cdots \wedge K_m$$

of clauses. A clause K_i consists of a *disjunction*

$$(L_{i1} \vee L_{i2} \vee \cdots \vee L_{in_i})$$

of literals. Finally, a *literal* is a variable (positive literal) or a negated variable (negative literal).

The formula $(A \vee B \vee \neg C) \wedge (A \vee B) \wedge (\neg B \vee \neg C)$ is in conjunctive normal form. The conjunctive normal form does not place a restriction on the set of formulas because:

Theorem 2.4 *Every propositional logic formula can be transformed into an equivalent conjunctive normal form.*

image not available

image not available

image
not
available

image
not
available

image not available

2.7 Applications and Limitations

Theorem provers for propositional logic are part of the developer's everyday toolset in digital technology. For example, the verification of digital circuits and the generation of test patterns for testing of microprocessors in fabrication are some of these tasks. Special proof systems that work with binary decision diagrams (BDD) () are also employed as a data structure for processing propositional logic formulas.

In AI, propositional logic is employed in simple applications. For example, simple expert systems can certainly work with propositional logic. However, the variables must all be discrete, with only a few values, and there may not be any cross-relations between variables. Complex logical connections can be expressed much more elegantly using predicate logic.

Probabilistic logic is a very interesting and current combination of propositional logic and probabilistic computation that allows modeling of uncertain knowledge. It is handled thoroughly in Chap. 7. Fuzzy logic, which allows infinitely many truth values, is also discussed in that chapter.

2.8 Exercises

➵ **Exercise 2.1** Give a Backus–Naur form grammar for the syntax of propositional logic.

Exercise 2.2 Show that the following formulas are tautologies:
(a) $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
(b) $A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A$
(c) $((A \Rightarrow B) \wedge (B \Rightarrow A)) \Leftrightarrow (A \Leftrightarrow B)$
(d) $(A \vee B) \wedge (\neg B \vee C) \Rightarrow (A \vee C)$

Exercise 2.3 Transform the following formulas into conjunctive normal form:
(a) $A \Leftrightarrow B$
(b) $A \wedge B \Leftrightarrow A \vee B$
(c) $A \wedge (A \Rightarrow B) \Rightarrow B$

Exercise 2.4 Check the following statements for satisfiability or validity.
(a) (play_lottery $\wedge$ six_right) $\Rightarrow$ winner
(b) (play_lottery $\wedge$ six_right $\wedge$ (six_right $\Rightarrow$ win)) $\Rightarrow$ win
(c) $\neg$($\neg$gas_in_tank $\wedge$ (gas_in_tank $\vee$ $\neg$car_starts) $\Rightarrow$ $\neg$car_starts)

❄❄ **Exercise 2.5** Using the programming language of your choice, program a theorem prover for propositional logic using the truth table method for formulas in conjunctive normal form. To avoid a costly syntax check of the formulas, you may represent clauses as lists or sets of literals, and the formulas as lists or sets of clauses. The program should indicate whether the formula is unsatisfiable, satisfiable, or true, and output the number of different interpretations and models.

image not available

Table 3.1 Examples of formulas in first-order predicate logic. Please note that *mother* here is a function symbol

Formula	Description
$\forall x\ frog(x) \Rightarrow green(x)$	All frogs are green
$\forall x\ frog(x) \wedge brown(x) \Rightarrow big(x)$	All brown frogs are big
$\forall x\ likes(x, cake)$	Everyone likes cake
$\neg\forall x\ likes(x, cake)$	Not everyone likes cake
$\neg\exists x\ likes(x, cake)$	No one likes cake
$\exists x\ \forall y\ likes(y, x)$	There is something that everyone likes
$\exists x\ \forall y\ likes(x, y)$	There is someone who likes everything
$\forall x\ \exists y\ likes(y, x)$	Everything is loved by someone
$\forall x\ \exists y\ likes(x, y)$	Everyone likes something
$\forall x\ customer(x) \Rightarrow likes(bob, x)$	Bob likes every customer
$\exists x\ customer(x) \wedge likes(x, bob)$	There is a customer whom bob likes
$\exists x\ baker(x) \wedge \forall y\ customer(y) \Rightarrow mag(x, y)$	There is a baker who likes all of his customers
$\forall x\ older(mother(x), x)$	Every mother is older than her child
$\forall x\ older(mother(mother(x)), x)$	Every grandmother is older than her daughter's child
$\forall x\ \forall y\ \forall z\ rel(x, y) \wedge rel(y, z) \Rightarrow rel(x, z)$	*rel* is a transitive relation

3.2 Semantics

In propositional logic, every variable is directly assigned a truth value by an interpretation. In predicate logic, the meaning of formulas is recursively defined over the construction of the formula, in that we first assign constants, variables, and function symbols to objects in the real world.

Definition 3.3 An *interpretation* $\mathbb{I}$ is defined as

- A mapping from the set of constants and variables $K \cup V$ to a set W of names of objects in the world.
- A mapping from the set of function symbols to the set of functions in the world. Every n-place function symbol is assigned an n-place function.
- A mapping from the set of predicate symbols to the set of relations in the world. Every n-place predicate symbol is assigned an n-place relation.

