Principles of Intelligent Systems:

a Knowledge-based Approach

Peter Lucas and *Linda van der Gaag*

Preface

The present book is an introductory text book at the undergraduate level, covering the subject of intelligent systems for students of computing science and information science.

The central topics in this book are formalisms for the representation and manipulation of knowledge in the computer, in a few words: logic, production rules, semantic nets, frames, formalisms for inexact reasoning, and model-based reasoning. The choice for the formalisms discussed in the present book, has been motivated on the one hand by the requirement that at least the formalisms which nowadays are of fundamental importance to the area of intelligent systems must be covered, and on the other hand, that the formalisms which have been in use for considerable time and have laid the foundation of current research into more advanced methods should also be treated. We have in particular paid attention to those formalisms which have been shown to be of practical importance for building intelligent systems.

In addition to formalisms, attention is given to the modelling aspects of building intelligent systems, such as those being used in the construction of model-based systems.

Peter Lucas
21st September, 2005
Institute for Computing and Information Sciences
Radboud University Nijmegen, Nijmegen

Contents

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

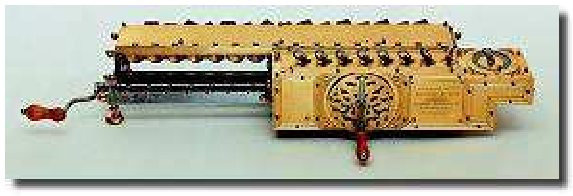


Figure 1.1: The Stepped Reckoner used a special type of gear named stepped drum which was a cylinder with nine bar-shaped teeth of incrementing length parallel to the cylinder's axis. When the drum is rotated by using a crank, a regular ten-tooth wheel, fixed over a sliding axis, is rotated zero to nine 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

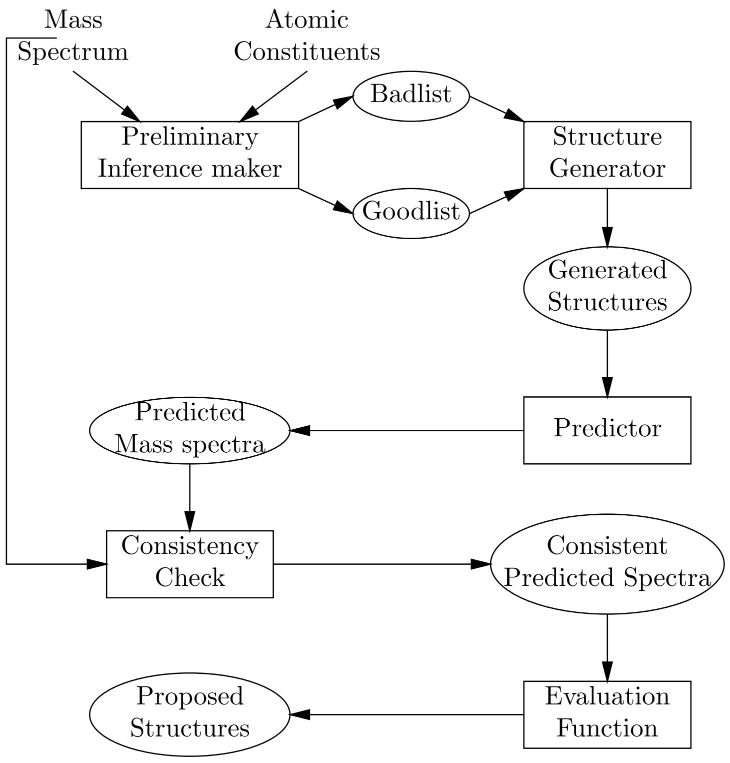


Figure 1.2: Structure of Heuristic DENDRAL.

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

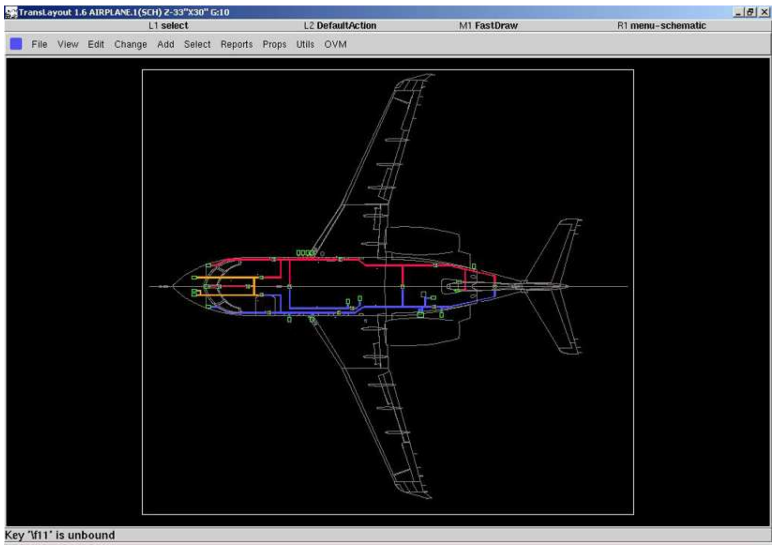

TransLayout 1.6 AIRPLANE.1(SCH) Z-33"X30" G:10
L1 select
L2 DefaultAction
M1 FastDraw
R1 menu-schematic
File View Edit Change Add Select Reports Props Utils OVM
Key '\f11' is unbound

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-

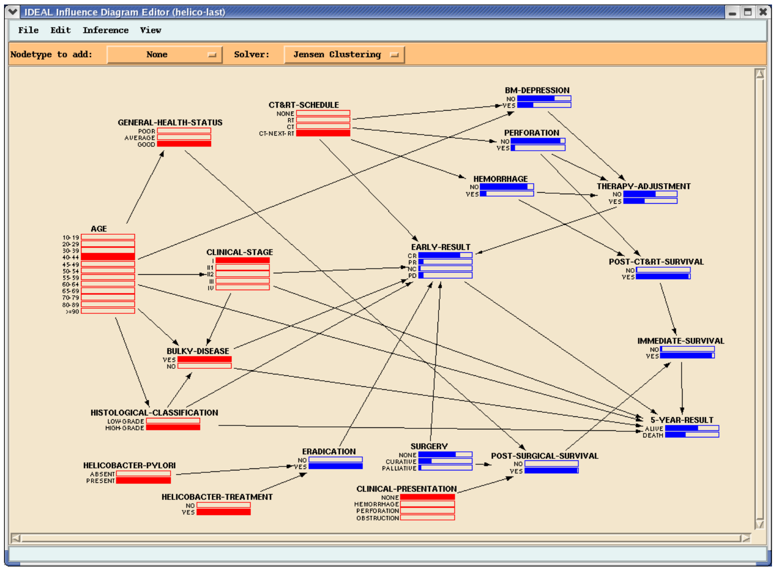

Nodetype to add: None
Solver: Jensen Clustering

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:

$$\textit{knowledge-based system} = \textit{knowledge} + \textit{inference}$$

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;

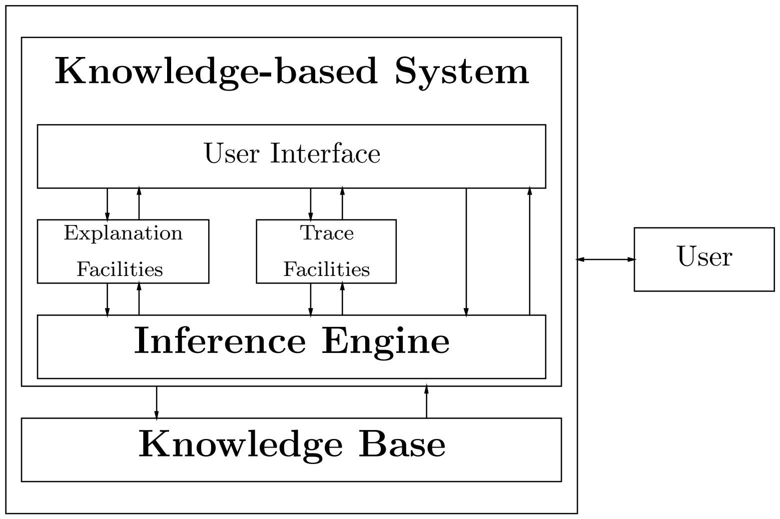


Figure 1.5: Global architecture of a knowledge-based system.

- 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

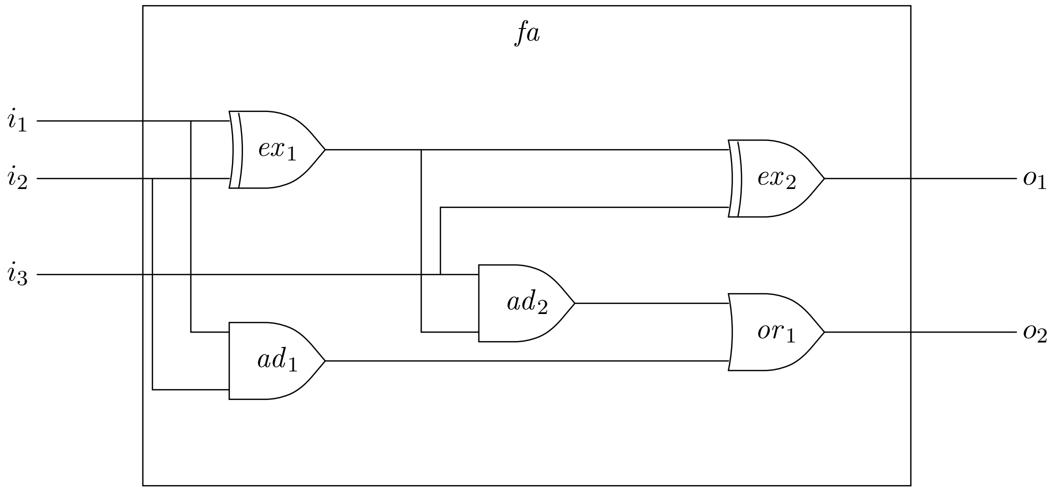


Figure 1.6: Full-adder.

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.

Chapter 2

Logic and Resolution

One of the earliest formalisms for the representation of knowledge is *logic*. The formalism is characterized by a well-defined syntax and semantics, and provides a number of *inference rules* to manipulate logical formulas on the basis of their form in order to derive new knowledge. Logic has a very long and rich tradition, going back to the ancient Greeks: its roots can be traced to Aristotle. However, it took until the present century before the mathematical foundations of modern logic were laid, amongst others by T. Skolem, J. Herbrand, K. Gödel, and G. Gentzen. The work of these great and influential mathematicians rendered logic firmly established before the area of computer science came into being.

Already from the early 1950s, as soon as the first digital computers became available, research was initiated on using logic for problem solving by means of the computer. This research was undertaken from different points of view. Several researchers were primarily interested in the mechanization of mathematical proofs: the efficient automated generation of such proofs was their main objective. One of them was M. Davis who, already in 1954, developed a computer program which was capable of proving several theorems from number theory. The greatest triumph of the program was its proof that the sum of two even numbers is even. Other researchers, however, were more interested in the study of human problem solving, more in particular in heuristics. For these researchers, mathematical reasoning served as a point of departure for the study of heuristics, and logic seemed to capture the essence of mathematics; they used logic merely as a convenient language for the formal representation of human reasoning. The classical example of this approach to the area of theorem proving is a program developed by A. Newell, J.C. Shaw and H.A. Simon in 1955, called the *Logic Theory Machine*. This program was capable of proving several theorems from the Principia Mathematica of A.N. Whitehead and B. Russell. As early as 1961, J. McCarthy, amongst others, pointed out that theorem proving could also be used for solving non-mathematical problems. This idea was elaborated by many authors. Well known is the early work on so-called *question-answering systems* by J.R. Slagle and the later work in this field by C.C. Green and B. Raphael.

After some initial success, it soon became apparent that the inference rules known at that time were not as suitable for application in digital computers as hoped for. Many AI researchers lost interest in applying logic, and shifted their attention towards the development of other formalisms for a more efficient representation and manipulation of information. The breakthrough came thanks to the development of an efficient and flexible inference rule in 1965, named *resolution*, that allowed applying logic for automated problem solving by the computer, and theorem proving finally gained an established position in artificial intelligence and, more recently, in the computer science as a whole as well.

Logic can directly be used as a knowledge-representation formalism for building expert systems; currently however, this is done only on a small scale. But then, the clear semantics of logic makes the formalism eminently suitable as a point of departure for understanding what the other knowledge-representation formalisms are all about. In this chapter, we first discuss the subject of how knowledge can be represented in logic, departing from propositional logic, which although having a rather limited expressiveness, is very useful for introducing several important notions. First-order predicate logic, which offers a much richer language for knowledge representation, is treated in Section 2.2. The major part of this chapter however will be devoted to the algorithmic aspects of applying logic in an automated reasoning system, and resolution in particular will be the subject of study.

2.1 Propositional logic

Propositional logic may be viewed as a representation language which allows us to express and reason with statements that are either *true* or *false*. Examples of such statements are:

‘A full-adder is a logical circuit’
‘10 is greater than 90’

Clearly, such statement need not be true. Statements like these are called *propositions* and are usually denoted in propositional logic by uppercase letters. Simple propositions such as P and Q are called *atomic propositions* or *atoms* for short. Atoms can be combined with so-called *logical connectives* to yield *composite propositions*. In the language of propositional logic, we have the following five connectives at our disposal:

negation:	$\neg$	(not)
conjunction:	$\wedge$	(and)
disjunction:	$\vee$	(or)
implication:	$\rightarrow$	(if then)
bi-implication:	$\leftrightarrow$	(if and only if)

For example, when we assume that the propositions G and D have the following meaning

$G =$ ‘A Bugatti is a car’
$D =$ ‘A Bugatti has 5 wheels’

then the composite proposition

$G \wedge D$

has the meaning:

‘A Bugatti is a car *and* a Bugatti has 5 wheels’

However, not all formulas consisting of atoms and connectives are (composite) propositions. In order to distinguish syntactically correct formulas that do represent propositions from those that do not, the notion of a well-formed formula is introduced in the following definition.

Definition 2.1 *A* well-formed formula *in propositional logic is an expression having one of the following forms:*

(1) An atom is a well-formed formula.

(2) If F is a well-formed formula, then $(\neg F)$ is a well-formed formula.

(3) If F and G are well-formed formulas, then $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$ and $(F \leftrightarrow G)$ are well-formed formulas.

(4) No other formula is well-formed.

EXAMPLE 2.1

Both formulas $(F \wedge (G \rightarrow H))$ and $(F \vee (\neg G))$ are well-formed according to the previous definition, but the formula $(\rightarrow H)$ is not.

In well-formed formulas, parentheses may be omitted as long as no ambiguity can occur; the adopted priority of the connectives is, in decreasing order, as follows:

$$\neg \;\wedge\; \vee \;\rightarrow\; \leftrightarrow$$

In the following, the term formula is used as an abbreviation when a well-formed formula is meant.

EXAMPLE 2.2

The formula $P \rightarrow Q \wedge R$ is the same as the formula $(P \rightarrow (Q \wedge R))$.

The notion of well-formedness of formulas only concerns the syntax of formulas in propositional logic: it does not express the formulas to be either *true* or *false*. In other words, it tells us nothing with respect to the semantics or meaning of formulas in propositional logic. The truth or falsity of a formula is called its *truth value*. The meaning of a formula in propositional logic is defined by means of a function $w : \text{PROP} \rightarrow \{\mathit{true}, \mathit{false}\}$ which assigns to each proposition in the set of propositions PROP either the truth value *true* or *false*. Consequently, the information that the atom P has the truth value *true*, is now denoted by $w(P) = \mathit{true}$, and the information that the atom P has the truth value *false*, is denoted by $w(P) = \mathit{false}$. Such a function w is called an interpretation function, or an *interpretation* for short, if it satisfies the following properties (we assume F and G to be arbitrary well-formed formulas):

(1) $w(\neg F) = \mathit{true}$ if $w(F) = \mathit{false}$, and $w(\neg F) = \mathit{false}$ if $w(F) = \mathit{true}$.

(2) $w(F \wedge G) = \mathit{true}$ if $w(F) = \mathit{true}$ and $w(G) = \mathit{true}$; otherwise $w(F \wedge G) = \mathit{false}$.

