Chapter 1

Introduction

Artificial Intelligence (AI) is an exciting field somewhere on the edge between computing science, mathematics, cognitive science, and philosophy, however with a strong basis in computing and mathematics. As computing science, it is also a relatively new area: the first steps towards the creation of systems that behave intelligently using computers were taken in the mid 1950s. However, the idea of mimicking human behaviour by devices is much older, and was, for example, already popular in the 18th century, the age of the Enlightenment, when the art of constructing mechanical toys, calculating engines and clocks was at its hight. Blaise Pascal (1623–1662) created a preliminary mechanical calculator in 1642, and Gottfried Wilhelm Von Leibniz (1646–1716) built a machine, the Stepped Reckoner, that was able to perform the four basic arithmetical operations of addition, subtraction, multiplication and division (See Figure 1).

Although begone in the 1950s as a research area without any obvious practical applications, AI has now become main stream, with applications in almost any area of information technology. As the research area of AI has always been looking for ways to explore new ideas, research has moved on steadily since the early days of AI. Some of the results of early AI research are therefore no longer recognized as belonging to AI, although they once were. Examples include functional and object-oriented programming, theorem proving and game playing.

During the last 20 years, the area of *knowledge-based systems*, one of the first areas of artificial intelligence to be commercially fruitful, has received a lot of attention. The phrase knowledge-based system is generally employed to indicate information systems in which some symbolic representation of human knowledge is applied, usually in a way resembling human reasoning. Knowledge-based systems are also called *knowledge systems* or *expert systems*. Building knowledge-based systems for specific application domains has even become a separate subject known as *knowledge engineering*.

Knowledge-based systems are not the only fruit of AI research, as the area of intelligent systems is much broader than systems that include representations of human knowledge. Other areas of AI research, some of them closely linked to knowledge-based systems, are: machine learning, qualitative and model-based reasoning, decision-making, planning, and intelligent agents. In this book, we will focus on intelligent systems in which the representation of knowledge and reasoning with knowledge is central, i.e. topics such as machine learning and model-based reasoning are approached from a *knowledge-based point of view*.

In the present chapter, we review the historical roots of AI and intelligent systems, and

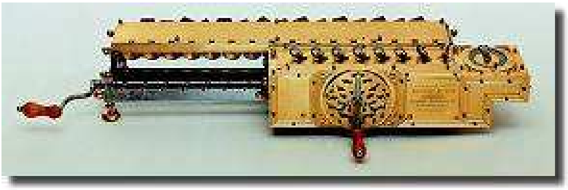

Figure 1.1: The Stepped Reckoner used a special type of gear named stepped drum which was a cylinder with nine bar-shaped teeth of incrementing length parallel to the cylinder's axis. When the drum is rotated by using a crank, a regular ten-tooth wheel, fixed over a sliding axis, is rotated zero to nine positions depending on its relative position to the drum. There is one set of wheels for each digit. This allows the user to slide the mobile axis so that when the drum is rotated it generates in the regular wheels a movement proportional to their relative position. This movement is then translated by the device into multiplication or division depending on which direction the stepped drum is rotated.

briefly discuss several classical examples of knowledge-based systems. Furthermore, the basic principles of intelligent systems are introduced and related to the subsequent chapters of the book, where these principles are treated in significant depth. The chapter concludes with a description of a problem domain.

1.1 Intelligent systems and AI

Although the digital computer was originally designed to be a number processor, even in the early days of its creation there was a small core of researchers engaged in non-numerical applications. The efforts of these researchers eventually led to what is known since the Dartmouth Summer Seminar in 1956, where now famous people such as John McCarthy, Alan Newell, Marvin Minsky, and Hebert Simon met to establish the field, as artificial intelligence (AI), the area of computer science concerned with systems producing results for which human behaviour would seem necessary.

The early areas of attention in the 1950s were *theorem proving* and *problem solving*. In both fields, the developed computer programs are characterized by being based on complex algorithms which have a general solving capability, independent of a specific problem domain and which furthermore operate on problems posed in rather simple primitives.

Theorem proving is the field concerned with proving theorems automatically from a given set of axioms by a computer. The theorems and axioms are expressed in logic, and logical inference rules are applied to the given set of axioms in order to prove the theorems. The first program that actually constructed a mathematical proof of a theorem in number theory was developed by Martin Davis as early as 1954. Nevertheless, the major breakthrough in theorem proving did not come until halfway the sixties. Only after the introduction of an inference rule called resolution, did theorem proving become interesting from a practical point of view. Further progress in the field during the seventies came from the development of several refinements of the original resolution principle.