Example 3.1 Let c_1, c_2, c_3 be constants, "*plus*" a two-place function symbol, and "*gr*" a two-place predicate symbol. The truth of the formula

$$F \equiv gr(plus(c_1, c_3), c_2)$$

image not available

image
not
available

Definition 3.5 We write $\varphi[x/t]$ for the formula that results when we replace every free occurrence of the variable x in φ with the term t. Thereby we do not allow any variables in the term t that are quantified in φ. In those cases variables must be renamed to ensure this.

Example 3.3 If, in the formula $\forall x\ x = y$, the free variable y is replaced by the term $x + 1$, the result is $\forall x\ x = x + 1$. With correct substitution we obtain the formula $\forall x\ x = y + 1$, which has a very different semantic.

3.3 Quantifiers and Normal Forms

By Definition 3.4 on page 34, the formula $\forall x\ p(x)$ is true if and only if it is true for all interpretations of the variable x. Instead of the quantifier, one could write $p(a_1) \wedge \cdots \wedge p(a_n)$ for all constants $a_1 \cdots a_n$ in K. For $\exists x\ p(x)$ one could write $p(a_1) \vee \cdots \vee p(a_n)$. From this it follows with de Morgan's law that

$$\forall x\ \varphi \equiv \neg\exists x \neg\varphi.$$

Through this equivalence, universal, and existential quantifiers are mutually replaceable.

Example 3.4 The proposition "*Everyone wants to be loved*" is equivalent to the proposition "*Nobody does not want to be loved*".

Quantifiers are an important component of predicate logic's expressive power. However, they are disruptive for automatic inference in AI because they make the structure of formulas more complex and increase the number of applicable inference rules in every step of a proof. Therefore our next goal is to find, for every predicate logic formula, an equivalent formula in a standardized normal form with as few quantifiers as possible. As a first step we bring universal quantifiers to the beginning of the formula and thus define

Definition 3.6 A predicate logic formula φ is in *prenex normal form* if it holds that

- $\varphi = Q_1 x_1 \cdots Q_n x_n\ \psi$.
- ψ is a quantifierless formula.
- $Q_i \in \{\forall, \exists\}$ for $i = 1, \ldots, n$.

image not available

image not available

Table 3.2 Simple proof with modus ponens and quantifier elimination

WB:	1	$child(eve, anne, oscar)$
WB:	2	$\forall x\ \forall y\ \forall z\ child(x, y, z) \Rightarrow child(x, z, y)$
$\forall E(2) : x/eve, y/anne, z/oscar$	3	$child(eve, anne, oscar) \Rightarrow child(eve, oscar, anne)$
$MP(1, 3)$	4	$child(eve, oscar, anne)$

or less intuitive and the calculi work on arbitrary PL1 formulas. In the next section we will primarily concentrate on the resolution calculus, which is in practice the most important efficient, automatizable calculus for formulas in conjunctive normal form. Here, using Example 3.2 on page 35 we will give a very small "natural" proof. We use the inference rule

$$\frac{A, \quad A \Rightarrow B}{B} \quad \text{(modus ponens, MP)} \quad \text{and} \quad \frac{\forall x\ A}{A[x/t]} \quad (\forall\text{-elimination}, \forall E).$$

The modus ponens is already familiar from propositional logic. When eliminating universal quantifiers one must keep in mind that the quantified variable x must be replaced by a ground term t, meaning a term that contains no variables. The proof of $child(eve, oscar, anne)$ from an appropriately reduced knowledge base is presented in Table 3.2.

The two formulas of the reduced knowledge base are listed in rows 1 and 2. In row 3 the universal quantifiers from row 2 are eliminated, and in row 4 the claim is derived with modus ponens.

The calculus consisting of the two given inference rules is not complete. However, it can be extended into a complete procedure by addition of further inference rules. This nontrivial fact is of fundamental importance for mathematics and AI. The Austrian logician Kurt Gödel proved in 1931 that [Göd31a]

Theorem 3.3 (Gödel's completeness theorem) *First-order predicate logic is complete. That is, there is a calculus with which every proposition that is a consequence of a knowledge base KB can be proved. If $KB \models \varphi$, then it holds that $KB \vdash \varphi$.*

Every true proposition in first-order predicate logic is therefore provable. But is the reverse also true? Is everything we can derive syntactically actually true? The answer is "yes":

Theorem 3.4 (Correctness) *There are calculi with which only true propositions can be proved. That is, if $KB \vdash \varphi$ holds, then $KB \models \varphi$.*

image
not
available

image not available

Theorem 3.5 *The resolution rule is correct. That is, the resolvent is a semantic consequence of the two parent clauses.*

For Completeness, however, we still need a small addition, as is shown in the following example.

Example 3.7 The famous Russell paradox reads "*There is a barber who shaves everyone who does not shave himself.*" This statement is contradictory, meaning it is unsatisfiable. We wish to show this with resolution. Formalized in PL1, the paradox reads

$$\forall x \; shaves(barber, x) \Leftrightarrow \neg shaves(x, x)$$

and transformation into clause form yields (see Exercise 3.6 on page 55)

$$(\neg shaves(barbier, x) \vee \neg shaves(x, x))_1 \wedge (shaves(barbier, x) \vee shaves(x, x))_2. \tag{3.7}$$

From these two clauses we can derive several tautologies, but no contradiction. Thus resolution is not complete. We need yet a further inference rule.