(3) $w(F \vee G) = \mathit{false}$ if $w(F) = \mathit{false}$ and $w(G) = \mathit{false}$; in all other cases, that is, if at least one of the function values $w(F)$ and $w(G)$ equals *true*, we have $w(F \vee G) = \mathit{true}$.

(4) $w(F \rightarrow G) = \mathit{false}$ if $w(F) = \mathit{true}$ and $w(G) = \mathit{false}$; in all other cases we have $w(F \rightarrow G) = \mathit{true}$.

(5) $w(F \leftrightarrow G) = \mathit{true}$ if $w(F) = w(G)$; otherwise $w(F \leftrightarrow G) = \mathit{false}$.

Table 2.1: The meanings of the connectives.

F	G	$\neg F$	$F \wedge G$	$F \vee G$	$F \rightarrow G$	$F \leftrightarrow G$
true	*true*	*false*	*true*	*true*	*true*	*true*
true	*false*	*false*	*false*	*true*	*false*	*false*
-*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*false*	*true*	*false*	*false*	*true*	*true*

Table 2.2: Truth table for $P \rightarrow (\neg Q \wedge R)$.

P	Q	R	$\neg Q$	$\neg Q \wedge R$	$P \rightarrow (\neg Q \wedge R)$
true	*true*	*true*	*false*	*false*	*false*
true	*true*	*false*	*false*	*false*	*false*
true	*false*	*true*	*true*	*true*	*true*
true	*false*	*false*	*true*	*false*	*false*
false	*true*	*true*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*true*
false	*false*	*true*	*true*	*true*	*true*
false	*false*	*false*	*true*	*false*	*true*

These rules are summarized in Table 2.1. The first two columns in this table list all possible combinations of truth values for the atomic propositions F and G; the remaining columns define the meanings of the respective connectives. If w is an interpretation which assigns to a given formula F the truth value *true*, then w is called a *model* for F.

By repeated applications of the rules listed in table 2.1, it is possible to express the truth value of an arbitrary formula in terms of the truth values of the atoms the formula is composed of. In a formula containing n different atoms, there are 2^n possible ways of assigning truth values to the atoms in the formula.

EXAMPLE 2.3

Table 2.2 lists all possible combinations of truth values for the atoms in the formula $P \rightarrow (\neg Q \wedge R)$; for each combination, the resulting truth value for this formula is determined. Such a table where all possible truth values for the atoms in a formula F are entered together with the corresponding truth value for the whole formula F, is called a *truth table.*

Definition 2.2 *A formula is called a* valid *formula if it is true under all interpretations. A valid formula is often called a* tautology. *A formula is called* invalid *if it is not valid.*

So, a valid formula is true regardless of the truth or falsity of its constituent atoms.

EXAMPLE 2.4

The formula $((P \rightarrow Q) \wedge P) \rightarrow Q$ is an example of a valid formula. In the previous example we dealt with an invalid formula.

Definition 2.3 *A formula is called* unsatisfiable *or* inconsistent *if the formula is false under all interpretations. An unsatisfiable formula is also called a* contradiction. *A formula is called* satisfiable *or* consistent *if it is not unsatisfiable.*

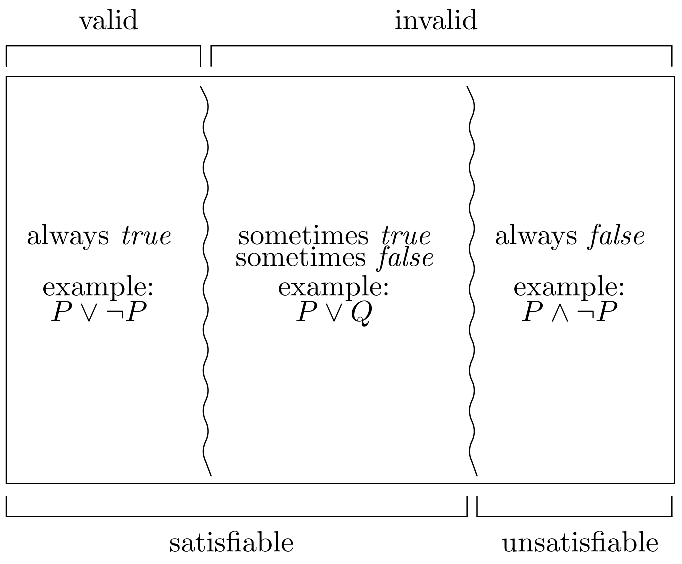


Figure 2.1: Relationship between validity and satisfiability.

Table 2.3: Truth table of $\neg(P \wedge Q)$ and $\neg P \vee \neg Q$.

P	Q	$\neg(P \wedge Q)$	$\neg P \vee \neg Q$
true	*true*	*false*	*false*
true	*false*	*true*	*true*
false	*true*	*true*	*true*
false	*false*	*true*	*true*

Note that a formula is valid precisely when its negation is unsatisfiable and vice versa.

EXAMPLE 2.5

The formulas $P \wedge \neg P$ and $(P \rightarrow Q) \wedge (P \wedge \neg Q)$ are both unsatisfiable.

Figure 2.1 depicts the relationships between the notions of valid, invalid, and satisfiable, and unsatisfiable formulas.

Definition 2.4 *Two formulas F and G are called* equivalent*, written as $F \equiv G$, if the truth values of F and G are the same under all possible interpretations.*

Two formulas can be shown to be equivalent by demonstrating that their truth tables are identical.

EXAMPLE 2.6

Table 2.3 shows that $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$.

Using truth tables the logical equivalences listed in Table 2.4 can easily be proven. These equivalences are called *laws of equivalence*. Law (a) is called the *law of double negation*; the laws (b) and (c) are called the *commutative laws*; (d) and (e) are the so-called *associative*

Table 2.4: Laws of equivalence.

$\neg(\neg F) \equiv F$	(a)
$F \vee G \equiv G \vee F$	(b)
$F \wedge G \equiv G \wedge F$	(c)
$(F \wedge G) \wedge H \equiv F \wedge (G \wedge H)$	(d)
$(F \vee G) \vee H \equiv F \vee (G \vee H)$	(e)
$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$	(f)
$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$	(g)
$F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F)$	(h)
$F \rightarrow G \equiv \neg F \vee G$	(i)
$\neg(F \wedge G) \equiv \neg F \vee \neg G$	(j)
$\neg(F \vee G) \equiv \neg F \wedge \neg G$	(k)

laws, and (f) and (g) are the *distributive laws*. The laws (j) and (k) are known as the *laws of De Morgan*. These laws often are used to transform a given well-formed formula into a logically equivalent but syntactically different formula.

In the following, a conjunction of formulas is often written as a set of formulas, where the elements of the set are taken as the conjunctive subformulas of the given formula.

EXAMPLE 2.7

The set $S = \{F \vee G, H\}$ represents the following formula: $(F \vee G) \wedge H$.

Truth tables can be applied to determine whether or not a given formula follows logically from a given set of formulas. Informally speaking, a formula logically follows from a set of formulas if it is satisfied by all interpretations satisfying the given set of formulas; we say that the formula is a logical consequence of the formulas in the given set. The following is a formal definition of this notion.

Definition 2.5 *A formula G is said to be a* logical consequence *of the set of formulas $F = \{F_1, \ldots, F_n\}$, $n \geq 1$, denoted by $F \vDash G$, if for each interpretation w for which $w(F_1 \wedge \cdots \wedge F_n) = true$, we have $w(G) = true$.*

EXAMPLE 2.8

The formula R is a logical consequence of the set of formulas $\{P \wedge \neg Q, P \rightarrow R\}$. Thus we can write $\{P \wedge \neg Q, P \rightarrow R\} \vDash R$.

Note that another way of stating that two formulas F and G are logically equivalent, that is, $F \equiv G$, is to say that both $\{F\} \vDash G$ and $\{G\} \vDash F$ hold. This tells us that the truth value of F and G are explicitly related to each other, which can also be expressed as $\vDash (F \leftrightarrow G)$.

Satisfiability, validity, equivalence and logical consequence are *semantic* notions; these properties are generally established using truth tables. However, for deriving logical consequences from of a set of formulas for example, propositional logic provides other techniques than using truth tables as well. It is possible to derive logical consequences by *syntactic* operations only. A formula which is derived from a given set of formulas then is guaranteed to be a logical consequence of that set if the syntactic operations employed meet certain conditions. Systems in which such syntactic operations are defined, are called (*formal*) *deduction systems*. Various sorts of deduction systems are known. An example of a deduction system is an *axiomatic system*, consisting of a formal language, such as the language of propositional logic described above, a set of *inference rules* (the syntactic operations) and a set of *axioms*. In Section 2.4 we shall return to the subject of logical deduction.

2.2 First-order predicate logic

In propositional logic, atoms are the basic constituents of formulas which are either *true* or *false*. A limitation of propositional logic is the impossibility to express general statements concerning similar cases. *First-order predicate logic* is more expressive than propositional logic, and such general statements can be specified in its language. Let us first introduce the language of first-order predicate logic. The following symbols are used:

- *Predicate symbols*, usually denoted by uppercase letters. Each predicate symbol has associated a natural number n, $n \geq 0$, indicating the number of arguments the predicate symbol has; the predicate symbol is called an *n-place* predicate symbol. 0-place or *nullary* predicate symbols are also called (*atomic*) *propositions*. One-place, two-place and three-place predicate symbols are also called *unary*, *binary* and *ternary* predicate symbols, respectively.

- *Variables*, usually denoted by lowercase letters from the end of the alphabet, such as x, y, z, possibly indexed with a natural number.

- *Function symbols*, usually denoted by lowercase letters halfway the alphabet. Each function symbol has associated a natural number n, $n \geq 0$, indicating its number of arguments; the function symbol is called *n-place*. Nullary function symbols are usually called *constants*.

- The *logical connectives* which have already been discussed in the previous section.

- Two *quantifiers*: the *universal quantifier* $\forall$, and the *existential quantifier* $\exists$. The quantifiers should be read as follows: if x is a variable, then $\forall x$ means 'for each x' or 'for all x', and $\exists x$ means 'there exists an x'.

- A number of *auxiliary symbols* such as parentheses and commas.

Variables and functions in logic are more or less similar to variables and functions in for instance algebra or calculus.

Before we define the notion of an atomic formula in predicate logic, we first introduce the notion of a term.

Definition 2.6 *A* term *is defined as follows:*

(1) A constant is a term.

(2) A variable is a term.

(3) If f is an n-place function symbol, $n \geq 1$, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

(4) *Nothing else is a term.*

So, a term is either a constant, a variable or a function of terms. Recall that a constant may also be viewed as a nullary function symbol. An atomic formula now consists of a predicate symbol and a number of terms to be taken as the arguments of the predicate symbol.

Definition 2.7 *An* atomic formula*, or* atom *for short, is an expression of the form $P(t_1, \ldots, t_n)$, where P is an n-place predicate symbol, $n \geq 0$, and $t_1, \ldots, t_n$ are terms.*

EXAMPLE 2.9

If P is a unary predicate symbol and x is a variable, then $P(x)$ is an atom. $Q(f(y), c, g(f(x), z))$ is an atom if Q is a ternary predicate symbol, c is a constant, f a unary function symbol, g a binary function symbol, and x, y and z are variables. For the same predicate symbols P and Q, $P(Q)$ is not an atom, because Q is not a term but a predicate symbol.

Composite formulas can be formed using the five connectives given in Section 2.1, together with the two quantifiers $\forall$ and $\exists$ just introduced. As was done for propositional logic, we now define the notion of a well-formed formula in predicate logic. The following definition also introduces the additional notions of free and bound variables.

Definition 2.8 *A* well-formed formula *in predicate logic, and the set of* free variables *of a well-formed formula are defined as follows:*

(1) *An atom is a well-formed formula. The set of free variables of an atomic formula consists of all the variables occurring in the terms in the atom.*

(2) *Let F be a well-formed formula with an associated set of free variables. Then, $(\neg F)$ is a well-formed formula. The set of free variables of $(\neg F)$ equals the set of free variables of F.*

(3) *Let F and G be well-formed formulas and let for each of these formulas a set of free variables be given. Then, $(F \vee G)$, $(F \wedge G)$, $(F \rightarrow G)$ and $(F \leftrightarrow G)$ are well-formed formulas. The set of free variables of each of these last mentioned formulas is equal to the union of the sets of free variables of F and G.*

(4) *If F is well-formed formula and x is an element of the set of free variables of F, then both $(\forall x F)$ and $(\exists x F)$ are well-formed formulas. The set of free variables of each of these formulas is equal to the set of free variables of F from which the variable x has been removed. The variable x is called* bound *by the quantifier $\forall$ or $\exists$.*

(5) *Nothing else is a well-formed formula.*

Note that we have introduced the notion of a formula in the preceding definition only from a purely syntactical point of view: nothing has been said about the meaning of such a formula.

Parentheses will be omitted from well-formed formulas as long as ambiguity cannot occur; the quantifiers then have a higher priority than the connectives.

Definition 2.9 *A well-formed formula is called a* closed formula*, or a* sentence*, if its set of free variables is empty; otherwise it is called an* open formula.

EXAMPLE 2.10

The set of free variables of the formula $\forall x \exists y(P(x) \rightarrow Q(y,z))$ is equal to $\{z\}$. So, only one of the three variables in the formula is a free variable. The formula $\forall x(P(x) \vee R(x))$ has no free variables at all, and thus is an example of a sentence.

In what follows, we shall primarily be concerned with closed formulas; the term formula will be used to mean a closed formula, unless explicitly stated otherwise.

In the formula $\forall x(A(x) \rightarrow G(x))$ all occurrences of the variable x in $A(x) \rightarrow G(x)$ are governed by the associated universal quantifier; $A(x) \rightarrow G(x)$ is called the *scope* of this quantifier.

EXAMPLE 2.11

The scope of the universal quantifier in the formula

$$\forall x(P(x) \rightarrow \exists y R(x,y))$$

is $P(x) \rightarrow \exists y R(x,y)$; the scope of the existential quantifier is the subformula $R(x,y)$.

In propositional logic, the truth value of a formula under a given interpretation is obtained by assigning either the truth value *true* or *false* to each of its constituent atoms according to this specific interpretation. Defining the semantics of first-order predicate logic is somewhat more involved than in propositional logic. In predicate logic, a structure representing the 'reality' is associated with the *meaningless* set of symbolic formulas: in a structure the objects or elements of the domain of discourse, or domain for short, are enlisted, together with functions and relations defined on the domain.

Definition 2.10 *A* structure S *is a tuple*

$$S = (D, \{\bar{f}_i^n : D^n \rightarrow D, n \geq 1\}, \{\bar{P}_i^m : D^m \rightarrow \{\mathit{true}, \mathit{false}\}, m \geq 0\})$$

having the following components:

(1) A non-empty set of elements D, called the domain *of S;*

(2) A set of functions *defined on D^n, $\{\bar{f}_i^n : D^n \rightarrow D, n \geq 1\}$;*

(3) A non-empty set of mappings, called predicates*, from D^m to the set of truth values $\{\mathit{true}, \mathit{false}\}$, $\{\bar{P}_i^m : D^m \rightarrow \{\mathit{true}, \mathit{false}\}, m \geq 0\}$.*

The basic idea underlying the definition of a structure is that we associate functions $\bar{f}_i^n$ to function symbols f_i and predicates $\bar{P}_i^m$ to predicate symbols P_i. Hence, we have to express how a given meaningless formula should be interpreted in a given structure: it is not possible to state anything about the truth value of a formula as long as it has not been prescribed which elements from the structure are to be associated with the elements in the formula.

EXAMPLE 2.12

Consider the formula $A(c)$. We associate the predicate having the intended meaning 'is a car' with the predicate symbol A. The formula should be *true* if the constant representing a Bugatti is associated with c; on the other hand, the same formula should be *false* if the constant representing a Volvo truck is associated with c. However, if we associate the predicate 'Truck' with A, the truth values of $A(c)$ for the two constants should be opposite to the ones mentioned before.

In the following definition, we introduce the notion of an assignment, which is a function that assigns elements from the domain of a structure to the variables in a formula.