Researchers in the field of problem solving focused on the development of computer systems with a general capability for solving different types of problems. The best known system is *GPS* (General Problem Solver), developed by Alan Newell, Herbert Simon and Jack Shaw.

A given problem is represented in terms of an initial state, a wished for final state and a set of transitions to transform states into new states. Given such a representation by means of states and operations, GPS generates a sequence of transitions that transform the initial state into the given final state when applied in order. GPS has not been very successful. First, representing a non-trivial problem in terms which could be processed by GPS proved to be no easy task. Secondly, GPS turned out to be rather inefficient. Since GPS was a general problem solver, specific knowledge of the problem at hand could not be exploited in choosing a transition on a given state, not even if such knowledge indicated that a specific transition would lead to the solution of the problem more efficiently. In each step GPS examined all possible transitions, thus yielding an exponential time complexity. Although the success of GPS as a problem solver has been rather limited, GPS initiated a significant shift of attention in artificial intelligence research towards more specialized systems. This shift in attention from general problem solvers to specialized systems in which the reasoning process could be monitored using knowledge of the given problem, is generally viewed as a breakthrough in artificial intelligence.

With the significant increase in computational power in modern computers, it is now less obvious than a decade ago that constructing general problem solvers is infeasible, and to some extent recent research has seen a return to general problem solvers. A typical example of success of modern general problem solvers is exemplified the research done by IBM's Thomas J. Watson Research Center, which finally led to the defeat of chess grand master Garry Kasparov by a programme, Deep Blue, on an RS/6000 SP parallel supercomputer in May 1997. However, it is now generally recognized that domain knowledge often plays a crucial role in problem solving, and therefore should be taken into account if possible.

For problems arising in practice in many domains, there are no well-defined solutions which can be found in the literature. The knowledge an expert in the field has, is generally not laid down in clear definitions or unambiguous algorithms, but merely exists in rules of thumb and facts learnt by experience, called *heuristics*. So, the knowledge incorporated in an knowledge-based system is highly domain dependent. The success of knowledge-based systems is mainly due to their capability for representing heuristic knowledge and techniques, and for making these applicable for computers. Generally, knowledge-based systems are able to comment upon the solutions and advice they have given, based on the knowledge present in the system. Moreover, knowledge-based systems offer the possibility for integrating new knowledge with the knowledge that is already present, in a flexible manner.

1.2 Some early examples

The first knowledge-based systems were developed as early as the late 1960s. However, it took until the 1970s before the research actually started on a large scale. At the time knowledge-based systems were often called *expert systems*. Currently, the term *expert systems* is still used as a synonym for knowledge-based systems, in particular if it contains a lot of domain knowledge, but some people prefer to use the term 'knowledge-based system' or 'knowledge systems'.

The early knowledge-based systems mostly concerned the field of medical diagnosis. The best-known knowledge-based system in medicine, developed in the seventies, is *MYCIN*. The development of this knowledge-based system took place at Stanford University; especially Edward (Ted) Shortliffe played an important role in its development. The MYCIN system is

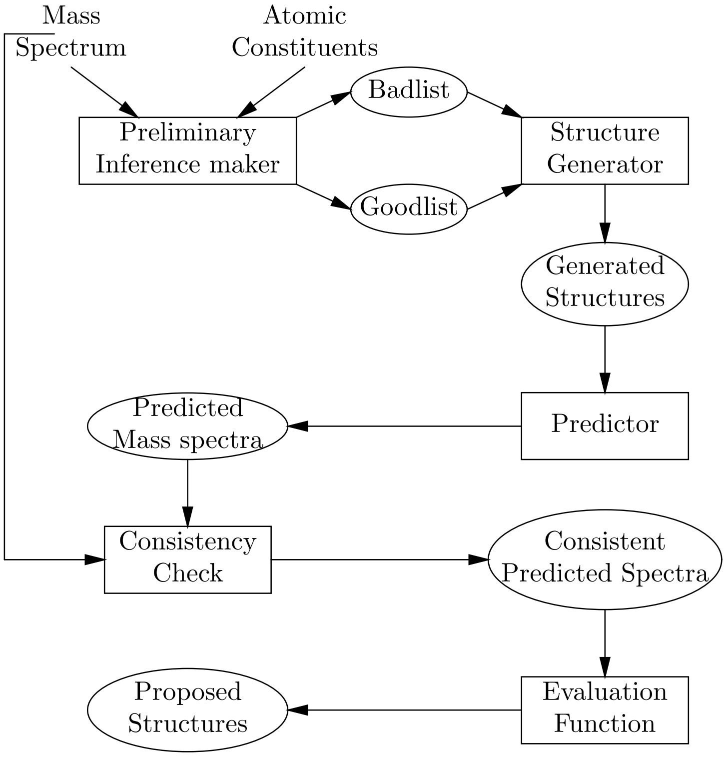

Figure 1.2: Structure of Heuristic DENDRAL.