Definition 3.9 *Factorization* of a clause is accomplished by

$$\frac{(A_1 \vee A_2 \vee \cdots \vee A_n) \quad \sigma(A_1) = \sigma(A_2)}{(\sigma(A_2) \vee \cdots \vee \sigma(A_n))},$$

where σ is the MGU of A_1 and A_2.

Now a contradiction can be derived from (3.7)

$$\begin{aligned} \text{Fak}(1, \sigma : x/barber): &\quad (\neg shaves(barber, barber))_3 \\ \text{Fak}(2, \sigma : x/barber): &\quad (shaves(barber, barber))_4 \\ \text{Res}(3, 4): &\quad ()_5 \end{aligned}$$

and we assert:

Theorem 3.6 *The resolution rule* (3.6) *together with the factorization rule* (3.9) *is refutation complete. That is, by application of factorization and resolution steps, the empty clause can be derived from any unsatisfiable formula in conjunctive normal form.*

image
not
available

image
not
available

Definition 3.10 A structure $(M, \cdot)$ consisting of a set M with a two-place inner operation "$\cdot$" is called a semigroup if the law of associativity

$$\forall x\ \forall y\ \forall z\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

holds. An element $e \in M$ is called left-neutral (right-neutral) if $\forall x\ e \cdot x = x$ $(\forall x\ x \cdot e = x)$.

It remains to be shown that

Theorem 3.7 *If a semigroup has a left-neutral element e_l and a right-neutral element e_r, then $e_l = e_r$.*

First we prove the theorem semi-formally by intuitive mathematical reasoning. Clearly it holds for all $x \in M$ that

$$e_l \cdot x = x \tag{3.8}$$

and

$$x \cdot e_r = x. \tag{3.9}$$

If we set $x = e_r$ in (3.8) and $x = e_l$ in (3.9), we obtain the two equations $e_l \cdot e_r = e_r$ and $e_l \cdot e_r = e_l$. Joining these two equations yields

$$e_l = e_l \cdot e_r = e_r,$$

which we want to prove. In the last step, incidentally, we used the fact that equality is symmetric and transitive.

Before we apply the automated prover, we carry out the resolution proof manually. First we formalize the negated query and the knowledge base KB, consisting of the axioms as clauses in conjunctive normal form:

$$\begin{array}{ll}
(\neg\, e_l = e_r)_1 & \text{negated query} \\
(m(m(x, y), z) = m(x, m(y, z)))_2 & \\
(m(e_l, x) = x)_3 & \\
(m(x, e_r) = x)_4 & \\
 & \\
\text{equality axioms:} & \\
(x = x)_5 & \text{(reflexivity)} \\
(\neg\, x = y \vee y = x)_6 & \text{(symmetry)} \\
(\neg\, x = y \vee \neg\, y = z \vee x = z)_7 & \text{(transitivity)} \\
(\neg\, x = y \vee m(x, z) = m(y, z))_8 & \text{substitution in } m \\
(\neg\, x = y \vee m(z, x) = m(z, y))_9 & \text{substitution in } m,
\end{array}$$

image
not
available

$\text{ROTATE}(l : \mathit{List})\ l' : \mathit{List}$
pre *true*
post
$(l = [] \Rightarrow l' = []) \wedge$
$(l \neq [] \Rightarrow l' = (\text{tail}\, l)\hat{}[\text{head}\, l])$

$\text{SHUFFLE}(x : \mathit{List})\ x' : \mathit{List}$
pre *true*
post $\forall i : \mathit{Item} \cdot$
$(\exists x_1, x_2 : \mathit{List} \cdot x = x_1\hat{}[i]\hat{}x_2 \Leftrightarrow$
$\exists y_1, y_2 : \mathit{List} \cdot x' = y_1\hat{}[i]\hat{}y_2)$

Here "ˆ" stands for the concatenation of lists, and "·" separates quantifiers with their variables from the rest of the formula. The functions "head l" and "tail l" choose the first element and the rest from the list, respectively. The specification of SHUFFLE indicates that every list element i that was in the list (x) before the application of SHUFFLE must be in the result (x') after the application, and vice versa. It must now be shown that the formula $(PRE_Q \Rightarrow PRE_M) \wedge (POST_M \Rightarrow POST_Q)$ is a consequence of the knowledge base containing a description of the data type *List*. The two VDM-SL specifications yield the proof task

$$\begin{aligned}
&\forall l, l', x, x' : \mathit{List} \cdot (l = x \wedge l' = x' \wedge (w \Rightarrow w)) \wedge \\
&\quad (l = x \wedge l' = x' \wedge ((l = [] \Rightarrow l' = []) \wedge (l \neq [] \Rightarrow l' = (tl\, l)\hat{}[hd\, l]) \\
&\quad \Rightarrow \forall i : \mathit{Item} \cdot (\exists x_1, x_2 : \mathit{List} \cdot x = x_1\hat{}[i]\hat{}x_2 \Leftrightarrow \exists y_1, y_2 : \mathit{List} \cdot x' = y_1\hat{}[i]\hat{}y_2))),
\end{aligned}$$

which can then be proven with the prover SETHEO.