Definition 2.11 *An* assignment *(*valuation*) v to a set of formulas F in a given structure S with domain D is a mapping from the set of variables in F to D.*

The interpretation of (terms and) formulas in a structure S under an assignment v now consists of the following steps. First, the constants in the formulas are assigned elements from D. Secondly, the variables are replaced by the particular elements from D that have been assigned to them by v. Then, the predicate and function symbols occurring in the formulas are assigned predicates and functions from S. Finally, the truth values of the formulas are determined.

Before the notion of an interpretation is defined more formally, a simple example in which no function symbols occur, is given. For the reader who is not interested in the formal aspects of logic, it suffices to merely study this example.

EXAMPLE 2.13

The open formula

$$F = A(x) \rightarrow O(x)$$

contains the unary predicate symbols A and O, and the free variable x. Consider the structure S consisting of the domain $D = \{bugatti, volvo\text{-}truck, alfa\text{-}romeo\}$ and the set of predicates comprising of the following elements:

- a unary predicate *Car*, with the intented meaning 'is a car', defined by $Car(bugatti) = true$, $Car(alfa\text{-}romeo) = true$ and $Car(volvo\text{-}truck) = false$, and
- the unary predicate *FourWheels* with the intended meaning 'has four wheels', defined by $FourWheels(bugatti) = false$, $FourWheels(volvo\text{-}truck) = false$ and $FourWheels(alfa\text{-}romeo) = true$.

Let us take for the predicate symbol A the predicate *Car*, and for the predicate symbol O the predicate *FourWheels*. It will be obvious that the atom $A(x)$ is *true* in S under any assignment v for which $Car(v(x)) = true$; so, for example for the assignment $v(x) = alfa\text{-}romeo$, we have that $A(x)$ is *true* in S under v. Furthermore, F is *true* in the structure S under the assignment v with $v(x) = alfa\text{-}romeo$, since $A(x)$ and $O(x)$ are both *true* in S under v. On the other hand, F is *false* in the structure S under the assignment v' with $v'(x) = bugatti$, because $Car(bugatti) = true$ and $FourWheels(bugatti) = false$ in S. Now, consider the closed formula

$$F' = \forall x(A(x) \rightarrow O(x))$$

and again the structure S. It should be obvious that F' is *false* in S.

Table 2.5: Laws of equivalence for quantifiers.

$\neg\exists x P(x) \equiv \forall x \neg P(x)$	(a)
$\neg\forall x P(x) \equiv \exists x \neg P(x)$	(b)
$\forall x(P(x) \wedge Q(x)) \equiv \forall x P(x) \wedge \forall x Q(x)$	(c)
$\exists x(P(x) \vee Q(x)) \equiv \exists x P(x) \vee \exists x Q(x)$	(d)
$\forall x P(x) \equiv \forall y P(y)$	(e)
$\exists x P(x) \equiv \exists y P(y)$	(f)

Definition 2.12 *An* interpretation *of terms in a structure $S = (D, \{\bar{f}_i^n\}, \{\bar{P}_i^m\})$ under an assignment v, denoted by I_v^S, is defined as follows:*

(1) $I_v^S(c_i) = d_i$, $d_i \in D$, where c_i is a constant.

(2) $I_v^S(x_i) = v(x_i)$, where x_i is a variable.

(3) $I_v^S(f_i^n(t_1, \ldots, t_n)) = \bar{f}_i^n(I_v^S(t_1), \ldots, I_v^S(t_n))$, where $\bar{f}_i^n$ is a function from S associated with the function symbol f_i^n.

The truth value of a formula in a structure S under an assignment v for a given interpretation I_v^S is obtained as follows:

(1) $I_v^S(P_i^m(t_1, \ldots, t_m)) = \bar{P}_i^m(I_v^S(t_1), \ldots, I_v^S(t_m))$, meaning that an atom $P_i^m(t_1, \ldots, t_m)$ is true in the structure S under the assignment v for the interpretation I_v^S if $\bar{P}_i^m(I_v^S(t_1), \ldots, I_v^S(t_m))$ is true, where $\bar{P}_i^m$ is the predicate from S associated with P_i^m.

(2) If the truth values of the formulas F and G have been determined, then the truth values of $\neg F$, $F \wedge G$, $F \vee G$, $F \rightarrow G$ and $F \leftrightarrow G$ are defined by the meanings of the connectives as listed in Table 2.1.

(3) $\exists x F$ is true under v if there exists an assignment v' differing from v at most with regard to x, such that F is true under v'.

(4) $\forall x F$ is true under v if for each v' differing from v at most with regard to x, F is true under v'.

The notions valid, invalid, satisfiable, unsatisfiable, logical consequence, equivalence and model have meanings in predicate logic similar to their meanings in propositional logic. In addition to the equivalences listed in Table 2.4, predicate logic also has some laws of equivalence for quantifiers, which are given in Table 2.5. Note that the properties $\forall x(P(x) \vee Q(x)) \equiv \forall x P(x) \vee \forall x Q(x)$ and $\exists x(P(x) \wedge Q(x)) \equiv \exists x P(x) \wedge \exists x Q(x)$ do *not* hold.

We conclude this subsection with another example.

EXAMPLE 2.14

We take the unary (meaningless) predicate symbols C, F, V, W and E, and the constants a and b from a given first-order language. Now, consider the following formulas:

(1) $\forall x(C(x) \rightarrow V(x))$

(2) $F(a)$

(3) $\forall x(F(x) \to C(x))$

(4) $\neg E(a)$

(5) $\forall x((C(x) \wedge \neg E(x)) \to W(x))$

(6) $F(b)$

(7) $\neg W(b)$

(8) $E(b)$

Consider the structure S in the reality with a domain consisting of the elements and *bugatti*, which are assigned to the constants a and b, respectively. The set of predicates in S comprises the unary predicates *Car*, *Fast*, *Vehicle*, *FourWheels*, and *Exception*, which are taken for the predicate symbols C, F, V, W, and E, respectively. The structure S and the mentioned interpretation have been carefully chosen so as to satisfy the above-given closed formulas, for instance by giving the following intended meaning to the predicates:

Car	=	'is a car'
Fast	=	'is a fast car'
Vehicle	=	'is a vehicle'
FourWheels	=	'has four wheels'
Exception	=	'is an exception'

In the given structure S, the formula numbered 1 expresses the knowledge that every car is a vehicle. The fact that an alfa-romeo is a fast car, has been stated in formula 2. Formula 3 expresses that every fast car is a car, and formula 4 states that an alfa-romeo is not an exception to the rule that cars have four wheels, which has been formalized in logic by means of formula 5. A Bugatti is a fast car (formula 6), but contrary to an alfa-romeo it does not have 4 wheels (formula 7), and therefore is an exception to the last mentioned rule; the fact that Bugattis are exceptions is expressed by means of formula 8.

It should be noted that in another structure with another domain and other predicates, the formulas given above might have completely different meanings.

2.3 Clausal form of logic

Before turning our attention to reasoning in logic, we introduce in this section a syntactically restricted form of predicate logic, called the *clausal form of logic*, which will play an important role in the remainder of this chapter. This restricted form however, can be shown to be as expressive as full first-order predicate logic. The clausal form of logic is often employed, in particular in the fields of theorem proving and logic programming.

We start with the definition of some new notions.

Definition 2.13 *A* literal *is an atom, called a* positive literal*, or a negation of an atom, called a* negative literal*.*

Definition 2.14 *A* clause *is a closed formula of the form*

$$\forall x_1 \cdots \forall x_s (L_1 \vee \cdots \vee L_m)$$

where each L_i, $i = 1, \ldots, m$, $m \geq 0$, is a literal, with $L_i \neq L_j$ for each $i \neq j$, and $x_1, \ldots, x_s$, $s \geq 0$, are variables occurring in $L_1 \vee \cdots \vee L_m$. If $m = 0$, the clause is said to be the empty clause*, denoted by $\Box$.*

The empty clause $\Box$ is interpreted as a formula which is always *false*, in other words, $\Box$ is an unsatisfiable formula.

A clause

$$\forall x_1 \cdots \forall x_s (A_1 \vee \cdots \vee A_k \vee \neg B_1 \vee \cdots \vee \neg B_n)$$

where $A_1, \ldots, A_k, B_1, \ldots, B_n$ are atoms and $x_1, \ldots, x_s$ are variables, is equivalent to

$$\forall x_1 \cdots \forall x_s (B_1 \wedge \cdots \wedge B_n \rightarrow A_1 \vee \cdots \vee A_k)$$

as a consequence of the laws $\neg F \vee G \equiv F \rightarrow G$ and $\neg F \vee \neg G \equiv \neg(F \wedge G)$, and is often written as

$$A_1, \ldots, A_k \leftarrow B_1, \ldots, B_n$$

The last notation is the more conventional one in logic programming. The commas in $A_1, \ldots, A_k$ each stand for a disjunction, and the commas in $B_1, \ldots, B_n$ indicate a conjunction. $A_1, \ldots, A_k$ are called the *conclusions* of the clause, and $B_1, \ldots, B_n$ the *conditions*.

Each well-formed formula in first-order predicate logic can be translated into a set of clauses, which is viewed as the conjunction of its elements. As we will see, this translation process may slightly alter the meaning of the formulas. We shall illustrate the translation process by means of an example. Before proceeding, we define two normal forms which are required for the translation process.

Definition 2.15 *A formula F is in* prenex normal form *if F is of the form*

$$Q_1 x_1 \cdots Q_n x_n M$$

where each Q_i, $i = 1, \ldots, n$, $n \geq 1$, equals one of the two quantifiers $\forall$ and $\exists$, and where M is a formula in which no quantifiers occur. $Q_1 x_1 \ldots Q_n x_n$ is called the prefix *and M is called the* matrix *of the formula F.*

Definition 2.16 *A formula F in prenex normal form is in* conjunctive normal form *if the matrix of F is of the form*

$$F_1 \wedge \cdots \wedge F_n$$

where each F_i, $i = 1, \ldots, n$, $n \geq 1$, is a disjunction of literals.

EXAMPLE 2.15

Consider the following three formulas:

$$\forall x (P(x) \vee \exists y Q(x, y))$$
$$\forall x \exists y \forall z ((P(x) \wedge Q(x, y)) \vee \neg R(z))$$
$$\forall x \exists y ((\neg P(x) \vee Q(x, y)) \wedge (P(y) \vee \neg R(x)))$$

The first formula is not in prenex normal form because of the occurrence of an existential quantifier in the 'inside' of the formula. The other two formulas are both in prenex normal form; moreover, the last formula is also in conjunctive normal form.

The next example illustrates the translation of a well-formed formula into a set of clauses. The translation scheme presented in the example however is general and can be applied to any well-formed formula in first-order predicate logic.

EXAMPLE 2.16

Consider the following formula:

$$\forall x(\exists y P(x,y) \vee \neg\exists y(\neg Q(x,y) \rightarrow R(f(x,y))))$$

This formula is transformed in eight steps, first into prenex normal form, subsequently into conjunctive normal form, amongst others by applying the laws of equivalence listed in the tables 2.4 and 2.5, and finally into a set of clauses.

Step 1. Eliminate all implication symbols using the equivalences $F \rightarrow G \equiv \neg F \vee G$ and $\neg(\neg F) \equiv F$:

$$\forall x(\exists y P(x,y) \vee \neg\exists y(Q(x,y) \vee R(f(x,y))))$$

If a formula contains bi-implication symbols, these can be removed by applying the equivalence

$$F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F)$$

Step 2. Diminish the scope of the negation symbols in such a way that each negation symbol only governs a single atom. This can be accomplished by using the equivalences $\neg\forall x F(x) \equiv \exists x \neg F(x)$, $\neg\exists x F(x) \equiv \forall x \neg F(x)$, $\neg(\neg F) \equiv F$, together with the laws of De Morgan:

$$\forall x(\exists y P(x,y) \vee \forall y(\neg Q(x,y) \wedge \neg R(f(x,y))))$$

Step 3. Rename the variables in the formula using the equivalences $\forall x F(x) \equiv \forall y F(y)$ and $\exists x F(x) \equiv \exists y F(y)$, so that each quantifier has its own uniquely named variable:

$$\forall x(\exists y P(x,y) \vee \forall z(\neg Q(x,z) \wedge \neg R(f(x,z))))$$

Formulas only differing in the names of their bound variables are called *variants*.

Step 4. Eliminate all existential quantifiers. For any existentially quantified variable x not lying within the scope of a universal quantifier, all occurrences of x in the formula within the scope of the existential quantifier can be replaced by a new, that is, not previously used, constant symbol c. The particular existential quantifier may then be removed. For instance, the elimination of the existential quantifier in the formula $\exists x P(x)$ yields a formula $P(c)$. However, if an existentially quantified variable y lies within the scope of one or more universal quantifiers with the variables $x_1, \ldots, x_n$, $n \geq 1$, then the variable y may be functionally dependent upon $x_1, \ldots, x_n$. Let this dependency be represented explicitly by means of a new n-place function symbol g such that $g(x_1, \ldots, x_n) = y$. All occurrences of y within the scope of the existential quantifier then are replaced by the function term $g(x_1, \ldots, x_n)$, after which the existential quantifier may be removed. The constants and functions introduced in order to allow for the elimination of existential quantifiers are called *Skolem functions.*

The existentially quantified variable y in the example lies within the scope of the universal quantifier with the variable x, and is replaced by $g(x)$:

$$\forall x(P(x, g(x)) \vee \forall z(\neg Q(x, z) \wedge \neg R(f(x, z))))$$

Note that by replacing the existentially quantified variables by Skolem functions, we lose logical equivalence. Fortunately, it can be shown that a formula F is satisfiable if and only if the formula F', obtained from F by replacing existentially quantified variables in F by Skolem functions, is satisfiable as well. In general, the satisfiability of F and F' will not be based on the same model, since F' contains function symbols not occurring in F. In the following, it will become evident that this property is sufficient for our purposes.

Step 5. Transform the formula into prenex normal form, by placing all the universal quantifiers in front of the formula:

$$\forall x \forall z(P(x, g(x)) \vee (\neg Q(x, z) \wedge \neg R(f(x, z))))$$

Note that this is allowed because by step 3 each quantifier applies to a uniquely named variable; this means that the scope of all quantifiers is the entire formula.

Step 6. Bring the matrix in conjunctive normal form using the distributive laws:

$$\forall x \forall z((P(x, g(x)) \vee \neg Q(x, z)) \wedge (P(x, g(x)) \vee \neg R(f(x, z))))$$

Step 7. Select the matrix by disregarding the prefix:

$$(P(x, g(x)) \vee \neg Q(x, z)) \wedge (P(x, g(x)) \vee \neg R(f(x, z)))$$

All variables in the matrix are now implicitly considered to be universally quantified.

Step 8. Translate the matrix into a set of clauses, by replacing formulas of the form $F \wedge G$ by a set of clauses $\{F', G'\}$, where F' and G' indicate that F and G are now represented using the notational convention of logic programming:

$$\{P(x, g(x)) \leftarrow Q(x, z)), P(x, g(x)) \leftarrow R(f(x, z))i\}$$

or the notation used in automated theorem proving:

$$\{P(x, g(x)) \vee \neg Q(x, z),\ P(x, g(x)) \vee \neg R(f(x, z))\}$$

We conclude this subsection with the definition of a special type of clause, a so-called Horn clause, which is a clause containing at most one positive literal.

Definition 2.17 *A* Horn clause *is a clause having one of the following forms:*

(1) $A \leftarrow$

(2) $\leftarrow B_1, \ldots, B_n$*,* $n \geq 1$

(3) $A \leftarrow B_1, \ldots, B_n$*,* $n \geq 1$

A clause of the form 1 is called a unit clause*; a clause of form 2 is called a* goal clause*.*

Horn clauses are employed in the programming language PROLOG. We will return to this observation in Section 2.7.2.