able to assist internists in the diagnosis and the treatment of a number of infectious diseases, in particular meningitis and bacterial septicaemia. When a patient shows the signs of such an infectious disease, usually a culture of blood and urine is made in order to determine the bacterium species that causes the infection. Generally, it takes 24 to 48 hours before the laboratory results become known. In case of the above mentioned infectious diseases however, the physician will have to start treatment before these results are available, since otherwise the disease may progress and actually cause death of the patient. Given the patient data that are available to the system but which are apt to be incomplete and inexact, MYCIN gives an interim indication of the organisms that are most likely to be the cause of the infection. Given this indication, MYCIN advises the administration of a number of drugs that should control the disease by suppressing the indicated organisms. The interaction of the prescribed drugs among themselves and with the drugs the patient already takes, possible toxic drug reactions, etc. are also taken into account. Moreover, MYCIN is able to comment on the diagnosis it has arrived at, and the prescription of the drugs. The MYCIN system clearly left its mark on the knowledge-based systems that have been developed since. Even at present, this knowledge-based system and its derivatives are sources of ideas concerning the representation and manipulation of medical knowledge. The MYCIN system also has given an important impulse to the development of similar knowledge-based systems in fields other than medicine.

To a growing extent, knowledge-based systems are also being developed in technical fields. One of the first systems with which the phrase knowledge-based system has been associated, is *Heuristic DENDRAL*. The DENDRAL project commenced in 1965 at Stanford University. The system was developed by Joshua Lederberg, an organic chemist (and Nobel prize winner in chemistry), in conjunction with Edward Feigenbaum and Bruce Buchanan both well-known research scientists in artificial intelligence. The Heuristic DENDRAL system offers assistance in the field of organic chemistry in determining the structural formula of a chemical compound that has been isolated from a given sample. In determining a structural formula, information concerning the chemical formula, such as C_4H_9OH for butanol, and the source the compound has been taken from, is used as well as information that has been obtained by subjecting the compound to physical, chemical and spectrometric tests. The method employed is called *generate-and-test*, since the system first generates all plausible molecular structures as hypotheses, which subsequently are tested against the observed data. The original DENDRAL algorithm was developed by Joshua Lederberg for generating all possible isomers of a chemical compound. Heuristic DENDRAL contains a subsystem, the so-called Structure Generator, which implements the DENDRAL algorithm, but in addition incorporates various heuristic constraints on possible structures, thus reducing the number of alternatives to be considered by the remainder of the system. In particular, mass spectrometry is useful for finding the right structural formula. The structure of the system is shown in Figure 1.2. In a mass spectrometer, the compound is bombarded with a beam of electrons in vacuum, causing the molecule to break up into several smaller charged fragments. These fragments are accelerated within an electrical field, and are bent off in proportion to their mass-charge ratio, using a magnetic field. The fragments that are separated this way cause a pattern called a spectrogram, which is recorded by means of a writing device. Such a spectrogram shows a number of peaks corresponding to the respective mass-charge ratios of the separated fragments. A spectrogram provides significant information about the structure of the original chemical compound. Heuristic DENDRAL helps in interpreting the patterns in a spectrogram. To this end, another subsystem of Heuristic DENDRAL, called the Predictor, suggests expected mass spectrograms for each molecular structure generated by the Structure Generator. Each expected mass spectrogram is then tested against the mass spectrogram observed using some measure of similarity for comparison; this has been implemented in the last part of the system, the Evaluation Function. Usually, more than one molecular structure matches the pattern found in the spectrogram. Therefore, the system usually produces more than one answer, ordered by the amount of evidence favouring them.