In the coming years the *semantic web* will likely represent an important application of PL1. The content of the World Wide Web is supposed to become interpretable not only for people, but for machines. To this end web sites are being furnished with a description of their semantics in a formal description language. The search for information in the web will thereby become significantly more effective than today, where essentially only text building blocks are searchable.

Decidable subsets of predicate logic are used as description languages. The development of efficient calculi for reasoning is very important and closely connected to the description languages. A query for a future semantically operating search engine could (informally) read: Where in Switzerland next Sunday at elevations under 2000 meters will there be good weather and optimally prepared ski slopes? To answer such a question, a calculus is required that is capable of working very quickly on large sets of facts and rules. Here, complex nested function terms are less important.

As a basic description framework, the World Wide Web Consortium developed the language RDF (Resource Description Framework). Building on RDF, the significantly more powerful language OWL (Web Ontology Language) allows the description of relations between objects and classes of objects, similarly to PL1 [SET09]. *Ontologies* are descriptions of relationships between possible objects.

image
not
available

image
not
available

image
not
available

Fig. 4.1 Possible consequences of the explosion of a search space

to the goal. Heuristic search is important not only to logic, but generally to problem solving in AI and will therefore be thoroughly handled in Chap. 6.

An interesting approach, which has been pursued since about 1990, is the application of machine learning techniques to the learning of heuristics for directing the search of inference systems, which we will briefly sketch now. A resolution prover has, during the search for a proof, hundreds or more possibilities for resolution steps at each step, but only a few lead to the goal. It would be ideal if the prover could ask an oracle which two clauses it should use in the next step to quickly find the proof. There are attempts to build such proof-directing modules, which evaluate the various alternatives for the next step and then choose the alternative with the best rating. In the case of resolution, the rating of the available clauses could be computed by a function that calculates a value based on the number of positive literals, the complexity of the terms, etc., for every pair of resolvable clauses.

How can this function be implemented? Because this knowledge is "intuitive", the programmer is not familiar with it. Instead, one tries to copy nature and uses machine learning algorithms to learn from successful proofs [ESS89, SE90]. The attributes of all clause pairs participating in successful resolution steps are stored as positive, and the attributes of all unsuccessful resolutions are stored as negative. Then, using this training data and a machine learning system, a program is generated which can rate clause pairs heuristically (see Sect. 9.5).

image
not
available

If a set of formulas is extended, then, after the extension, all previously derivable statements can still be proved, and additional statements can potentially also be proved. The set of provable statements thus grows monotonically when the set of formulas is extended. For our example this means that the extension of the knowledge base will never lead to our goal. We thus modify *KB* by replacing the obviously false statement "*(all) birds can fly*" with the more exact statement "*(all) birds except penguins can fly*" and obtain as KB_2 the following clauses:

$$\begin{array}{l} penguin(tweety) \\ penguin(x) \Rightarrow bird(x) \\ bird(x) \wedge \neg penguin(x) \Rightarrow fly(x) \\ penguin(x) \Rightarrow \neg fly(x) \end{array}$$

Now the world is apparently in order again. We can derive $\neg fly(tweety)$, but not $fly(tweety)$, because for that we would need $\neg penguin(x)$, which, however, is not derivable. As long as there are only penguins in this world, peace reigns. Every normal bird, however, immediately causes problems. We wish to add the raven Abraxas (from the German book "The Little Witch") and obtain

$$\begin{array}{l} raven(abraxas) \\ raven(x) \Rightarrow bird(x) \\ penguin(tweety) \\ penguin(x) \Rightarrow bird(x) \\ bird(x) \wedge \neg penguin(x) \Rightarrow fly(x) \\ penguin(x) \Rightarrow \neg fly(x) \end{array}$$

We cannot say anything about the flight attributes of Abraxas because we forgot to formulate that ravens are not penguins. Thus we extend KB_3 to KB_4:

$$\begin{array}{l} raven(abraxas) \\ raven(x) \Rightarrow bird(x) \\ raven(x) \Rightarrow \neg pinguin(x) \\ penguin(tweety) \\ penguin(x) \Rightarrow bird(x) \\ bird(x) \wedge \neg penguin(x) \Rightarrow fly(x) \\ penguin(x) \Rightarrow \neg fly(x) \end{array}$$

The fact that ravens are not penguins, which is self-evident to humans, must be explicitly added here. For the construction of a knowledge base with all 9,800 or so types of birds worldwide, it must therefore be specified for every type of bird (except for penguins) that it is not a member of penguins. We must proceed analogously for all other exceptions such as the ostrich.