2.4 Reasoning in logic: inference rules

In the Sections 2.1 and 2.2 we described how a meaning could be attached to a meaningless set of logical formulas. This is sometimes called the *declarative semantics* of logic. The declarative semantics offers a means for investigating for example whether or not a given formula is a logical consequence of a set of formulas. However, it is also possible to answer this question without examining the semantic contents of the formulas concerned, by applying so-called *inference rules.* Contrary to truth tables, inference rules are purely syntactic operations which only are capable of modifying the form of the elements of a given set of formulas. Inference rules either add, replace or remove formulas; most inference rules discussed in this book however add new formulas to a given set of formulas. In general, an inference rule is given as a schema in which a kind of meta-variables occur that may be substituted by arbitrary formulas. An example of such a schema is shown below:

$$\frac{A, A \rightarrow B}{B}$$

The formulas above the line are called the *premises*, and the formula below the line is called the *conclusion* of the inference rule. The above-given inference rule is known as *modus ponens*, and when applied, removes an implication from a formula. Another example of an inference rule, in this case for introducing a logical connective, is the following schema:

$$\frac{A, B}{A \wedge B}$$

Repeated applications of inference rules give rise to what is called a *derivation* or *deduction.* For instance, modus ponens can be applied to draw the conclusion S from the two formulas $P \wedge (Q \vee R)$ and $P \wedge (Q \vee R) \rightarrow S$. It is said that there exists a derivation of the formula S from the set of clauses $\{P \wedge (Q \vee R), P \wedge (Q \vee R) \rightarrow S\}$. This is denoted by:

$$\{P \wedge (Q \vee R), P \wedge (Q \vee R) \rightarrow S\} \vdash S$$

The symbol $\vdash$ is known as the *turnstile.*

EXAMPLE 2.17

Consider the set of formulas $\{P, Q, P \wedge Q \rightarrow S\}$. If the inference rule

$$\frac{A, B}{A \wedge B}$$

is applied to the formulas P and Q, the formula $P \wedge Q$ is derived; the subsequent application of modus ponens to $P \wedge Q$ and $P \wedge Q \rightarrow S$ yields S. So,

$$\{P, Q, P \wedge Q \rightarrow S\} \vdash S$$

Now that we have introduced inference rules, it is relevant to investigate how the declarative semantics of a particular class of formulas and its *procedural semantics*, described by means of inference rules, are interrelated: if these two notions are related to each other, we are in the desirable circumstance of being able to assign a meaning to formulas which have been derived using inference rules, simply by our knowledge of the declarative meaning of the original set of formulas. On the other hand, when starting with the known meaning of a set of formulas, it will then be possible to derive only formulas which can be related to that meaning. These two properties are known as the soundness and the completeness, respectively, of a collection of inference rules.

More formally, a collection of inference rules is said to be *sound* if and only if for each formula F derived by applying these inference rules on a given set of well-formed formulas S of a particular class (for example clauses), we have that F is a logical consequence of S. This property can be expressed more tersely as follows, using the notations introduced before:

if $S \vdash F$ then $S \vDash F$.

In other words, a collection of inference rules is sound if it preserves truth under the operations of a derivation. This property is of great importance, because only by applying sound inference rules it is possible to assign a meaning to the result of a derivation.

EXAMPLE 2.18

The previously discussed inference rule modus ponens is an example of a sound inference rule. From the given formulas F and $F \rightarrow G$, the formula G can be derived by applying modus ponens, that is, we have $\{F, F \rightarrow G\} \vdash G$. On the other hand, if $F \rightarrow G$ and F are both *true* under a particular interpretation w, then from the truth Table 2.1 we have that G is *true* under w as well. So, G is a logical consequence of the two given formulas: $\{F, F \rightarrow G\} \vDash G$.

The reverse property that by applying a particular collection of inference rules, each logical consequence F of a given set of formulas S can be derived, is called the *completeness* of the collection of inference rules:

if $S \vDash F$ then $S \vdash F$.

EXAMPLE 2.19

The collection of inference rules only consisting of modus ponens is not complete for all well-formed formulas in propositional logic. For example, it is not possible to derive the formula P from $\neg Q$ and $P \vee Q$, although P is a logical consequence of the two formulas. However, by combining modus ponens with other inference rules, it is possible to obtain a complete collection of inference rules.

The important question now arises if there exists a mechanical proof procedure, employing a particular sound and complete collection of inference rules, which is capable of determining whether or not a given formula F can be derived from a given set of formulas S. In 1936, A. Church and A.M. Turing showed, independently, that such a general proof procedure does not exist for first-order predicate logic. This property is called the *undecidability* of first-order predicate logic. All known proof procedures are only capable of deriving F from S (that is, are able to prove $S \vdash F$) if F is a logical consequence of S (that is, if $S \vDash F$); if F is not a logical consequence of S, then the proof procedure is not guaranteed to terminate.

However, for propositional logic there do exist proof procedures which always terminate and yield the right answer: for checking whether a given formula is a logical consequence of a certain set of formulas, we can simply apply truth tables. So, propositional logic is decidable.

The undecidability of first-order predicate logic has not refrained the research area of automated theorem proving from further progress. The major result of this research has been the development of an efficient and flexible inference rule, which is both sound and complete for proving inconsistency, called *resolution*. This is sometimes called *refutation completeness* (see below). However, the resolution rule is only suitable for manipulating formulas in clausal form. Hence, to use this inference rule on a set of arbitrary logical formulas in first-order predicate logic, it is required to translate each formula into the clausal form of logic by means of the procedure discussed in Section 2.3. This implies that resolution is *not* complete for *unrestricted* first-order predicate logic. The formulation of resolution as a suitable inference rule for automated theorem proving in the clausal form of logic has been mainly due to J.A. Robinson, who departed from earlier work by D. Prawitz. The final working-out of resolution in various algorithms, supplemented with specific implementation techniques, has been the work of a large number of researchers. Resolution is the subject of the remainder of this chapter.

2.5 Resolution and propositional logic

We begin this section with a brief, informal sketch of the principles of resolution. Consider a set of formulas S in clausal form. Suppose we are given a formula G, also in clausal form, for which we have to prove that it can be derived from S by applying resolution. Proving $S \vdash G$ is equivalent to proving that the set of clauses W, consisting of the clauses in S supplemented with the negation of the formula G, that is $W = S \cup \{\neg G\}$, is unsatisfiable. Resolution on W now proceeds as follows: first, it is checked whether or not W contains the empty clause $\Box$; if this is the case, then W is unsatisfiable, and G is a logical consequence of S. If the empty clause $\Box$ is not in W, then the resolution rule is applied on a suitable pair of clauses from W, yielding a new clause. Every clause derived this way is added to W, resulting in a new set of clauses on which the same resolution procedure is applied. The entire procedure is repeated until some generated set of clauses has been shown to contain the empty clause $\Box$, indicating unsatisfiability of W, or until all possible new clauses have been derived.

The basic principles of resolution are best illustrated by means of an example from propositional logic. In Section 2.6 we turn our attention to predicate logic.

EXAMPLE 2.20

Consider the following set of clauses:

$$\{C_1 = P \vee R, C_2 = \neg P \vee Q\}$$

These clauses contain *complementary* literals, that is, literals having opposite truth values, namely P and $\neg P$. Applying resolution, a new clause C_3 is derived being the disjunction of the original clauses C_1 and C_2 in which the complementary literals have been cancelled out. So, application of resolution yields the clause

$$C_3 = R \vee Q$$

which then is added to the original set of clauses.

The resolution principle is described more precisely in the following definition.

Definition 2.18 *Consider the two clauses C_1 and C_2 containing the literals L_1 and L_2 respectively, where L_1 and L_2 are complementary. The procedure of* resolution *proceeds as follows:*

(1) Delete L_1 from C_1 and L_2 from C_2, yielding the clauses C_1' and C_2';

(2) Form the disjunction C' of C_1' and C_2';

(3) Delete (possibly) redundant literals from C', thus obtaining the clause C.

The resulting clause C is called the resolvent *of C_1 and C_2. The clauses C_1 and C_2 are said to be the* parent clauses *of the resolvent.*

Resolution has the important property that when two given parent clauses are *true* under a given interpretation, their resolvent is *true* under the same interpretation as well: resolution is a sound inference rule. In the following theorem we prove that resolution is sound for the case of propositional logic.

THEOREM 1 *(*soundness of resolution*) Consider two clauses C_1 and C_2 containing complementary literals. Then, any resolvent C of C_1 and C_2 is a logical consequence of $\{C_1, C_2\}$.*

Proof: We are given that the two clauses C_1 and C_2 contain complementary literals. So, it is possible to write C_1 and C_2 as $C_1 = L \vee C_1'$ and $C_2 = \neg L \vee C_2'$ respectively for some literal L. By definition, a resolvent C is equal to $C_1' \vee C_2'$ from which possibly redundant literals have been removed. Now, suppose that C_1 and C_2 are both *true* under an interpretation w. We then have to prove that C is *true* under the same interpretation w as well. Clearly, either L or $\neg L$ is *false*. Suppose that L is *false* under w, then C_1 obviously contains more than one literal, since otherwise C_1 would be *false* under w. It follows that C_1' is *true* under w. Hence, $C_1' \vee C_2'$, and therefore also C, is *true* under w. Similarly, it can be shown that the resolvent is *true* under w if it is assumed that L is *true*. So, if C_1 and C_2 are *true* under w then C is *true* under w as well. Hence, C is a logical consequence of C_1 and C_2. $\diamondsuit$

Resolution is also a complete inference rule. Proving the completeness of resolution is beyond the scope of this book; we therefore confine ourselves to merely stating the property.

EXAMPLE 2.21

In the definition of a clause in Section 2.3, it was mentioned that a clause was not allowed to contain duplicate literals. This condition appears to be a necessary requirement for the completeness of resolution. For example, consider the following set of formulas:

$$S = \{P \vee P, \neg P \vee \neg P\}$$

It will be evident that S is unsatisfiable, since $P \vee P \equiv P$ and $\neg P \vee \neg P \equiv \neg P$. However, if resolution is applied to S then in every step the tautology $P \vee \neg P$ is derived. It is not possible to derive the empty clause $\Box$.

Until now we have used the notion of a derivation only in an intuitive sense. Before giving some more examples, we define the notion of a derivation in a formal way.

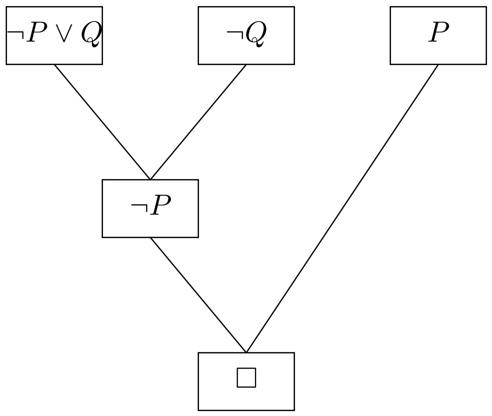


Figure 2.2: A refutation tree.

Definition 2.19 *Let S be a set of clauses and let C be a single clause. A* derivation *of C from S, denoted by $S \vdash_{\mathcal{B}} C$, is a finite sequence of clauses $C_1, C_2, \ldots, C_n$, $n \geq 1$, where each C_k either is a clause in S or a resolvent with parent clauses C_i and C_j, $i < k$, $j < k$, $i \neq j$, from the sequence, and $C = C_n$. If $C_n = \Box$, then the derivation is said to be a* refutation *of S, indicating that S is unsatisfiable.*

EXAMPLE 2.22

Consider the following set of clauses:

$$S = \{\neg P \vee Q, \neg Q, P\}$$

From $C_1 = \neg P \vee Q$ and $C_2 = \neg Q$ we obtain the resolvent $C_3 = \neg P$. From the clauses C_3 and $C_4 = P$ we derive $C_5 = \Box$. So, S is unsatisfiable. The sequence of clauses C_1, C_2, C_3, C_4, C_5 is a refutation of S. Note that it is not the only possible refutation of S. In general, a set S of clauses may have more than one refutation.

Notice that by the choice of the empty clause $\Box$ as a formula that is *false* under all interpretations, which is a *semantic* notion, the *proof-theoretical* notion of a refutation has obtained a suitable meaning. A derivation can be depicted in a graph, called a *derivation graph.* In the case of a refutation, the vertices in the derivation graph may be restricted to those clauses and resolvents which directly or indirectly contribute to the refutation. Such a derivation graph has the form of a tree and is usually called a *refutation tree.* The leaves of such a tree are clauses from the original set, and the root of the tree is the empty clause $\Box$. The refutation tree for the derivation discussed in the previous example is shown in Figure 2.2. Note that another refutation of S gives rise to another refutation tree.

2.6 Resolution and first-order predicate logic

An important feature of resolution in first-order predicate logic, taking place in the basic resolution method, is the manipulation of terms. This has not been dealt with in the previous section, where we only had atomic propositions, connectives and auxiliary symbols as building blocks for propositional formulas. In this section, we therefore first discuss the manipulation of terms, before we provide a detailed description of resolution in first-order predicate logic.

2.6.1 Substitution and unification

The substitution of terms for variables in formulas in order to make these formulas syntactically equal, plays a central role in a method known as *unification.* We first introduce the notion of substitution formally and then discuss its role in unification.

Definition 2.20 *A* substitution σ *is a finite set of the form*

$$\{t_1/x_1, \ldots, t_n/x_n\}$$

where each x_i is a variable and where each t_i is a term not equal to x_i, $i = 1, \ldots, n$, $n \geq 0$; the variables $x_1, \ldots, x_n$ differ from each other. An element t_i/x_i of a substitution σ is called a binding *for the variable x_i. If none of the terms t_i in a substitution contains a variable, we have a so-called* ground substitution*. The substitution defined by the empty set is called the* empty substitution*, and is denoted by ϵ.*

Definition 2.21 *An* expression *is a term, a literal, a conjunction of literals or a disjunction of literals; a* simple expression *is a term or an atom.*

A substitution σ can be applied to an expression E, yielding a new expression $E\sigma$ which is similar to E with the difference that the variables in E occurring in σ have been replaced by their associated terms.

Definition 2.22 *Let $\sigma = \{t_1/x_1, \ldots, t_n/x_n\}$, $n \geq 0$, be a substitution and E an expression. Then, $E\sigma$ is an expression obtained from E by simultaneously replacing all occurrences of the variables x_i by the terms t_i. $E\sigma$ is called an* instance *of E. If $E\sigma$ does not contain any variables, then $E\sigma$ is said to be a* ground instance *of E.*

EXAMPLE 2.23

Let $\sigma = \{a/x, w/z\}$ be a substitution and let $E = P(f(x, y), z)$ be an expression. Then, $E\sigma$ is obtained by replacing each variable x in E by the constant a and each variable z by the variable w. The result of the substitution is $E\sigma = P(f(a, y), w)$. Note that $E\sigma$ is not a ground instance.

The application of a substitution to a single expression can be extended to a set of expressions, as demonstrated in the following example.

EXAMPLE 2.24

Application of the substitution $\sigma = \{a/x, b/z\}$ to the set of expressions $\{P(x, f(x, z)), Q(x, w)\}$ yields the following set of instances:

$$\{P(x, f(x, z)), Q(x, w)\}\sigma = \{P(a, f(a, b)), Q(a, w)\}$$

The first element of the resulting set of instances is a ground instance; the second one is not ground, since it contains the variable w.

Definition 2.23 *Let* $\theta = \{t_1/x_1, \ldots, t_m/x_m\}$ *and* $\sigma = \{s_1/y_1, \ldots, s_n/y_n\}$*,* $m \geq 1$*,* $n \geq 1$*, be substitutions. The* composition *of these substitutions, denoted by* $\theta\sigma$*, is obtained by removing from the set*

$$\{t_1\sigma/x_1, \ldots, t_m\sigma/x_m, s_1/y_1, \ldots, s_n/y_n\}$$

all elements $t_i\sigma/x_i$ *for which* $x_i = t_i\sigma$*, and furthermore, all elements* s_j/y_j *for which* $y_j \in \{x_1, \ldots, x_m\}$

The composition of subtitutions is *associative*, i.e., for any expression E and substitutions ϕ, θ and σ we have that $E(\phi\sigma)\theta = E\phi(\sigma\theta)$; the operation is not commutative. Let $\theta = \{t_1/x_1, \ldots, t_m/x_m\}$ be a substitution, and let V be the set of variables occurring in $\{t_1, \ldots, t_m\}$, then θ is *idempotent*, i.e., $E(\theta\theta) = E\theta$, iff $V \cap \{x_1, \ldots, x_m\} = \varnothing$.

Note that the last definition gives us a means for replacing two substitutions by a single one, being the composition of these substitutions. However, it is not always necessary to actually compute the composition of two subsequent substitutions σ and θ before applying them to an expression E: it can easily be proven that $E(\sigma\theta) = (E\sigma)\theta$. The proof of this property is left to the reader as an exercise (see Exercise 2.11); here, we merely give an example.