XCON, previously called *R1*, was a knowledge-based system able to configure VAX, PDP11, and microVAX computer systems from Digital Equipment Corporation (DEC). DEC offered the customer a wide choice in components when purchasing computer equipment, so that each client can be provided with a custom-made system. Given the customer's order a configuration is made, possibly showing that a specific component has to be replaced by another equivalent component, or that a certain component has to be added in order to arrive at a fully operational system. The problem is not so much that the information is incomplete or inexact but merely that the information is subject to rapid change. Moreover, configuring a computer system requires considerable skill and effort. In the late seventies, DEC in conjunction with John McDermott from Carnegie-Mellon University commenced the development of XCON. Since 1981, XCON is fully operational. At present, XCON is supplemented with *XSEL*, a system that assists DEC agents in drawing up orders.

The knowledge-based systems mentioned above are classics. Inspired by their success, many more knowledge-based systems have been constructed since the end of the seventies. The systems have also led to the construction of various direct derivatives. For example, Heuristic DENDRAL has been elaborated further by incorporating improved subsystems for generating and testing plausible molecular structures. Moreover, a system capable of learning heuristics from example has been developed, called METADENDRAL, to ease the transfer of

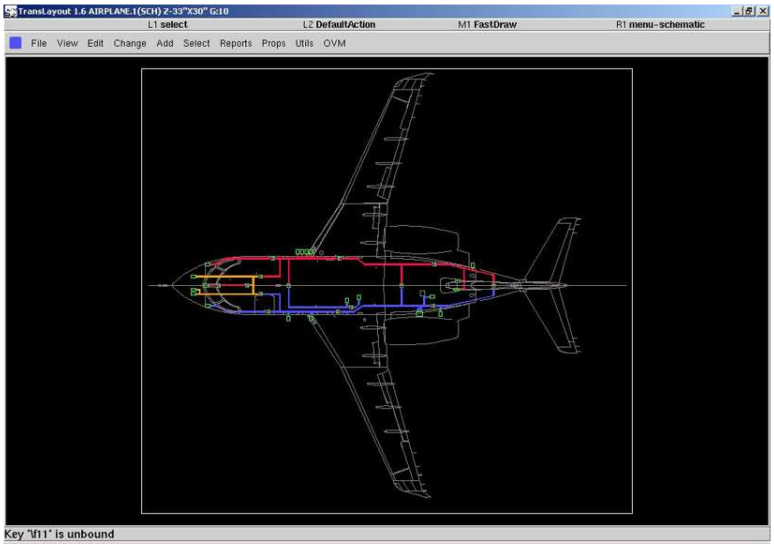

Figure 1.3: Model of the wiring of an aeroplane.

domain knowledge for use in Heuristic DENDRAL.

1.3 Some recent examples

Intelligent systems are increasingly used in systems that are too complex to be handled without computer-based assistance. Examples include cars, washing machines, aeroplanes, cell-phones, computer operating systems. In some of these systems, for example components of cars as built in the automotive industry, there is a need for explicit models of the system, such that the model of the system can be used to aid in activities such as design, fault finding and trouble shooting, and so on. *Model-based diagnosis*, i.e., the isolation of faults in a system using explicit models of the system and actually observed behaviour, is a process that has been investigated extensively in AI; results of this research have been incorporated into actual products.

For example, TransDesign is a complete electrical distribution design environment. It has been developed in cooperation with automotive engineers, and covers the design of an entire electrical system, from concept to manufacturing support. It enables manufacturers to design electrical harness systems and packaging details concurrently, using a mechanical design environment. One of the nicest features of a model-based systems is that it allows giving users feedback about a problem-solving process in terms of the actual model. An example is shown in Figure 1.3.

Microsoft Research has also invested a great deal of money in AI research, in particular focused on system that assist users in working with Microsoft software products, such as Microsoft Office. In the Lumiere project, the formalism of Bayesian networks, which allows representing and reasoning with uncertain knowledge, was investigated as the prime basis for office assistance software. The results of this AI research is now part of Microsoft Office, and similar systems have been subsequently incorporated into Microsoft Windows.