For every object in the knowledge base, in addition to its attributes, all of the attributes it does not have must be listed.

image
not
available

image not available

image not available

image
not
available

image
not
available

image
not
available

```
start :-
  fd_domain([Mayer, Hoover, Miller, Smith],1,4),
  fd_all_different([Mayer, Miller, Hoover, Smith]),

  fd_domain([German, English, Math, Physics],1,4),
  fd_all_different([German, English, Math, Physics]),

  fd_labeling([Mayer, Hoover, Miller, Smith]),

  Mayer #\=4,                     % Mayer not in room 4
  Miller #= German,               % Miller tests German
  dist(Miller,Smith) #>= 2,       % Distance Miller/Smith >= 2
  Hoover #= Math,                 % Hoover tests mathematics
  Physics #= 4,                   % Physics in room 4
  German #\=1,                    % German not in room 1
  English #\=1,                   % English not in room 1
  nl,
  write([Mayer, Hoover, Miller, Smith]), nl,
  write([German, English, Math, Physics]), nl.
```

Fig. 5.5 CLP program for the room scheduling problem

Represented somewhat more conveniently, we have the following room schedule:

Room num.	1	2	3	4
Teacher	Hoover	Miller	Mayer	Smith
Subject	Math	German	English	Physics

GNU-PROLOG has, like most other CLP languages, a so-called *finite domain constraint solver*, with which variables can be assigned a finite range of integers. This need not necessarily be an interval as in the example. We can also input a list of values. As an exercise the user is invited, in Exercise 5.9 on page 81, to create a CLP program, for example with GNU-PROLOG, for a not-so-simple logic puzzle. This puzzle, supposedly created by Einstein, can very easily be solved with a CLP system. If we tried using PROLOG without constraints, on the other hand, we could easily grind our teeth out. Anyone who finds an elegant solution with PROLOG or a prover, please let it find its way to the author.

5.8 Summary

Unification, lists, declarative programming, and the relational view of procedures, in which an argument of a predicate can act as both input and output, allow the development of short, elegant programs for many problems. Many programs would be significantly longer and thus more difficult to understand if written in a procedural language. Furthermore, these language features save the programmer time. Therefore PROLOG is also an interesting tool for rapid prototyping, particularly for AI

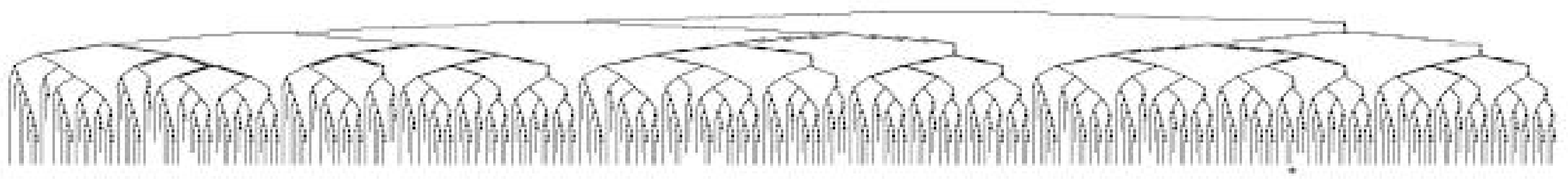
image not available

image
not
available

Search, Games and Problem Solving 6

6.1 Introduction

The search for a solution in an extremely large search tree presents a problem for nearly all inference systems. From the starting state there are many possibilities for the first inference step. For each of these possibilities there are again many possibilities in the next step, and so on. Even in the proof of a very simple formula from [Ert93] with three Horn clauses, each with at most three literals, the search tree for SLD resolution has the following shape:



The tree was cut off at a depth of 14 and has a solution in the leaf node marked by $*$. It is only possible to represent it at all because of the small branching factor of at most two and a cutoff at depth 14. For realistic problems, the branching factor and depth of the first solution may become significantly bigger.

Assume the branching factor is a constant equal to 30 and the first solution is at depth 50. The search tree has $30^{50} \approx 7.2 \times 10^{73}$ leaf nodes. But the number of inference steps is even bigger because not only every leaf node, but also every inner node of the tree corresponds to an inference step. Therefore we must add up the nodes over all levels and obtain the total number of nodes of the search tree

$$\sum_{d=0}^{50} 30^d = \frac{1-30^{51}}{1-30} = 7.4 \times 10^{73},$$

which does not change the node count by much. Evidently, nearly all of the nodes of this search tree are on the last level. As we will see, this is generally the case. But now back to the search tree with the 7.4×10^{73} nodes. Assume we had 10,000 computers which can each perform a billion inferences per second, and that we could

W. Ertel, *Introduction to Artificial Intelligence*,
Undergraduate Topics in Computer Science,
DOI 10.1007/978-0-85729-299-5_6, © Springer-Verlag London Limited 2011

image not available

image
not
available

Definition 6.1 A search problem is defined by the following values

State: Description of the state of the world in which the search agent finds itself.

Starting state: The initial state in which the search agent is started.

Goal state: If the agent reaches a goal state, then it terminates and outputs a solution (if desired).

Actions: All of the agents allowed actions.

Solution: The path in the search tree from the starting state to the goal state.

Cost function: Assigns a cost value to every action. Necessary for finding a cost-optimal solution.

State space: Set of all states.

Applied to the 8-puzzle, we get

State: 3×3 matrix S with the values 1, 2, 3, 4, 5, 6, 7, 8 (once each) and one empty square.

Starting state: An arbitrary state.

Goal state: An arbitrary state, e.g. the state given to the right in Fig. 6.2 on page 86.