EXAMPLE 2.25

Consider the expression $E = Q(x, f(y), g(z, x))$ and the two substitutions $\sigma = \{f(y)/x, z/y\}$ and $\theta = \{a/x, b/y, y/z\}$. We compute the composition $\sigma\theta$ of σ and θ: $\sigma\theta = \{f(b)/x, y/z\}$. Application of the compound substitution $\sigma\theta$ to E yields the instance $E(\sigma\theta) = Q(f(b), f(y), g(y, f(b)))$. We now compare this instance with $(E\sigma)\theta$. We first apply σ to E, resulting in $E\sigma = Q(f(y), f(z), g(z, f(y)))$. Subsequently, we apply θ to $E\sigma$ and obtain the instance $(E\sigma)\theta = Q(f(b), f(y), g(y, f(b)))$. So, for the given expression and substitutions, we have $E(\sigma\theta) = (E\sigma)\theta$.

In propositional logic, a resolvent of two parent clauses containing complementary literals, such as P and $\neg P$, was obtained by taking the disjunction of these clauses after cancelling out such a pair of complementary literals. It was easy to check for complementary literals in this case, since we only had to verify equality of the propositional atoms in the chosen literals and the presence of a negation in exactly one of them. Now, suppose that we want to compare the two literals $\neg P(x)$ and $P(a)$ occurring in two different clauses in first-order predicate logic. These two literals are 'almost' complementary. However, the first literal contains a variable as an argument of its predicate symbol, whereas the second one contains a constant. It is here where substitution comes in. Note that substitution can be applied to make expressions syntactically equal. Moreover, the substitution which is required to obtain syntactic equality of two given expressions also indicates the difference between the two. If we apply the substitution $\{a/x\}$ to the example above, we obtain syntactic equality of the two atoms $P(x)$ and $P(a)$. So, the two literals $\neg P(x)$ and $P(a)$ become complementary after substitution.

The *unification algorithm* is a general method for comparing expressions; the algorithm computes, if possible, the substitution that is needed to make the given expressions syntactically equal. Before we discuss the algorithm, we introduce some new notions.

Definition 2.24 *A substitution σ is called a* unifier *of a given set of expressions $\{E_1, \ldots, E_m\}$ if $E_1\sigma = \cdots = E_m\sigma, m \geq 2$. A set of expressions is called* unifiable *if it has a unifier.*

Definition 2.25 *A unifier θ of a unifiable set of expressions $E = \{E_1, \ldots, E_m\}$, $m \geq 2$, is said to be a* most general unifier *(mgu) if for each unifier σ of E there exists a substitution λ such that $\sigma = \theta\lambda$.*

A set of expressions may have more than one most general unifier; however, a most general unifier is unique but for a renaming of the variables.

EXAMPLE 2.26

Consider the set of expressions $\{R(x, f(a, g(y))), R(b, f(z, w))\}$. Some possible unifiers of this set are $\sigma_1 = \{b/x, a/z, g(c)/w, c/y\}$, $\sigma_2 = \{b/x, a/z, f(a)/y, g(f(a))/w\}$ and $\sigma_3 = \{b/x, a/z, g(y)/w\}$. The last unifier is also a most general unifier: by the composition of this unifier with the substitution $\{c/y\}$ we get σ_1; the second unifier is obtained by the composition of σ_3 with $\{f(a)/y\}$.

The unification algorithm, more precisely, is a method for constructing a most general unifier of a finite, non-empty set of expressions. The algorithm considered in this book operates in the following manner. First, the left-most subexpressions in which the given expressions differ is computed. Their difference is placed in a set, called the *disagreement set*. Based on this disagreement set a ('most general') substitution is computed, which is subsequently applied to the given expressions, yielding a partial or total equality. If no such substitution exists, the algorithm terminates with the message that the expressions are not unifiable. Otherwise, the procedure proceeds until each element within each of the expressions has been processed. It can be proven that the algorithm either terminates with a failure message or with a most general unifier of the finite, unifiable set of expressions.

EXAMPLE 2.27

Consider the following set of expressions:

$$S = \{Q(x, f(a), y), Q(x, z, c), Q(x, f(a), c)\}$$

The left-most subexpression in which the three expressions differ is in the second argument of the predicate symbol Q. So, the first disagreement set is $\{f(a), z\}$. By means of the substitution $\{f(a)/z\}$ the subexpressions in the second argument position are made equal. The next disagreement set is $\{y, c\}$. By means of the substitution $\{c/y\}$ these subexpressions are also equalized. The final result returned by the unification algorithm is the unifier $\{f(a)/z, c/y\}$ of S. It can easily be seen that this unifier is a most general one.

The following section shows an implementation of the unification algorithm. In the next section we discuss the role of unification in resolution in first-order predicate logic.

2.6.2 Resolution

Now that we have dealt with the subjects of substitution and unification, we are ready for a discussion of resolution in first-order predicate logic. We start with an informal introduction to the subject by means of an example.

EXAMPLE 2.28

Consider the following set of clauses:

$$\{C_1 = P(x) \vee Q(x), C_2 = \neg P(f(y)) \vee R(y)\}$$

As can be seen, the clauses C_1 and C_2 do not contain complementary literals. However, the atoms $P(x)$, occurring in C_1, and $P(f(y))$, occurring in the literal $\neg P(f(y))$ in the clause C_2, are unifiable. For example, if we apply the substitution $\sigma = \{f(a)/x, a/y\}$ to $\{C_1, C_2\}$, we obtain the following set of instances:

$$\{C_1\sigma = P(f(a)) \vee Q(f(a)), C_2\sigma = \neg P(f(a)) \vee R(a)\}$$

The resulting instances $C_1\sigma$ and $C_2\sigma$ *do* contain complementary literals, namely $P(f(a))$ and $\neg P(f(a))$ respectively. As a consequence, we are now able to find a resolvent of $C_1\sigma$ and $C_2\sigma$, being the clause

$$C'_3 = Q(f(a)) \vee R(a)$$

The resolution principle in first-order predicate logic makes use of the unification algorithm for constructing a most general unifier of two suitable atoms; the subsequent application of the resulting substitution to the literals containing the atoms, renders them complementary. In the preceding example, the atoms $P(x)$ and $P(f(y))$ have a most general unifier $\theta = \{f(y)/x\}$. The resolvent obtained after applying θ to C_1 and C_2, is

$$C_3 = Q(f(y)) \vee R(y)$$

The clause C'_3 from the previous example is an instance of C_3, the so-called *most general clause*: if we apply the substitution $\{a/y\}$ to C_3, we obtain the clause C'_3.

It should be noted that it is necessary to rename different variables having the same name in both parent clauses before applying resolution, since the version of the unification algorithm discussed in the previous section is not capable of distinguishing between equally named variables actually being the same variable, and equally named variables being different variables because of their occurrence in different clauses.

EXAMPLE 2.29

Consider the atoms $Q(x, y)$ and $Q(x, f(y))$ occurring in two different clauses. In this form our unification algorithm reports failure in unifying these atoms (due to the occur check). We rename the variables x and y in $Q(x, f(y))$ to u and v respectively, thus obtaining the atom $Q(u, f(v))$. Now, if we apply the unification algorithm again to compute a most general unifier of $\{Q(u, f(v)), Q(x, y)\}$, it will come up with the (correct) substitution $\sigma = \{u/x, f(v)/y\}$.

We already mentioned in Section 2.3 that the meaning of a formula is left unchanged by renaming variables. We furthermore recall that formulas only differing in the names of their (bound) variables are called variants.

From the examples presented so far, it should be clear by now that resolution in first-order predicate logic is quite similar to resolution in propositional logic: literals are cancelled out from clauses, thus generating new clauses. From now on, cancelling out a literal L from a clause C will be denoted by $C\backslash L$.

Definition 2.26 *Consider the parent clauses C_1 and C_2, respectively containing the literals L_1 and L_2. If L_1 and $\neg L_2$ have a most general unifier σ, then the clause $(C_1\sigma\backslash L_1\sigma) \vee (C_2\sigma\backslash L_2\sigma)$ is called a* binary resolvent *of C_1 and C_2. Resolution in which each resolvent is a binary resolvent, is known as* binary resolution*.*

A pair of clauses may have more than one resolvent, since they may contain more than one pair of complementary literals. Moreover, not every resolvent is necessarily a binary resolvent: there are more general ways for obtaining a resolvent. Before giving a more general definition of a resolvent, we introduce the notion of a factor.

Definition 2.27 *If two or more literals in a clause C have a most general unifier σ, then the clause $C\sigma$ is said to be a* factor *of C.*

EXAMPLE 2.30

Consider the following clause:

$$C = P(g(x), h(y)) \vee Q(z) \vee P(w, h(a))$$

The literals $P(g(x), h(y))$ and $P(w, h(a))$ in C have a most general unifier $\sigma = \{g(x)/w, a/y\}$. So,

$$C\sigma = P(g(x), h(a)) \vee Q(z) \vee P(g(x), h(a)) = P(g(x), h(a)) \vee Q(z)$$

is a factor of C. Note that one duplicate literal $P(g(x), h(a))$ has been removed from $C\sigma$.

The generalized form of resolution makes it possible to cancel out more than one literal from one or both of the parent clauses by first computing a factor of one or both of these clauses.

EXAMPLE 2.31

Consider the following set of clauses:

$$\{C_1 = P(x) \vee P(f(y)) \vee R(y), C_2 = \neg P(f(a)) \vee \neg R(g(z))\}$$

In the clause C_1 the two literals $P(x)$ and $P(f(y))$ have a most general unifier $\sigma = \{f(y)/x\}$. If we apply this substitution σ to the clause C_1, then one of these literals can be removed:

$$\begin{aligned}(P(x) \vee P(f(y)) \vee R(y))\sigma &= P(f(y)) \vee P(f(y)) \vee R(y)\\ &= P(f(y)) \vee R(y)\end{aligned}$$

The result is a factor of C_1. The literal $P(f(y))$ in $C_1\sigma$ can now be unified with the atom $P(f(a))$ in the literal $\neg P(f(a))$ occurring in C_2, using the substitution $\{a/y\}$. We obtain the resolvent

$$C_3 = R(a) \vee \neg R(g(z))$$

Note that a total of three literals has been removed from C_1 and C_2. The reader can easily verify that there are several other resolvents from the same parent clauses:

- By taking $L_1 = P(x)$ and $L_2 = \neg P(f(a))$ we get the resolvent $P(f(y)) \vee R(y) \vee \neg R(g(z))$;
- Taking $L_1 = P(f(y))$ and $L_2 = \neg P(f(a))$ results in the resolvent $P(x) \vee R(a) \vee \neg R(g(z))$;
- By taking $L_1 = R(y)$ and $L_2 = \neg R(g(z))$ we obtain $P(x) \vee P(f(g(z))) \vee \neg P(f(a))$.

We now give the generalized definition of a resolvent in which the notion of a factor is incorporated.

Definition 2.28 *A* resolvent *of the parent clauses C_1 and C_2 is one of the following binary resolvents:*

(1) A binary resolvent of C_1 and C_2;

(2) A binary resolvent of C_1 and a factor of C_2;

(3) A binary resolvent of a factor of C_1 and C_2;

(4) A binary resolvent of a factor of C_1 and a factor of C_2.

The most frequent application of resolution is refutation: the derivation of the empty clause $\square$ from a given set of clauses. The following procedure gives the general outline of this resolution algorithm.

```
procedure Resolution($S$)
    clauses ← $S$;
    while $\square \notin$ clauses do
        $\{c_i, c_j\}$ ← SelectResolvable(clauses);
        resolvent ← Resolve($c_i$, $c_j$);
        clauses ← clauses ∪ {resolvent}
    od
end
```

This algorithm is non-deterministic. The selection of parent clauses c_i and c_j can be done in many ways; how it is to be done has not been specified in the algorithm. Several different strategies have been described in the literature, each of them prescribing an unambiguous way of choosing parent clauses from the clause set. Such strategies are called the *control strategies* of resolution or *resolution strategies*. Several of these resolution strategies offer particularly efficient algorithms for making computer-based theorem proving feasible. Some well-known strategies are: *semantic resolution*, which was developed by J.R. Slagle in 1967, *hyperresolution* developed by J.A. Robinson in 1965, and various forms of *linear resolution*, such as *SLD resolution*, in the development of which R.A. Kowalski played an eminent role. At present, SLD resolution in particular is a strategy of major interest, because of its relation to the programming language PROLOG.

2.7 Resolution strategies

Most of the basic principles of resolution have been discussed in the previous section. However, one particular matter, namely the efficiency of the resolution algorithm, has not explicitly been dealt with as yet. It is needless to say that the subject of efficiency is an important one for automated reasoning.

Unfortunately, the general refutation procedure introduced in Section 2.6.3 is quite inefficient, since in many cases it will generate a large number of redundant clauses, that is, clauses not contributing to the derivation of the empty clause.

EXAMPLE 2.32

Consider the following set of clauses:

$$S = \{P, \neg P \vee Q, \neg P \vee \neg Q \vee R, \neg R\}$$

To simplify referring to them, the clauses are numbered as follows:

(1) P

(2) $\neg P \vee Q$

(3) $\neg P \vee \neg Q \vee R$

(4) $\neg R$

If we apply the resolution principle by systematically generating all resolvents, without utilizing a more specific strategy in choosing parent clauses, the following resolvents are successively added to S:

(5) Q (using 1 and 2)

(6) $\neg Q \vee R$ (using 1 and 3)

(7) $\neg P \vee R$ (using 2 and 3)

(8) $\neg P \vee \neg Q$ (using 3 and 4)

(9) R (using 1 and 7)

(10) $\neg Q$ (using 1 and 8)

(11) $\neg P \vee R$ (using 2 and 6)

(12) $\neg P$ (using 2 and 8)

(13) $\neg P \vee R$ (using 3 and 5)

(14) $\neg Q$ (using 4 and 6)

(15) $\neg P$ (using 4 and 7)

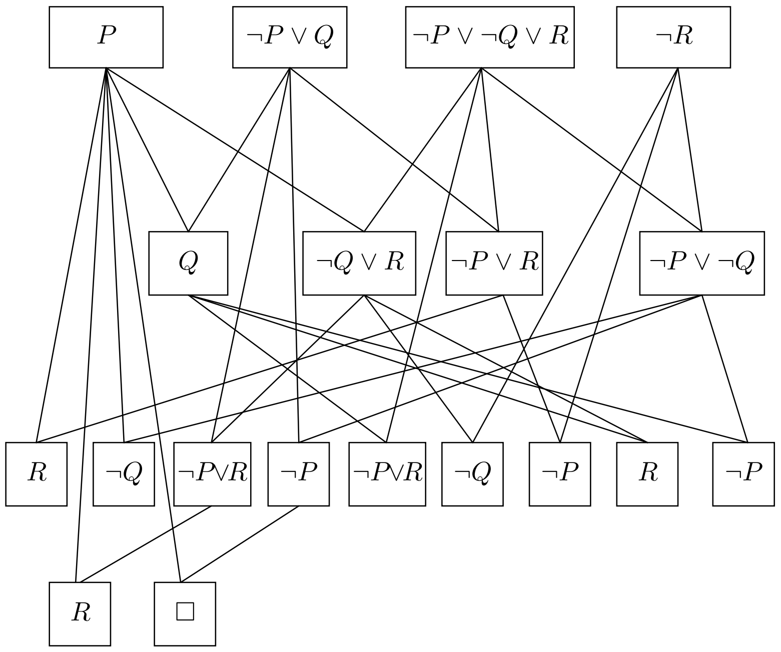


Figure 2.3: Refutation of $\{P, \neg P \vee Q, \neg P \vee \neg Q \vee R, \neg R\}$.

(16) R (using 5 and 6)

(17) $\neg P$ (using 5 and 8)

(18) R (using 1 and 11)

(19) $\Box$ (using 1 and 12)

This derivation of the empty clause $\Box$ from S has been depicted in Figure 2.3 by means of a derivation graph. As can be seen, by systematically generating all resolvents in a straightforward manner, fifteen of them were obtained, while, for instance, taking the two resolvents

(5′) $\neg P \vee R$ (using 2 and 3)

(6′) R (using 1 and 5′)

would lead directly to the derivation of the empty clause:

(7′) $\Box$ (using 4 and 6′)

In the latter refutation, significantly less resolvents were generated.