In other areas where reasoning with uncertain knowledge is important, such as medicine, Bayesian networks are also gaining ground. Many researchers are currently working on sys-

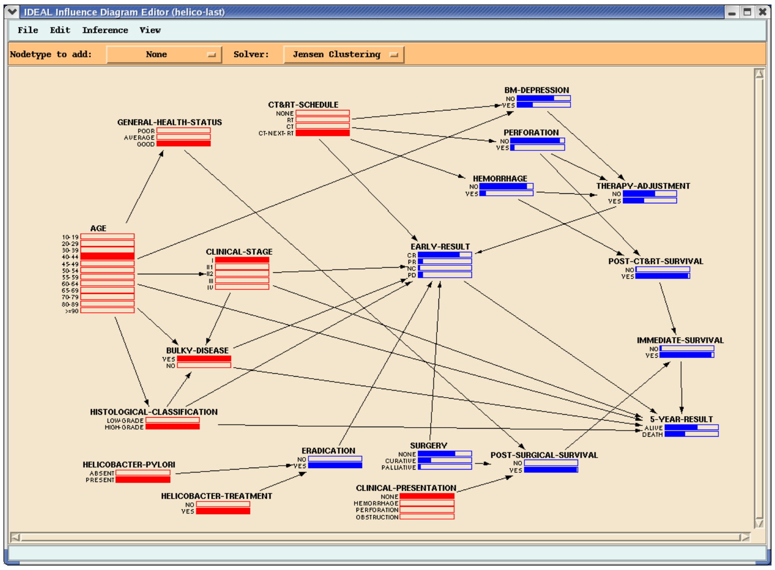

Figure 1.4: Bayesian network of gastric non-Hodgkin lymphoma. The network contains boxes with bar-graphs displaying probability distributions. At the left side of the figure are the patient findings; here the probability of the state that has been filled in for a patient is equal to 1. At the right side of the diagram are predictions based on these findings and the probabilistic information encoded in the Bayesian network. The variable '5-year-result' represents the result, death or survival, after 5 years. For the chosen combination of treatment: antibiotics, no surgery, and chemotherapy followed by radiotherapy, the probability of being alive after 5 years is about 70%.

tem that are able to help physicians with diagnosing disease or with selecting appropriate treatment. An example of such work is a Bayesian network model that has been designed with the help of cancer specialists from the Netherlands Cancer Institute, and that is able to assist in the selection of treatment (surgery, chemotherapy, radiotherapy or a combination of the three), based on patient findings, for a rare type of cancer of the stomach, called gastric non-Hodgkin lymphoma (See Figure 1.4 for details).

1.4 Separating knowledge and inference

In the early years, intelligent systems were usually written in a high-level programming language. LISP, in particular, was frequently chosen for the implementation language. LISP is still a popular language for the development of intelligent systems, although it has obtained significant competition from Java and C++, in particular for the development of intelligent systems that involve some numerical computation. For the development of knowledge-based systems the situation is more clearcut: when using a high-level programming language as a knowledge-based system building tool one has to pay a disproportionate amount of attention to the implementation aspects of the system which have nothing to do with the field to be modelled. Moreover, the expert knowledge of the field and the algorithms for applying this knowledge automatically, will be highly interwoven and not easily set apart. This led to systems that once constructed, were practically not adaptable to changing views on the field of concern. Expert knowledge however has a dynamic nature: knowledge and experience are continuously subject to changes. Awareness of these properties has led to the view that the explicit separation of the algorithms for applying the highly-specialized knowledge from the knowledge itself is highly desirable if not mandatory for developing knowledge-based systems. This fundamental insight for the development of present-day knowledge-based systems is formulated in the following equation, sometimes called the paradigm of knowledge-based system design:

knowledge-based system = *knowledge* + *inference*

Consequently, a knowledge-based system typically comprises the following two essential components:

- A *knowledge base* capturing the domain-specific knowledge, and
- An *inference engine* consisting of algorithms for manipulating the knowledge represented in the knowledge base.

Nowadays, a knowledge-based system is rarely written in a high-level programming language. It frequently is constructed in a special, restricted environment, called an *knowledge-based system shell* or *expert-system shell*. An early example of such an environment was *EMYCIN* (Essential MYCIN), a system that originated from MYCIN by stripping it of its knowledge concerning infectious disease. Several more general tools for building knowledge-based systems, more like special-purpose programming languages, have also become available, where again such a separation between knowledge and inference is enforced. These systems will be called *knowledge-based system builder tools* and are also known as *expert-system builder tools*.