Actions: Movement of the empty square S_{ij} to the left (if $j \neq 1$), right (if $j \neq 3$), up (if $i \neq 1$), down (if $i \neq 3$).

Cost function: The constant function 1, since all actions have equal cost.

State space: The state space is degenerate in domains that are mutually unreachable (Exercise 6.4 on page 110). Thus there are unsolvable 8-puzzle problems.

For analysis of the search algorithms, the following terms are needed:

Definition 6.2

- The number of successor states of a state s is called the *branching factor* $b(s)$, or b if the branching factor is constant.
- The *effective branching factor* of a tree of depth d with n total nodes is defined as the branching factor that a tree with constant branching factor, equal depth, and equal n would have (see Exercise 6.3 on page 110).
- A search algorithm is called *complete* if it finds a solution for every solvable problem. If a complete search algorithm terminates without finding a solution, then the problem is unsolvable.

For a given depth d and node count n, the effective branching factor can be calculated by solving the equation

$$n = \frac{b^{d+1} - 1}{b - 1} \tag{6.1}$$

image
not
available

```
DEPTH-FIRST-SEARCH(Node, Goal)

If GoalReached(Node, Goal) Return("Solution found")
NewNodes = Successors(Node)
While NewNodes ≠ ∅
    Result = DEPTH-FIRST-SEARCH(First(NewNodes), Goal)
    If Result = "Solution found" Return("Solution found")
    NewNodes = Rest(NewNodes)
Return("No solution")
```

Fig. 6.8 The algorithm for depth-first search. The function "First" returns the first element of a list, and "Rest" the rest of the list

list of nodes is always expanded, and the new nodes sorted in. Thus we find the optimal solution. The memory problem is not yet solved, however. A solution for this problem is provided by depth-first search.

6.2.2 Depth-First Search

In depth-first search only a few nodes are stored in memory at one time. After the expansion of a node only its successors are saved, and the first successor node is immediately expanded. Thus the search quickly becomes very deep. Only when a node has no successors and the search fails at that depth is the next open node expanded via *backtracking* to the last branch, and so on. We can best perceive this in the elegant recursive algorithm in Fig. 6.8 and in the search tree in Fig. 6.9 on page 92.

Analysis Depth-first search requires much less memory than breadth-first search because at most b nodes are saved at each depth. Thus we need $b \cdot d$ memory cells.

However, depth-first search is not complete for infinitely deep trees because depth-first search runs into an infinite loop when there is no solution in the far left branch. Therefore the question of finding the optimal solution is obsolete. Because of the infinite loop, no bound on the computation time can be given. In the case of a finitely deep search tree with depth d, a total of about b^d nodes are generated. Thus the computation time grows, just as in breadth-first search, exponentially with depth.

We can make the search tree finite by setting a depth limit. Now if no solution is found in the pruned search tree, there can nonetheless be solutions outside the limit. Thus the search becomes incomplete. There are obvious ideas, however, for getting the search to completeness.

image
not
available

```
HEURISTICSEARCH(Start, Goal)
NodeList = [Start]
While True
    If NodeList = ∅ Return("No solution")
    Node = First(NodeList)
    NodeList = Rest(NodeList)
    If GoalReached(Node, Goal) Return("Solution found", Node)
    NodeList = SortIn(Successors(Node),NodeList)
```

Fig. 6.12 The algorithm for heuristic search

trouble. However, we do not carry out this kind of analysis because there is a very high probability that our heuristic selection will succeed and will quickly get us to our goal of eating tasty strawberries.

Heuristic decisions are closely linked with the need to make *real-time decisions with limited resources*. In practice a good solution found quickly is preferred over a solution that is optimal, but very expensive to derive.

A heuristic evaluation function $f(s)$ for states is used to mathematically model a heuristic. The goal is to find, with little effort, a solution to the stated search problem with minimal total cost. Please note that there is a subtle difference between the effort to find a solution and the total cost of this solution. For example it may take Google Maps half a second's worth of effort to find a route from the City Hall in San Francisco to Tuolumne Meadows in Yosemite National Park, but the ride from San Francisco to Tuolumne Meadows by car may take four hours and some money for gasoline etc. (total cost).

Next we will modify the breadth-first search algorithm by adding the evaluation function to it. The currently open nodes are no longer expanded left to right by row, but rather according to their heuristic rating. From the set of open nodes, the node with the minimal rating is always expanded first. This is achieved by immediately evaluating nodes as they are expanded and sorting them into the list of open nodes. The list may then contain nodes from different depths in the tree.

Because heuristic evaluation of states is very important for the search, we will differentiate from now on between states and their associated nodes. The node contains the state and further information relevant to the search, such as its depth in the search tree and the heuristic rating of the state. As a result, the function "Successors", which generates the successors (children) of a node, must also immediately calculate for these successor nodes their heuristic ratings as a component of each node. We define the general search algorithm HEURISTICSEARCH in Fig. 6.12.