The main goal of applying a resolution strategy is to restrict the number of redundant clauses generated in the process of resolution. This improvement in efficiency is achieved by incorporating particular algorithmic refinements in the resolution principle. Some important resolution strategies will be discussed in the following two sections.

2.7.1 Semantic resolution

Semantic resolution is the name of a class of resolution strategies all having in common that the process of resolution is controlled by the declarative semantics of the clauses to be processed. We will briefly introduce the general idea and present some special forms of semantic resolution informally.

Basic ideas

Consider an unsatisfiable set of clauses S. It is possible to divide the set of clauses S into two separate subsets on the basis of a particular interpretation I: the subset S_1 contains the clauses from S which are *false* in I, and the subset S_2 contains the clauses which are *true* in I. Since S is unsatisfiable, no interpretation can ever make all clauses either *true* or *false*. So, the clause set S is split into two non-empty subsets. This semantic splitting can be used as the basis for a control strategy in which one of the parent clauses is chosen from S_1, and the other one from S_2. The generated resolvent is either added to S_1 or to S_2, dependent upon the interpretation I. In the next example, the particulars of this form of resolution are illustrated.

EXAMPLE 2.33

Consider once more the following unsatisfiable set of clauses: $S = \{P, \neg P \vee Q, \neg P \vee \neg Q \vee R, \neg R\}$. Furthermore, consider the interpretation I, defined by

$$\begin{aligned} I(P) &= \textit{false}, \\ I(Q) &= \textit{false}, \text{ and} \\ I(R) &= \textit{false}. \end{aligned}$$

Using this interpretation, we divide the set S into the following two subsets S_1 and S_2:

$$\begin{aligned} S_1 &= \{P\} \\ S_2 &= \{\neg P \vee Q, \neg P \vee \neg Q \vee R, \neg R\} \end{aligned}$$

The reader can verify that using the control strategy mentioned above, only the resolvents Q, $\neg Q \vee R$, $\neg P \vee R$, R and $\Box$ will successively be generated.

A further refinement of the described strategy can be obtained by assigning a particular *order* to the literals in the clauses. For example, in propositional logic an ordering is imposed on the propositional symbols occurring in the set of clauses. Resolution now is restricted not only by requiring that the two parent clauses are selected from the different subsets S_1 and S_2 of S (obtained from an interpretation I), but in addition, by demanding that the literal from the clause selected from S_1 to be resolved upon, is in that clause the highest one according to the ordering imposed.

Set-of-support strategy

A popular form of semantic resolution is the *set-of-support strategy*. As we mentioned in the foregoing, resolution is generally applied to prove that a specific clause G is the logical consequence of a satisfiable set of clauses S. Usually such a proof is by refutation, that is, it has the form of a derivation of the empty clause $\Box$ from $W = S \cup \{\neg G\}$. The information that the set of clauses S is satisfiable, is exploited in the set-of-support strategy to decrease the number of resolvents generated. Obviously, it is not sensible to select both parent clauses from S: since S is satisfiable, the resulting resolvent could never be the empty clause $\Box$. In the set-of-support strategy a given set of clauses W is divided into two disjoint sets: the set S being the original satisfiable set of clauses and the set T initially only containing the clauses to be proven. The set T is called the *set of support*. Now, in each resolution step at least one of the parent clauses has to be a member of the set of support. Each resulting resolvent is added to T. It is said that these clauses 'support' the clauses that were to be proven, hence the name 'set of support'. The set-of-support strategy is a powerful control strategy, which prevents the generation of many resolvents not contributing to the actual proof. The strategy is both sound and complete.

EXAMPLE 2.34

Consider the following set of clauses:

$$W = \{P, \neg P \vee Q, \neg P \vee \neg Q \vee R, \neg R\}$$

It can easily be seen that the following subset $S \subset W$ is satisfiable:

$$S = \{P, \neg P \vee Q, \neg P \vee \neg Q \vee R\}$$

(For example, choose an interpretation I such that $I(P) = I(Q) = I(R) = \mathit{true}$.) The remaining clause from W constitutes the set of support $T = \{\neg R\}$; so, $S \cup T = W$. For ease of exposition, we again number the clauses in S and T:

(1) P

(2) $\neg P \vee \neg Q \vee R$

(3) $\neg P \vee Q$

(4) $\neg R$

Resolution using the set-of-support strategy successively generates the following resolvents:

(5) $\neg P \vee \neg Q$ (using 2 and 4)

(6) $\neg Q$ (using 1 and 5)

(7) $\neg P$ (using 3 and 5)

(8) $\Box$ (using 1 and 7).

The set-of-support strategy can be described aptly in algorithmic form as follows:

```
procedure SOS-strategy($W, S$)
    $consistent \leftarrow true$;
    while $consistent$ and $S \neq \varnothing$ do
```

```
            $C \leftarrow$ SelectClause($S$);
            $S \leftarrow S \backslash \{C\}$;
            $resolvents \leftarrow$ Resolve($C, W$);
            $W \leftarrow W \cup \{C\}$;
            Process($resolvents$,$W$);
            $consistent \leftarrow \Box \notin resolvents$;
            if $consistent$ then
              $S \leftarrow S \cup resolvents$
            fi
        od;
        if not $consistent$ then
          print("Inconsistency found.")
        fi
    end
```

The *SelectClause* select a clause from the set-of-support. This clause is called the *given clause*. One possibility is to let S behave like a queue; in that case clauses are selected in the order in which they appear in S.

Note that this strategy can be considered to be a form of top-down inference: the set of support when explored using this strategy can be looked upon as a set of goals.

Hyperresolution

It is often possible to shorten the derivation obtained by binary resolution, by combining a number of derivation steps into one. An advantage of this is that the number of newly generated clauses is decreased and this has normally a favourable effect on the length of derivations. Hyperresolution is a popular inference rule that exactly does this.

Definition 2.29 *Let I be an interpretation for the set of clause S. A set of clauses $\{S_1, \ldots, S_n, N\} \subseteq S$ is called a* semantic conflict *if the following holds:*

(1) $I(S_i) = \mathit{false}$, $1 \leq i \leq n$;

(2) if $R_1 = N$, *then* $\{S_i, R_i\} \vdash_{\mathcal{B}} R_{i+1}$, $i = 1, \ldots, n$;

(3) $\mathcal{I}(R_{n+1}) = \mathit{false}$.

The clauses S_i are called satellites *(or electrons), the clause N is called the* nucleus*, and the clause R_{n+1} is known as the* hyperresolvent*. The inference rule:*

$$\frac{S_1, \ldots, S_n, N}{R_{n+1}}$$

is called the hyperresolution rule*. A derivation of C from S by means of hyperresolution is denoted by $S \vdash_{\mathcal{H}} C$.*

Different inference rules are obtained for particular choices of the interpretation I. One choice leads to the rule known as positive hyperresolution, whereas the complementary choice leads to negative hyperresolution.

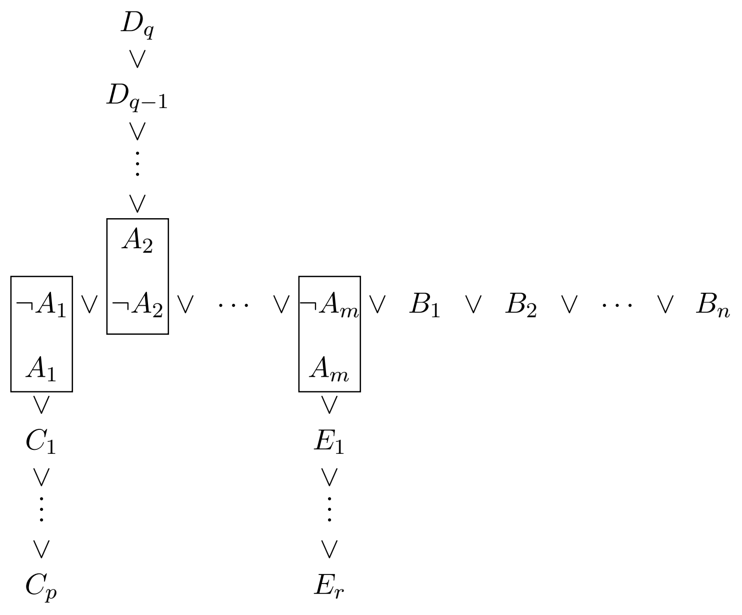

D_q
$\vee$
D_{q-1}
$\vee$
$\vdots$
$\vee$
A_2
$\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_m \vee B_1 \vee B_2 \vee \cdots \vee B_n$
A_1
A_m
C_1
C_p
E_1
E_r

Figure 2.4: Nucleus and satellites.

Definition 2.30 *Let I be an interpretation, such that for each positive literal L it holds that $I(L) = false$, then the application of hyperresolution is called* positive hyperresolution*. The satellites and the hyperresolvent are positive clauses in that case. If it holds that $I(L) = true$, then the application of hyperresolution is called* negative hyperresolution*. The satellites and the hyperresolvent are then negative clauses. A derivation by means of positive hyperresolution is denoted by $S \vdash_{\mathcal{P}} C$; similarly, for negative hyperresolution a derivation is denoted by $S \vdash_{\mathcal{N}} C$.*

Positieve hyperresolution is used as a form of bottom-up problem solving, i.e., a form of data-driven inference. The satellites can then be interpreted as data, and each newly generated hyperresolvent (a positive clause) as newly generated data. In contrast, negative hyperresolution can be seen as a form of top-down problem solving, i.e. hypothesis-driven inference.

The underlying idea of positive hyperresolution is that negative literals are deleted by the positive satellites from the nucleus, yielding a positive clause. This is illustrated schematically in Figure 2.4. Similarly, in negative hyperresolution positive literals are deleted.

EXAMPLE 2.35

Consider the following set of clauses:

$$V = \{\neg P \vee \neg Q \vee R \vee S,\ P \vee S,\ Q \vee R,\ \neg R,\ \neg S\}$$

Application of positive hyperresolution yields the following clause in one step:

$$\{\neg P \vee \neg Q \vee R \vee S,\ P \vee S,\ Q \vee R\} \vdash_{\mathcal{P}} R \vee S$$

The nucleus of positive hyperresolution is the mixed clause: $\neg P \vee \neg Q \vee R \vee S$; the satellites are $P \vee S$ and $Q \vee R$. This is the only possibility in this case. Using binary resolution would have required two derivation steps.

We obtain the following derivation:

1. $\neg P \vee \neg Q \vee R \vee S$
2. $P \vee S$
3. $Q \vee R$
4. $\neg R$
5. $\neg S$
6. $R \vee S$ (via clause 1, 2 en 3)
7. S (via clause 4 en 6)
8. $\square$ (via clause 5 en 7).

Negative hyperresolution yields the following:

$$\{\neg P \vee \neg Q \vee R \vee S,\ \neg R,\ \neg S\} \vdash_{\mathcal{N}} \neg P \vee \neg Q$$

The clause $\neg P \vee \neg Q \vee R \vee S$ is again the nucleus, but this time there are two negative (unit) clause $\neg R$ and $\neg S$ that act as satellites. The complete derivation of the empty clause using negative hyperresolution is as follows:

1. $\neg P \vee \neg Q \vee R \vee S$
2. $P \vee S$
3. $Q \vee R$
4. $\neg R$
5. $\neg S$
6. $\neg P \vee \neg Q$ (via clause 1, 4 en 5)
7. $\neg Q$ (via clause 2, 5 en 6)
8. $\square$ (via clause 3,4 en 7).

In the next example we study the use of hyperresolution using predicate logic.

EXAMPLE 2.36

Consider the following clauses:

$$\begin{aligned} S = \{&Q(a) \vee R(x),\\ &\neg Q(x) \vee R(x),\\ &\neg R(a) \vee \neg S(a),\\ &S(x)\} \end{aligned}$$

First, the clause are numbered:

1. $Q(a) \vee R(x)$
2. $\neg Q(x) \vee R(x)$
3. $\neg R(a) \vee \neg S(a)$

4. $S(x)$.

Application of positive hyperresolution yields:

5. $R(x) \vee R(a)$ (via clause 1 and 2)
6. $\Box$ (via clause 5, 3 and 4).

The following derivation is obtained by negative hyperresolution:

5. $\neg R(a)$ (via clause 3 and 4)
6. $\neg Q(a)$ (via clause 5 and 2)
7. $\Box$ (via clause 5, 6 and 1).

Note that in some cases, a hyperresolution step is exactly the same as the application of binary resolution.

Hyperresolution is not only sound but also refutation complete. However, when combined with the set-of-support strategy it is possible that refutation completeness is lost, as illustrated by means of the following example.

EXAMPLE 2.37

Consider the set of clauses $V = S \cup W$, where

$$S = \{\neg P \vee \neg R\}$$
$$W = \{P,\ R \vee \neg Q,\ Q\}$$

where S is the set-of-support. Note that W is satisfiable. Application of positive hyperresolution with the set-of-support strategy does yield no resolvents, as it is not possible to generate a positive clause from the clause in S as a nucleus and the clauses in W as satellites. Note, however, that S is unsatisfiable. Negative hyperresolution is able to find the inconsistency in this case.

2.7.2 SLD resolution: a special form of linear resolution

Linear resolution has been named after the structure of the derivation graph created by this class of strategies: in every resolution step the last generated resolvent is taken as a parent clause. The other parent clause is either a clause from the original set of clauses or a resolvent that has been generated before. A special form of linear resolution is *input resolution.* In this strategy, each resolution step, with the exception of the first one, is carried out on the last generated resolvent and a clause from the original set of clauses. The former clauses are called *goal clauses*; the latter clauses are called *input clauses*, thus explaining the name of the strategy. Input resolution is a complete strategy for Horn clauses; for the clausal form of logic in general however, input resolution is not complete.

A variant of input resolution which currently attracts a great deal of attention is *SLD resolution* for Horn clauses. In this resolution strategy, input resolution is extended with a *selection rule* which determines at every step which literal from the goal clause is selected for resolution. The remainder of this section discusses SLD resolution.

An SLD derivation is defined as follows:

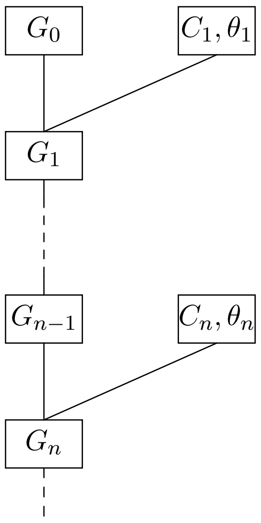


Figure 2.5: Derivation tree of SLD resolution.

Definition 2.31 *Let $\{C_i\}$ be a set of Horn clauses with*

$$C_i = B \leftarrow B_1, \ldots, B_p$$

where $p \geq 0$, and let G_0 be a goal clause of the form

$$G_0 = \ \leftarrow A_1, \ldots, A_q$$

where $q \geq 0$. An SLD derivation *is a finite or infinite sequence $G_0, G_1, \ldots$ of goal clauses, a sequence $C_1, C_2, \ldots$ of variants of input clauses and a sequence $\theta_1, \theta_2, \ldots$ of most general unifiers, such that each G_{i+1} is derived from $G_i = \ \leftarrow A_1, \ldots, A_k$ and C_{i+1} using θ_{i+1} if the following conditions hold:*

(1) A_j is the atom in the goal clause G_i chosen by the selection rule to be resolved upon, and

(2) C_{i+1} is an input clause of the form

$$C_{i+1} = B \leftarrow B_1, \ldots, B_p$$

(in which variables have been renamed, if necessary), such that $A_j\theta_{i+1} = B\theta_{i+1}$, where θ_{i+1} is a most general unifier of A_j and B.

(3) $G_i + 1$ is the clause

$$G_{i+1} = \ \leftarrow (A_1, \ldots, A_{j-1}, B_1, \ldots, B_p, A_{j+1}, \ldots, A_k)\theta_{i+1}$$

If for some $n \geq 0$, $G_n = \square$, then the derivation is called an SLD refutation *and the number n is called the* length of the refutation.

Note that a new goal clause G_{i+1} is the resolvent of the last computed resolvent G_i and (a variant of) an input clause C_{i+1}. Figure 2.5 shows the general form of a derivation tree by SLD resolution. In this figure the sequence of successive goal clauses (resolvents) $G_0, G_1, \ldots$ has been indicated.