The domain-specific knowledge is laid down in the knowledge base using a special *knowledge-representation formalism*. In a knowledge-based system shell or a knowledge-based system builder tool, one or more knowledge-representation formalisms are predefined for encoding the domain knowledge. Furthermore, a corresponding inference engine is present that is capable of manipulating the knowledge represented in such a formalism. In developing an actual knowledge-based system only the domain-specific knowledge has to be provided and expressed in the knowledge-representation formalism. Several advantages arise from the fact that a knowledge base can be developed separately from the inference engine, for instance, a knowledge base can be developed and refined stepwise, and errors and inadequacies can easily be remedied without making major changes in the program text necessary. Explicit separation of knowledge and inference has the further advantage that a given knowledge base can be substituted by a knowledge base on another subject thus rendering quite a different knowledge-based system. The typical structure of a knowledge-based system is shown in Figure 1.5.

Representing the knowledge that is to be used in the process of problem solving is a central topic of AI, i.e. the topic of *knowledge representation*. A suitable knowledge-representation formalism should:

- Have sufficient expressive power for encoding the particular domain knowledge;
- Posses a clean semantic basis, such that the meaning of the knowledge present in the knowledge base is easy to grasp, especially by the user;
- Permit efficient algorithmic interpretation;

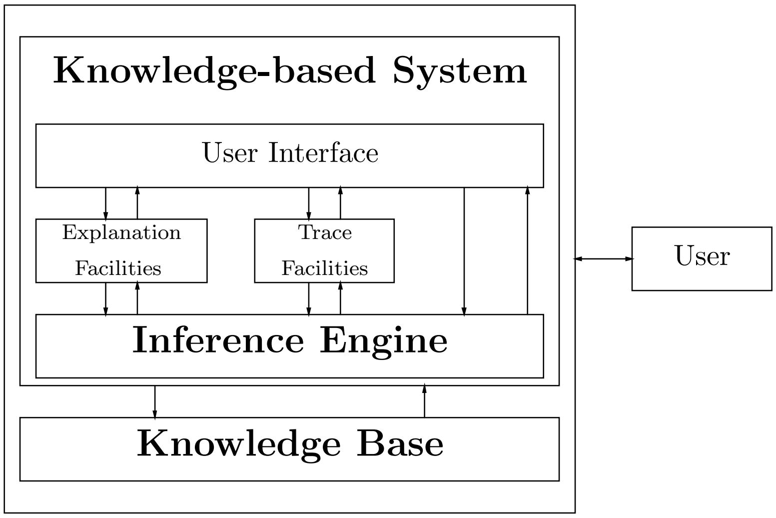

Figure 1.5: Global architecture of a knowledge-based system.

- Allow for explanation and justification of the solutions obtained by showing why certain questions were asked of the user, and how certain conclusions were drawn.

Part of these conditions concerns the form (*syntax*) of a knowledge-representation formalism; others concern its meaning (*semantics*). Unfortunately, it turns out that there is not a single knowledge-representation formalism which meets all of the requirements mentioned. In particular, the issues of expressive power of a formalism and its efficient interpretation seem to be conflicting. However, as we shall see in the following chapters, by restricting the expressive power of a formalism (in such a way that the domain knowledge can still be represented adequately), we often arrive at a formalism that does indeed permit efficient interpretation.

Developing a specific knowledge-based system is done by consulting various *knowledge sources*, such as human experts, text books, and databases. If data are being used, machine learning will be used to extract knowledge from data. However, building a knowledge-based system with the help of human experts is a task requiring high skills; the person performing this task is called the *knowledge engineer*. The process of collecting and structuring knowledge in a problem domain is called *knowledge acquisition*. If, more in particular, the knowledge is obtained by interviewing domain experts, one speaks of *knowledge elicitation*. Part of the work of a knowledge engineer concerns the selection of a suitable knowledge-representation formalism for presenting the domain knowledge to the computer in an encoded form.

1.5 Content of the book

We now present a short overview of the subjects which will be dealt with in this book.

From the proliferation of ideas that arose in AI, four types of knowledge-representation formalisms have emerged:

- *Logic*,
- *Production rules*,

- *Semantic nets* and *frames*,
- *Uncertainty formalisms.*

In the four subsequent chapters, we shall deal with the question how knowledge can be represented using these respective knowledge-representation formalisms. These formalisms can be seen as offering building blocks for the actual building intelligent systems. For an actual system, a formalism is often used in a particular fashion, which, in itself, is also part of the knowledge-representation enterprise. For example, a system that is used for the diagnosis of defects in a car is based on principles different from an intelligent systems that helps in route planning of a car, even if exactly the same formalism, for example production rules, is used for its implementation.