The node list is initialized with the starting nodes. Then, in the loop, the first node from the list is removed and tested for whether it is a solution node. If not, it will be expanded with the function "Successors" and its successors added to the list

image
not
available

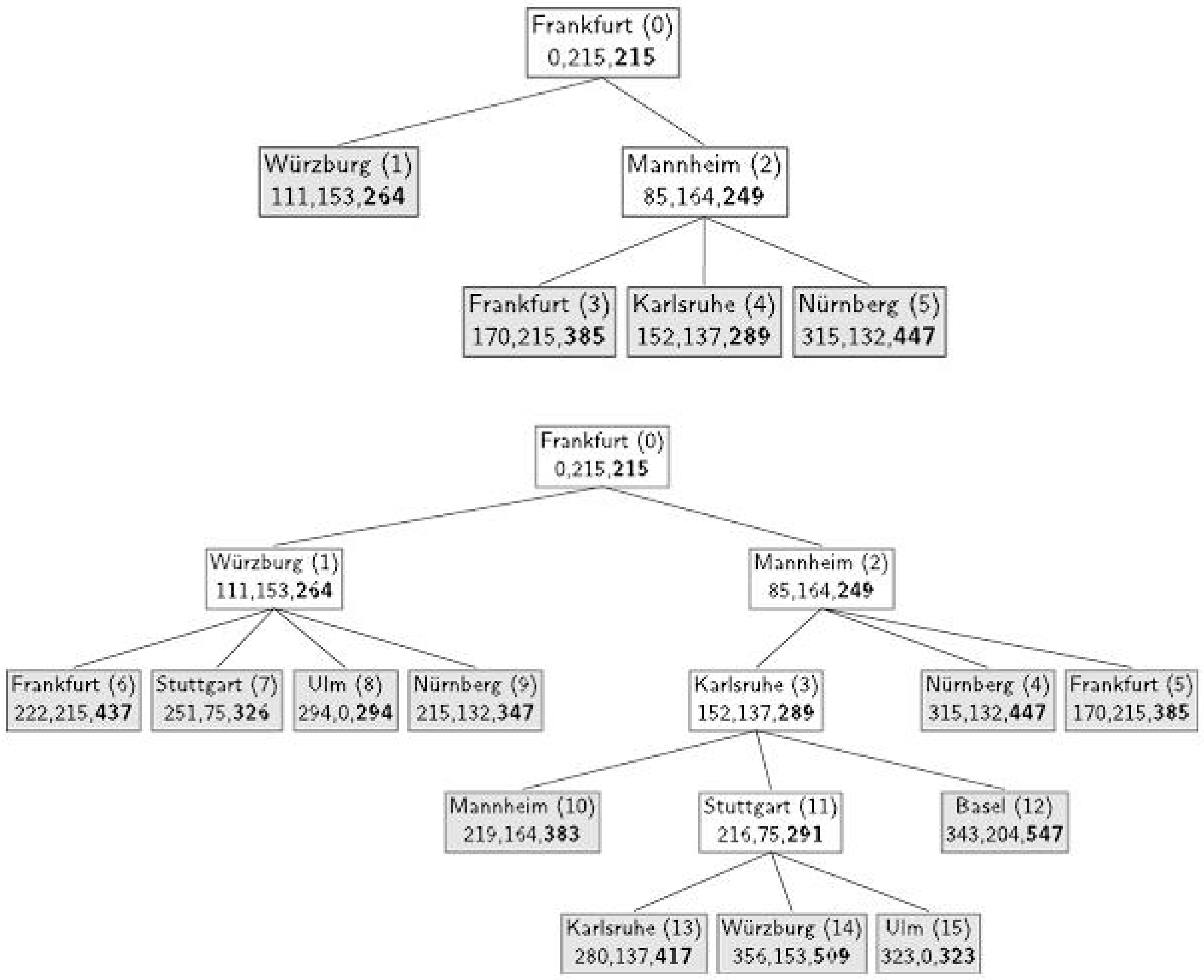

Frankfurt (0) 0,215,**215**
Würzburg (1) 111,153,**264**
Mannheim (2) 85,164,**249**
Frankfurt (3) 170,215,**385**
Karlsruhe (4) 152,137,**289**
Nürnberg (5) 315,132,**447**
Frankfurt (0) 0,215,**215**
Würzburg (1) 111,153,**264**
Mannheim (2) 85,164,**249**
Frankfurt (6) 222,215,**437**
Stuttgart (7) 251,75,**326**
Ulm (8) 294,0,**294**
Nürnberg (9) 215,132,**347**
Karlsruhe (3) 152,137,**289**
Nürnberg (4) 315,132,**447**
Frankfurt (5) 170,215,**385**
Mannheim (10) 219,164,**383**
Stuttgart (11) 216,75,**291**
Basel (12) 343,204,**547**
Karlsruhe (13) 280,137,**417**
Würzburg (14) 356,153,**509**
Ulm (15) 323,0,**323**

Fig. 6.16 Two snapshots of the A* search tree for the optimal route from Frankfurt to Ulm. In the boxes below the name of the city s we show $g(s)$, $h(s)$, $f(s)$. Numbers in parentheses after the city names show the order in which the nodes have been generated by the "Successor" function

In the top part of Fig. 6.16 we see that the successors of Mannheim are generated before the successors of Würzburg. The optimal solution Frankfurt–Würzburg–Ulm is generated shortly thereafter in the eighth step, but it is not yet recognized as such. Thus the algorithm does not terminate yet because the node Karlsruhe (3) has a better (lower) f value and thus is ahead of the node Ulm (8) in line. Only when all f values are greater than or equal to that of the solution node Ulm (8) have we ensured that we have an optimal solution. Otherwise there could potentially be another solution with lower costs. We will now show that this is true generally.