EXAMPLE 2.38 ____

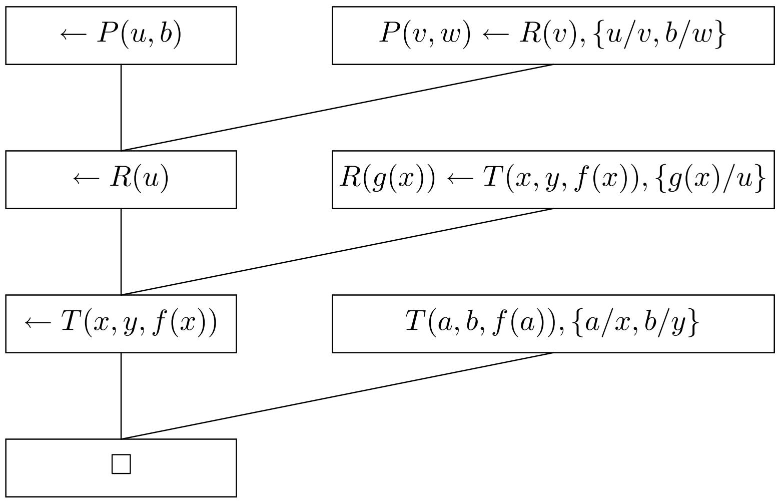

$\leftarrow P(u,b)$
$P(v,w) \leftarrow R(v), \{u/v, b/w\}$
$\leftarrow R(u)$
$R(g(x)) \leftarrow T(x,y,f(x)), \{g(x)/u\}$
$\leftarrow T(x,y,f(x))$
$T(a,b,f(a)), \{a/x, b/y\}$
$\square$

Figure 2.6: An SLD refutation.

Consider the following set of Horn clauses:

$$\{R(g(x)) \leftarrow T(x,y,f(x)), T(a,b,f(a)), P(v,w) \leftarrow R(v)\}$$

Furthermore, let the following goal clause be given:

$$\leftarrow P(u,b)$$

The clause set obtained by adding the goal clause to the original set of clauses is unsatisfiable. This can be proven using SLD resolution. Figure 2.6 depicts this proof by SLD refutation as a derivation tree.

SLD resolution is both sound and complete for Horn clauses. It furthermore is similar to the set-of-support strategy in the sense that it is also a resolution strategy controlled by a set of goals. So, SLD resolution is a form of top-down inference as well. In general it is advantageous to restrict applying the resolution principle to clauses satisfying the Horn clause format: various resolution algorithms for propositional Horn clause logic are known to have a worst-case time complexity almost linear in the number of literals. When applying some resolution strategy suitable for the clausal form of logic in general, we always have to face the danger of a combinatorial explosion. Moreover, for systems based on SLD resolution many efficient implementation techniques have been developed by now, one of which will be discussed in the next section. But there definitely are problems for which a resolution strategy applying some form of bottom-up inference turns out to be more efficient than SLD resolution.

Before introducing the notion of a search space for SLD resolution, we give another example.

EXAMPLE 2.39

Consider the following set of Horn clauses:

$$\begin{aligned} C_1 &= P(x) \leftarrow P(f(x)) \\ C_2 &= P(f(f(a))) \leftarrow \end{aligned}$$

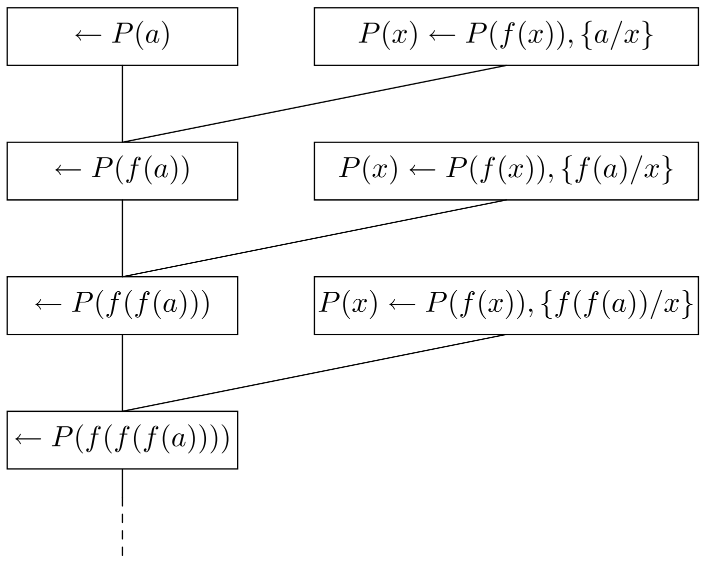

$\leftarrow P(a)$
$P(x) \leftarrow P(f(x)), \{a/x\}$
$\leftarrow P(f(a))$
$P(x) \leftarrow P(f(x)), \{f(a)/x\}$
$\leftarrow P(f(f(a)))$
$P(x) \leftarrow P(f(x)), \{f(f(a))/x\}$
$\leftarrow P(f(f(f(a))))$

Figure 2.7: Infinite derivation tree by SLD resolution.

If these clauses are 'tried' in the order in which they are specified, then for the goal clause $\leftarrow P(a)$ no refutation is found in a finite number of steps, although the resulting set of clauses obviously is unsatisfiable. The corresponding derivation tree is shown in Figure 2.7. However, if the clauses C_1 and C_2 are processed in the reverse order C_2, C_1, then a refutation will be found in finite time: the resulting refutation tree is shown in Figure 2.8.

Now let the search space for SLD resolution for a given goal on a set of clauses be a graph in which every possible SLD derivation is shown. Such a search space is often called an *SLD tree*. The branches of the tree terminating in the empty clause $\square$ are called *success branches*. Branches corresponding to infinite derivations are called *infinite branches*, and the branches representing derivations which have not been successful and cannot be pursued any further are called *failure branches*. The *level* of a vertex in an SLD tree is obtained by assigning the number 0 to the root of the tree; the level of each other vertex of the tree is obtained by incrementing the level of its parent vertex by 1.

EXAMPLE 2.40

Figure 2.9 shows the SLD tree corresponding to SLD resolution on the set of clauses from the previous example. The right branch of the tree is a success branch and corresponds to the refutation depicted in Figure 2.8; the left branch is an example of a failure branch.

It can easily be seen that a specific, fixed order in choosing parent clauses for resolution such as in the previous example, corresponds to a depth-first search in the search space. Note that such a depth-first search defines an incomplete resolution procedure, whereas a breadth-first search strategy defines a complete one. Although SLD resolution is both sound and complete for Horn clauses, in practical realizations for reasons of efficiency, variants of the algorithm are used that are neither sound nor complete. First of all, in many implementations the

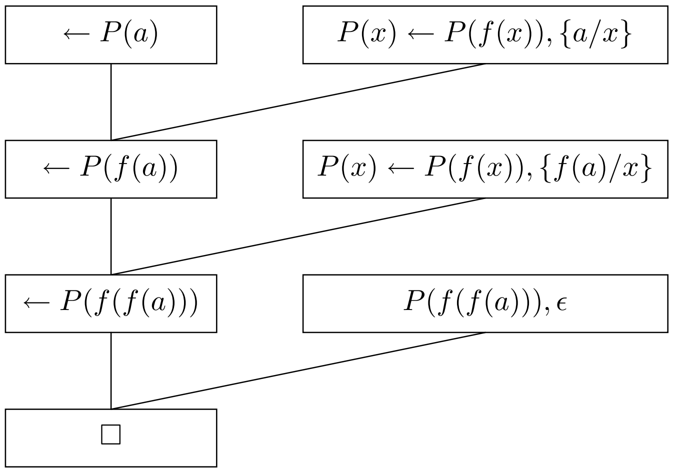


Figure 2.8: Refutation by SLD resolution.

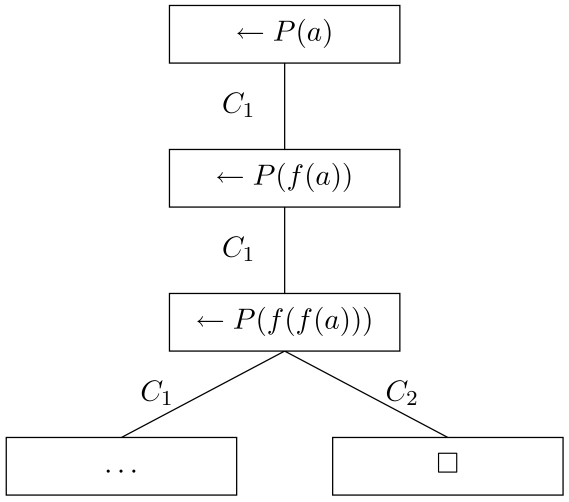


Figure 2.9: An SLD tree.

'expensive' occur check has been left out from the unification algorithm, thus destroying the soundness; the lack of the occur check might lead to circular variable bindings and yield 'resolvents' that are no logical consequences of the set of clauses. Furthermore, often the original clauses are 'tried' in some specific order, such as for example the order in which the clauses have been specified; the next input clause is only examined after the previous one has been fully explored. As a consequence, the algorithm might not be able to find a proof of a given theorem: due to an inappropriate choice of the order in which the clauses are processed, an infinite derivation tree can be created. This way, completeness of SLD resolution will be lost.

We have mentioned before that SLD resolution is of major interest because of its relation with the programming language PROLOG. In PROLOG, the control strategy employed is roughly an implementation of SLD resolution; the variant used however, is neither sound nor complete. In most (standard) PROLOG systems, the selection rule picks the leftmost atom from a goal for resolution. A depth-first strategy for searching the SLD tree is used: most PROLOG systems 'try' the clauses in the order in which they have been specified. Furthermore, in many PROLOG systems, for efficiency reasons, the occur check has been left out from the implementation.

The Horn clause subset of logic is not as expressive as the full clausal form of logic is. As is shown in the following example, this might lead to problems when translating the logical formulas into the Horn clause subset. We next show what solution PROLOG offers to this problem.

EXAMPLE 2.41

In Section 2.2 we defined the following predicates with their associated intended meaning:

Car	=	'is a car'
Fast	=	'is a fast car'
Vehicle	=	'is a vehicle'
FourWheels	=	'has four wheels'
Exception	=	'is an exception'

The formula $\forall x(Car(x) \rightarrow Vehicle(x))$ represents the knowledge that every car is a vehicle. This formula is logically equivalent to $\forall x(\neg Car(x) \vee Vehicle(x))$ and results in the following PROLOG clause:

```
vehicle(X) :- car(X).
```

The knowledge that a Bugatti is a fast car, is represented in PROLOG by a single fact:

```
fast(bugatti).
```

The implication

$$\forall x((Car(x) \wedge \neg Exception(x)) \rightarrow FourWheels(x))$$

stating that almost every car, except for instance a Bugatti, has four wheels. This formula is equivalent to

$$\forall x(\neg(Car(x) \wedge \neg Exception(x)) \vee FourWheels(x))$$

and to the formula

$$\forall x(\neg Car(x) \vee Exception(x) \vee FourWheels(x))$$

in disjunctive normal form. Unfortunately, it is not possible to translate this formula directly into PROLOG representation, since the clause contains two positive literals instead of at most one.

However, it is possible to represent the knowledge expressed by the clause in PROLOG, by means of the rather special programming trick offered by the meaning of the standard predicate `not`, which will be discussed below. The PROLOG clause we arrive at is the following:

```
fourwheels(X) :-
     car(X),
     not(exception(X)).
```

Note that in the analogous example in Section 2.2 it was necessary to specify that an alfa-romeo is not an exception to the general rule that cars have four wheels. In fact, for a correct behaviour of a proof procedure it was necessary to specify for each artery explicitly whether or not it is an exception to the rule. In most applications however, it is unreasonable to expect users to explicitly express all negative information relevant to the employed proof procedure. This problem can be handled by considering a ground literal $\neg P$ proven if an attempt to prove P using SLD resolution has not succeeded. So, in the particular case of the example, it is assumed that the goal clause `not(exception(alfa-romeo))` is proved.

The inference rule that a negative literal is assumed proven when the attempt to prove the complementary literal has failed is called *negation as failure*. Negation as failure is similar to the so-called *closed-world assumption* which is quite common in database applications. In PROLOG, an even stronger assumption, known as negation as *finite* failure, is made by taking $\neg P$ proven only if proving P using SLD resolution has failed in a finite number of steps. The PROLOG predicate `not` is the implementation of this negation as finite failure and therefore should not be taken as the ordinary negation: it is an extra-logical feature of PROLOG.

2.8 Applying logic for building intelligent systems

In the preceding sections, much space has been devoted to the many technical details of knowledge representation and automated reasoning using logic. In the present section, we shall indicate how logic can actually be used for building a logic-based intelligent system. As logic offers suitable basis for model-based systems, more about the use of logic in the context of intelligent systems will said in Chapter 6.

In the foregoing, we have seen that propositional logic offers rather limited expressiveness, which in fact is too limited for most real-life applications. First-order predicate logic offers much more expressive power, but that alone does not yet render the formalism suitable for building intelligent systems. There are some problems: any automated reasoning method for full first-order logic is doomed to have a worst-case time complexity at least as bad as that of checking satisfiability in propositional logic, which is known to be NP-complete (this means that no one has been able to come up with a better deterministic algorithm than an exponentially time-bounded one, although it has not been proven that better ones do not exist). Furthermore, we know that first-order predicate logic is undecidable; so, it is not even sure that an algorithm for checking satisfiability will actually terminate. Fortunately, the circumstances are not always as bad as that. A worst-case characterization seldom gives a realistic indication of the time an algorithm generally will spend on solving an arbitrary problem. Moreover, several suitable syntactic restrictions on first-order formulas have been formulated from which a substantial improvement of the time complexity of the algorithm is obtained; the Horn clause format we have paid attention to is one such restriction.

Since syntactic restrictions are only acceptable as far as permitted by a problem domain, we consider some examples, such as the logical circuit introduced in Chapter 1. However, first we discuss special standard predicates that are often used in modelling practical applications.

2.8.1 Reasoning with equality and ordering predicates

The special binary predicate symbols $>$ (*ordering predicate*) and $=$ *equality predicate* are normally specified in infix position, since this is normal mathematical practice. Both equality and the ordering predicates have a special meaning, which is described by means of a collection of axioms. The meaning of the equality predicate is defined by means of the following four axioms:

E_1 (*reflexivity*): $\forall x(x = x)$

E_2 (*symmetry*): $\forall x \forall y(x = y \rightarrow y = x)$

E_3 (*transitivity*): $\forall x \forall y \forall z(x = y \wedge y = z \rightarrow x = z)$

E_4 (*substitutivity*): $\forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n((x_1 = y_1 \wedge \ldots \wedge x_n = y_n) \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n))$, and $\forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n((x_1 = y_1 \wedge \ldots \wedge x_n = y_n \wedge P(x_1, \ldots, x_n)) \rightarrow P(y_1, \ldots, y_n))$

Axiom E_1 states that each term in the domain of discourse is equal to itself; axiom E_2 expresses that the order of the arguments of the equality predicate is irrelevant. Axiom E_3 furthermore states that two terms which are equal to some common term, are equal to each other. Note that axiom E_2 follows from the axioms E_1 and E_3; nevertheless, it is usually mentioned explicitly. The three axioms E_1, E_2 and E_3 together imply that equality is an *equivalence relation*. Addition of axiom E_4 renders it a *congruence relation*. The first part of axiom E_4 states that equality is preserved under the application of a function; the second part expresses that equal terms may be substituted for each other in formulas.

EXAMPLE 2.42

Consider the following set of clauses S:

$$S = \{\neg P(f(x), y) \vee Q(x, x), P(f(a), a), a = b\}$$

Suppose that, in addition, we have the equality axioms. If we add the clause $\neg Q(b, b)$ to S, the resulting set of clauses will be unsatisfiable. This can easily be seen informally as follows: we have $P(f(a), a) \equiv P(f(b), a)$ using the given clause $a = b$ and the equality axiom E_4. Now, we replace the atom $P(f(a), a)$ by the equivalent atom $P(f(b), a)$ and apply binary resolution.