Associated with each of the knowledge-representation formalisms are specific methods for handling the represented knowledge. Inferring new information from the available knowledge is called *reasoning* or *inference*. With the availability of the first digital computers, automated reasoning in logic became one of the first subjects of research, yielding results which concerned proving theorems from mathematics. However, in this field, the immanent conflict between the expressiveness of the logic required for representing mathematical problems and its efficient interpretation was soon encountered. Sufficiently efficient algorithms were lacking for applying logic in a broader context. In 1965 though, Alan Robinson formulated a general inference rule, known as the *resolution principle*, which made automated theorem proving more feasible. In Chapter 2 we pay attention to the representation of knowledge in logic and automated reasoning with logical formulas; it will also be indicated how logic can be used for building a knowledge-based system.

Since the late sixties, considerable effort in artificial intelligence research has been spent on developing knowledge-representation formalisms other than logic, resulting in the before-mentioned production rules and frames. For each of these formalisms, special inference methods have been developed that on occasion closely resemble logical inference. Usually, two basic types of inference are discerned. The phrases *top-down inference* and *goal-directed inference* are used to denote the type of inference in which given some initial goal, subgoals are generated by employing the knowledge in the knowledge base until such subgoals can be reached using the available data. The second type of inference is called *bottom-up inference* or *data-driven inference*. When applying this type of inference, new information is derived from the available data and the knowledge in the knowledge base. This process is repeated until it is not possible any more to derive new information. The distinction between top-down inference and bottom-up inference is most explicitly made in reasoning with production rules, although the two types of reasoning are distinguished in the context of the other knowledge-representation formalisms as well. The production rule formalism and its associated reasoning methods are the topics of Chapter 3.

Chapter 4 is concerned with the third major approach in knowledge representation: semantic nets and frames. These knowledge representation schemes are characterized by a hierarchical structure for storing information. Since semantic nets and frames have several properties in common and the semantic net generally is viewed as the predecessor of the frame formalism, these formalisms are dealt with in one chapter. The method used for the manipulation of knowledge represented in semantic nets and frames is called *inheritance*.

As we have noted before, knowledge-based systems are used to solve real-life problems which do not have a predefined solution to be found in the relevant literature. Generally, the knowledge that is explicitly available on the subject is incomplete or uncertain. Nevertheless, a human expert often can arrive at a sound solution to the given problem using such deficient knowledge. Consequently, knowledge-based systems research aims at building systems capable of handling incomplete and uncertain information as well as human experts are. Several models for reasoning with uncertainty have been developed. Some of these will be discussed in Chapter 5.

In Chapter 6 we will focus on model-based approached to the development of intelligent systems, where in particular model-based diagnosis will be studied. In Chapter 7 the principles of machine learning will covered.

Finally, Chapter 8 deals with a number of well-known tools for the construction of intelligent systems. Discussed are OPS5, a special-purpose programming language designed for developing production systems, the closely related language and environment CLIPS, and the theorem-proving system Otter.

1.6 A problem domain

In each of the three subsequent chapters, a specific knowledge-representation formalism and its associated inference method will be treated. An example that will be used in some of the chapters concerns the representation and reasoning with a logical circuit. This is a simple example that is useful to illustrate a number of issues.

Consider Figure 1.6 that depicts a schematic representation of a full-adder. The full-adder is an example of a circuit that is of practical use in electronics; it is also simple enought to allow us to illustrate particular ideas. The full-adder consists of two exclusive OR gates (*ex1* and *ex2*), two AND gates (*ad1* and *ad2*) and one OR gate (*or1*). There are three inputs i_1, i_2 and i_3, and two outputs o_1 and o_2. Both inputs and outputs can be either 0 or 1. Output o_1 represents the sum of the inputs, whereas o_2 stands for the carry-out, i.e. a bit that represents overflow. For example, if $i_1 = 1, i_2 = 1, i_3 = 0$, then $o_1 = 0$ and $o_2 = 1$. This represents the binary number 10_2. The input i_3 is used to keep overflow information from previous computations; it is called the *carry-in*.

There are various AI approaches to problem solving that can be studied using this simple example. For example, one could design a representation and reasoning method to *simulate* the behaviour of the full-adder. This can be done by representing individual components—gates and wiring—of the full-adder, with associated behaviour, and combining these to obtain an overall behaviour. Note that although, for example, *ex1* and *ex2* are separate gates, their behaviour needs only be described once, i.e. it is possible to provide a *generic description* of the behaviour of these components.