Theorem 6.2 *The A* algorithm is optimal. That is, it always finds the solution with the lowest total cost if the heuristic h is admissible.*

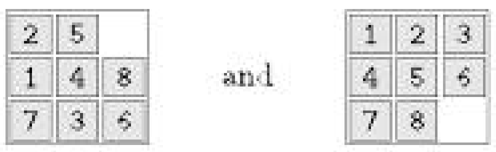
Proof In the HEURISTICSEARCH algorithm, every newly generated node s is sorted in by the function "SortIn" according to its heuristic rating $f(s)$. The node with the smallest rating value thus is at the beginning of the list. If the node l at the

image
not
available

Table 6.2 Comparison of the computation cost of uninformed search and heuristic search for solvable 8-puzzle problems with various depths. Measurements are in steps and seconds. All values are averages over multiple runs (see last column)

Depth	Iterative deepening		A* algorithm				Num. runs
			Heuristic h_1		Heuristic h_2		
	Steps	Time [sec]	Steps	Time [sec]	Steps	Time [sec]	
2	20	0.003	3.0	0.0010	3.0	0.0010	10
4	81	0.013	5.2	0.0015	5.0	0.0022	24
6	806	0.13	10.2	0.0034	8.3	0.0039	19
8	6455	1.0	17.3	0.0060	12.2	0.0063	14
10	50512	7.9	48.1	0.018	22.1	0.011	15
12	486751	75.7	162.2	0.074	56.0	0.031	12
			IDA*				
14	–	–	10079.2	2.6	855.6	0.25	16
16	–	–	69386.6	19.0	3806.5	1.3	13
18	–	–	708780.0	161.6	53941.5	14.1	4

square the horizontal and vertical distances to that square's location in the goal state are added together. This value is then summed over all squares. For example, the Manhattan distance of the two states



is calculated as

$$h_2(s) = 1 + 1 + 1 + 1 + 2 + 0 + 3 + 1 = 10.$$

The admissibility of the Manhattan distance is also obvious (see Exercise 6.13 on page 111).

The described algorithms were implemented in Mathematica. For a comparison with uninformed search, the A* algorithm with the two heuristics h_1 and h_2 and iterative deepening was applied to 132 randomly generated 8-puzzle problems. The average values for the number of steps and computation time are given in Table 6.2. We see that the heuristics significantly reduce the search cost compared to uninformed search.

If we compare iterative deepening to A* with h_1 at depth 12, for example, it becomes evident that h_1 reduces the number of steps by a factor of about 3,000, but the computation time by only a factor of 1,023. This is due to the higher cost per step for the computation of the heuristic.

Closer examination reveals a jump in the number of steps between depth 12 and depth 14 in the column for h_1. This jump cannot be explained solely by the repeated work done by IDA*. It comes about because the implementation of the A* algorithm

image
not
available

leaf nodes would be created. This means that the depth limit and thus also the search horizon are doubled with alpha-beta pruning. However, this is only true in the case of optimally sorted successors because the child nodes' ratings are unknown at the time when they are created. If the child nodes are randomly sorted, then the branching factor is reduced to $b^{3/4}$ and the number of leaf nodes to

$$n_d = b^{\frac{3}{4}d}.$$

With the same computing power a chess computer using alpha-beta pruning can, for example, compute eight half-moves ahead instead of six, with an effective branching factor of about 14. A thorough analysis with a derivation of these parameters can be found in [Pea84].

To double the search depth as mentioned above, we would need the child nodes to be optimally ordered, which is not the case in practice. Otherwise the search would be unnecessary. With a simple trick we can get a relatively good node ordering. We connect alpha-beta pruning with iterative deepening over the depth limit. Thus at every new depth limit we can access the ratings of all nodes of previous levels and order the successors at every branch. Thereby we reach an effective branching factor of roughly 7 to 8, which is not far from the theoretical optimum of $\sqrt{35}$ [Nil98].

6.4.3 Non-deterministic Games

Minimax search can be generalized to all games with non-deterministic actions, such as backgammon. Each player rolls before his move, which is influenced by the result of the dice roll. In the game tree there are now therefore three types of levels in the sequence

Max, dice, Min, dice, ...,

where each dice roll node branches six ways. Because we cannot predict the value of the die, we average the values of all rolls and conduct the search as described with the average values from [RN10].

6.5 Heuristic Evaluation Functions

How do we find a good heuristic evaluation function for the task of searching? Here there are fundamentally two approaches. The classical way uses the knowledge of human experts. The knowledge engineer is given the usually difficult task of formalizing the expert's implicit knowledge in the form of a computer program. We now want to show how this process can be simplified in the chess program example.

In the first step, experts are questioned about the most important factors in the selection of a move. Then it is attempted to quantify these factors. We obtain a list of relevant features or attributes. These are then (in the simplest case) combined into

image
not
available

image
not
available

Index