The same description of the full-adder can also be used to diagnose defective components, i.e. it can be used for *diagnosis*. This can be done by assuming that particular components are faulty whereas others are behaving normally. These assumptions have an effect on the simulated behaviour, and by comparing the behaviour observed in reality with the predictions made by simulation, it is possible to establish which components are faulty. Ways to accomplish this are discussed in Chapters 2 and 6.

The kind of knowledge concerning the full-adder, an example of a logical circuit, as presented above, is called *deep knowledge*. Deep knowledge entails the detailed structure and function of some system in the domain of discourse. Clearly, deep knowledge may be valuable for *diagnostic reasoning*, that is, reasoning aimed at finding the cause of failure of some system. However, it is also possible to base a diagnosis on experience gathered in the course of

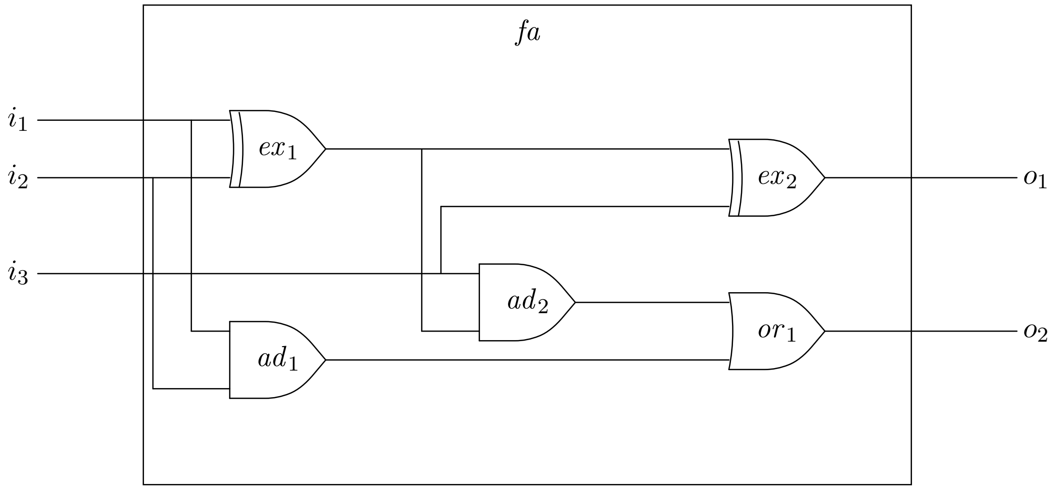

Figure 1.6: Full-adder.

time in dealing with defective systems. For example, based on experience it might be said that if the output o_1 is always equal to 0 irrespective of the input, component *ex1* is faulty. This kind of knowledge is often called *shallow knowledge* to distinguish it from deep knowledge. As one can see, no knowledge concerning the structure and function is used here; instead, the empirical association between observations "'o_1 is equal to 0 irrespective of the values of the inputs" is used as evidence for the diagnosis of a faulty component. Many knowledge-based systems only contain such shallow knowledge, since this is the kind of knowledge employed in daily practice by field experts for rapidly handling the problems they encounter. However, using deep knowledge frequently leads to a better justification of the solution proposed by the system. Other examples of the application of deep and shallow knowledge will be met with in the next chapters. It is not always easy to sharply distinguish between deep and shallow knowledge in a problem domain.

Exercises

1. One of the questions raised in the early days of artificial intelligence was: 'Can machines think?'. Nowadays, the question remains the subject of heated debates. This question was most lucidly formulated and treated by A. Turing in the paper 'Computing Machinery and Intelligence' which appeared in Mind, vol. 59, no. 236, 1950. Read the paper by A. Turing, and try to think what your answer would be when someone posed that question to you.

2. Read the description of GPS in section 1.1 of this chapter again. Give a specification of the process of shopping in terms of an initial state, final state, and transitions, as would be required by GPS.

3. An important component of the HEURISTIC DENDRAL system is the Structure Generator subsystem which generates plausible molecular structures. Develop a program in PROLOG or LISP that enumerates all possible structural formulas of a given alkane (that is, a compound having the chemical formula C_nH_{2n+2}) given the chemical formula as input for $n = 1, \ldots, 8$.

4. The areas of knowledge engineering and software engineering have much in common. However, there are also some evident distinctions. Which similarities and differences do you see between these fields?

5. Give some examples of deep and shallow knowledge from a problem domain you are familiar with.

6. Mention some problem areas in which knowledge-based systems can be of real help.