The explicit addition of the equality axioms to the other formulas in a knowledge base suffices for rendering equality available for use in an intelligent system. However, it is well known that proving theorems in the presence of the equality axioms can be very inefficient, since many redundant clauses may be generated using resolution. Again, several refinements of the (extended) resolution principle have been developed to overcome the inefficiency problem. For dealing with equality, the resolution principle has for example been extended with an extra inference rule: *paramodulation*. Informally speaking, the principle of paramodulation is the following: if clause C contains a term t and if we have a clause $t = s$, then derive a clause by substituting s for a single occurrence of t in C. Therefore, in practical realizations equality is often only present implicitly in the knowledge base, that is, it is used as a 'built-in' predicate.

Another, more restrictive way to deal with equality is *demodulation*, which adds directionality to equality, meaning that one side of the equality may be replaces (rewitten) by the other side, but not the other way around.

Definition 2.32 *A* demodulator *is a positive unit clause with equality predicate of the form $(l = r)$, where l and r are terms. Let $C \vee L^t$ a clause; where L^t indicates that the literal L contains the term t. Let σ be a substitution such that $l\sigma = t$, then* demodulatie *is defined as the inference rule:*

$$\frac{C \vee L^t, (l = r)}{C \vee L^{t \to r\sigma}}$$

where $L^{t \to r\sigma}$ indicates that term t is rewritten to $r\sigma$.

Thus a demodulator $(l = r)$ can be interpreted as a *rewrite rule* $l \to r$ with a particular orientation (here from left to right).

EXAMPLE 2.43 ---

Consider the demodulator

$$(\mathit{brother}(\mathit{father}(x)) = \mathit{uncle}(x))$$

and the clause

$$(\mathit{age}(\mathit{brother}(\mathit{father}(\mathit{John}))) = 55)$$

Application of the demodulator yields

$$(\mathit{age}(\mathit{uncle}(\mathit{John})) = 55)$$

The applied substitution was $\sigma = \{\mathit{Jan}/x\}$.

In many real-life applications, a universally quantified variable ranges over a finite domain $D = \{c_i \mid i = 1, \ldots, n, n \geq 0\}$. The following property usually is satisfied: $\forall x(x = c_1 \vee x = c_2 \vee \ldots \vee x = c_n)$, with $c_i \neq c_j$ if $i \neq j$. This property is known as the *unique names assumption*; from this assumption we have that objects with different names are different.

EXAMPLE 2.44

Consider the following set of clauses S:

$$S = \{\neg P(x) \vee x = a\}$$

We suppose that the equality axioms as well as the unique name assumption hold. Now, if we add the clause $P(b)$ to S, we obtain an inconsistency, since the derivable clause $b = a$ contradicts with the unique name assumption.

The ordering predicates $<$ and $>$ define a *total order* on the set of real numbers. They express the usual, mathematical 'less than' and 'greater than' binary relations between real numbers. Their meaning is defined by means of the following axioms:

O_1 (*irreflexivity*): $\forall x \neg(x < x))$

O_2 (*antisymmetry*): $\forall x \forall y(x < y \rightarrow \neg(y < x))$

O_3 (*transitivity*): $\forall x \forall y \forall z((x < y \wedge y < z) \rightarrow x < z)$

O_4 (*trichonomy law*): $\forall x \forall y((x < y \vee x = y \vee x > y)$

Axiom O_1 states that no term is less that itself; axiom O_2 expresses that reversing the order of the arguments of the predicate $<$ reverses the meaning. Axiom O_3 furthermore states that if a term is less than some other term, and this term is less than a third term, then the first term is less than the third one as well. Note that axiom O_2 follows from O_1 and O_3. The axioms O_1, O_2 and O_3 concern the ordering predicate $<$. The axioms for the ordering predicate $>$ are similar to these: we may just substitute $>$ for $<$ to obtain them. Axiom O_4 states that a given term is either less than, equal to or greater than another given term. Again, in practical realizations, these axioms usually are not added explicitly to the knowledge base, but are assumed to be present implicitly as 'built-in' predicates or as evaluable predicates, that after it has been checked that all variables are instantiated to constants, are evaluated as an expression, returning true or false.

2.8.2 Reasoning models

In this section, a number of examples will be discussed in order to give the reader a feeling how real-world models look like. Of course, given the space available, the examples discussed are still toy examples, but nevertheless inspired by the realistic problems.

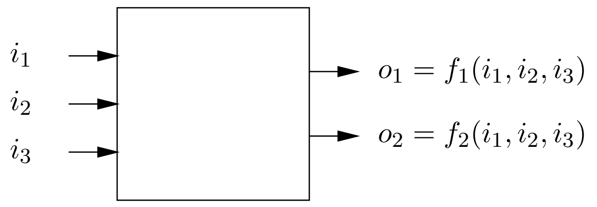


Figure 2.10: Black-box logical circuit.

Functional models

In Chapter 1 we have discussed logical circuits as examples of deep knowledge. Here, we study a logical formalisation of this circuit and dicuss ways in which a specification of a logical circuit can be used for simulation purposes.

A logical circuit can be considered a black-box with *inputs* $i_1, \ldots, i_n$ and *outputs* $o_1, \ldots, o_m$, such that $o_k = f_k(i_1, \ldots, i_n)$, as shown in Figure 2.10. For simplicities sake it is assumed that i_j and o_k are Boolian variables which can take values *false* and *true*, normally in this context of Boolean logic represented as 0 and 1. Hence, the functions f_k are Boolean functions of n Boolean variables. The following components make up the internals of a circuit:

- *gates*, and
- *wires*, connecting the gates.

Here, the following gates with two inputs and one output are distinguished:

- the AND-gate,
- the OR-gate,
- the XOR-gate ('exclusive' OR).

In addition, we have the NOT-gate (or 'inverter') which has one input and one output. Logical gates are drawn using pictograms as shown in Figure 2.11.

Let $\mathcal{L} = (C, A, B, I, O)$ be a *logical circuit*, where C denotes a set of components, A their properties, and B represents the wires; I and O are sets of inputs and outputs, respectively. If $x \in C$ is a components, then, for example, $Andg(x)$ indices an AND-gate.

Input and outputs are specified as follows:

- $i(k, x)$ is the kth input of gate x;
- $o(k, x)$ is the kth output of gate x.

The wires B are represented by the predicate symbol

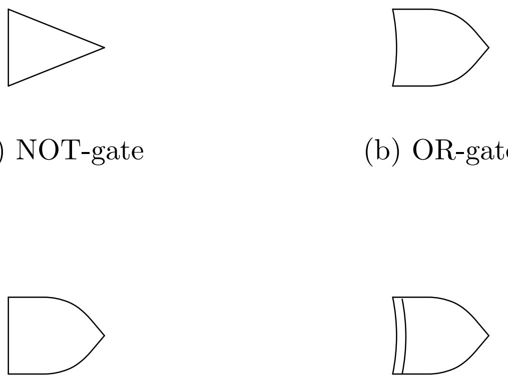
$$Connects(x, y)$$

indicating that gates x and y are connected by a wire. Values of input or output signals are specified as follows

$$v(k) = b$$

indicating that input or output k transmits a signal with value b. For example, $v(i(1, a)) = 1$ indicates that the first input of component a is equal to 1.

The properties of the gates can now be specified as follows:



(a) NOT-gate (b) OR-gate

(c) AND-gate (d) XOR-gate

Figure 2.11: Pictograms of gates.

$$\begin{aligned}
&\forall x((Andg(x) \land \\
&\quad (v(i(1,x)) = 1) \land \\
&\quad (v(i(2,x)) = 1)) \\
&\quad \to \\
&\quad (v(o(1,x)) = 1))
\end{aligned}$$

$$\begin{aligned}
&\forall x \forall n((Andg(x) \land \\
&\qquad (v(i(n,x)) = 0)) \\
&\qquad \to \\
&\qquad (v(o(1,x)) = 0))
\end{aligned}$$

$$\begin{aligned}
&\forall x \forall n(Org(x) \land \\
&\qquad (v(i(n,x)) = 1)) \\
&\qquad \to \\
&\qquad (v(o(1,x)) = 1))
\end{aligned}$$

$$\begin{aligned}
&\forall x((Org(x) \land \\
&\quad (v(i(1,x)) = 0) \land \\
&\quad (v(i(2,x)) = 0)) \\
&\quad \to \\
&\quad (v(o(1,x)) = 0))
\end{aligned}$$

$$\begin{aligned}
&\forall x \forall z((Xorg(x) \land \\
&\qquad (v(i(1,x)) = z) \land \\
&\qquad (v(i(2,x)) = z)) \\
&\qquad \to \\
&\qquad (v(o(1,x)) = 0))
\end{aligned}$$

$$\begin{aligned}
&\forall x \forall y \forall z((Xorg(x) \land \\
&\qquad\quad (v(i(1,x)) = y) \land \\
&\qquad\quad (v(i(2,x)) = z) \land \\
&\qquad\quad (y \neq z))
\end{aligned}$$

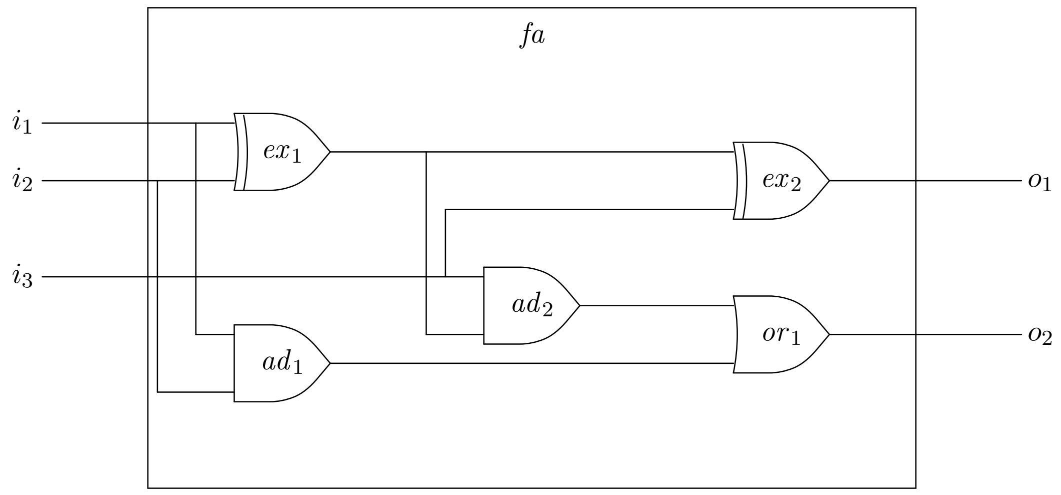


Figure 2.12: The full adder.

$$\begin{array}{l} \rightarrow \\ (v(o(1,x)) = 1)) \end{array}$$

The binary predicate symbol *Connects* is used to specify properties of wires, and these properties are also part of the set A:

$$\begin{array}{l} \forall x \forall y \forall z((\mathit{Connects}(x,y) \wedge \\ \quad (v(x) = z)) \\ \quad \rightarrow \\ \quad (v(y) = z)) \end{array}$$

Now, let us again consider the full-adder circuit already introduced in Chapter 1.

EXAMPLE 2.45

The full-adder shown in Figure 2.12 consists of the following gates:

$\mathit{Adder}(\mathit{fa})$
$\mathit{Xorg}(ex_1)$
$\mathit{Xorg}(ex_2)$
$\mathit{Andg}(ad_1)$
$\mathit{Andg}(ad_2)$
$\mathit{Org}(or_1)$

where *Adder*(*fa*); represents the entire full-adder, i.e. the back-box as shown in Figure 2.10. The three inputs to the circuit are specified as follows:

$v(i(1, fa)) = 1$
$v(i(2, fa)) = 0$
$v(i(3, fa)) = 1$

i.e. we have that $i_1 = 1$, $i_2 = 0$ and $i_3 = 1$. Similarly, the outputs o_1 and o_2 are represented as follows:

$v(o(1, fa)) = 0$
$v(o(2, fa)) = 1$

Hence, it holds that $o_1 = 0$ and $o_2 = 1$. Finally, we need to specify the wires connecting the gates:

$Connects(i(1, fa), i(1, ex_1))$
$Connects(i(2, fa), i(2, ex_1))$
$Connects(i(1, fa), i(1, ad_1))$
$Connects(i(2, fa), i(2, ad_1))$
$Connects(i(3, fa), i(2, ex_2))$
$Connects(i(3, fa), i(1, ad_2))$
$Connects(o(1, ex_1), i(1, ex_2))$
$Connects(o(1, ex_1), i(2, ad_2))$
$Connects(o(1, ad_2), i(1, or_1))$
$Connects(o(1, ad_1), i(2, or_1))$
$Connects(o(1, ex_2), o(1, fa))$
$Connects(o(1, or_1), o(2, fa))$

Note that we have not indicated that the *Connects* predicate is symmetric, i.e. signals go in one direction only.

An obvious application of a logical circuit is *simulation*. As this would amount to transforming input into output, positive hyperresolution, i.e. a form of bottom-up inference, seems most suitable. We thus obtain the following specification of the problem we need to solve:

$$C \cup B \cup A \cup \{v(i(k, j)) = b \mid \text{for certain } k, j\} \cup \{\neg v(o(x, y)) = z\} \vdash_{\mathcal{P}} \Box$$

Note that $\neg v(o(x, y)) = z$ has to be included in order make sure that the empty clause $\Box$ is derived.

Spatial and anatomical reasoning

In certain fields of medicine, for example neurology, knowledge concerning the anatomy of the human body is of major importance. The form of automated reasoning in which knowledge concerning the anatomy of the human body is applied, is known as *anatomical reasoning* or, more in general, as *spatial reasoning*.

The point of departure for any intelligent system implementing anatomical reasoning, is the axiomatization of the basic anatomical relations. The more precise the description of the anatomical structures must be, the more complex the resulting axiomatization will be. Not always is a precise three-dimensional specification of anatomical relations required. In our approach to anatomical reasoning, it suffices to indicate only that certain anatomical structures are connected to each other in a qualitative way, as axiomatized by the *Connected* predicate. This predicate is defined as a transitive, irreflexive relation, as follows:

$\forall x \forall y \forall z (Connected(x, y) \land Connected(y, z) \to Connected(x, z))$
$\forall x (\neg Connected(x, x))$

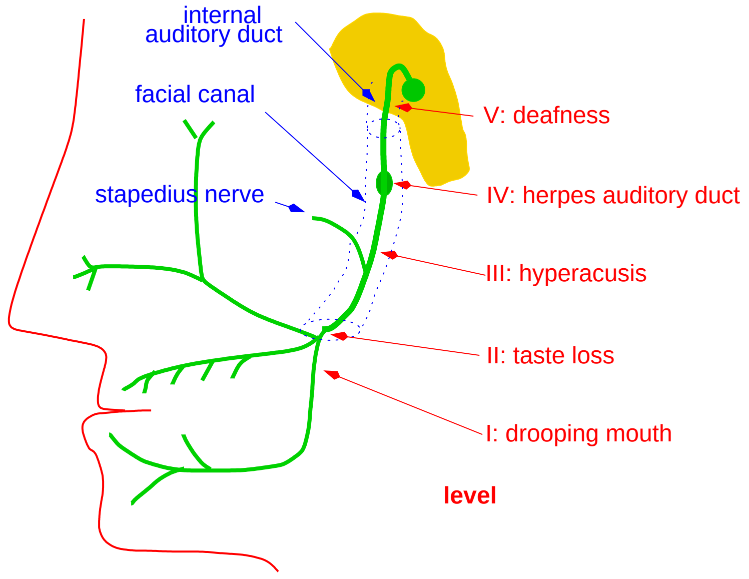


Figure 2.13: Levels in the facial nerve.

Note that the *Connected* predicate is by its transitive and irreflexive properties also antisymmetric. (So, a theorem prover is capable of detecting an inconsistency given the formulas $Connected(a,b)$ and $Connected(b,a)$ from the two axioms given above.)

As a starting point for our discussion concerning the specification of anatomical reasoning in logic, we consider an actual example (the diagnosis of lesions of the facial nerve) taken from a textbook of neurology.

The classical picture of *facial palsy* is well known. These patients have a mouth that droops and may draw to the opposite side. They cannot wrinkle the forehead or close the eye at the affected side. Facial palsy is due to a lesion of the facial nerve (cranial nerve VII); this nerve can be affected by a large variety of disorders. The severity and nature of the complaints and signs that may be observed in the patient depend on the level of facial nerve lesion. Knowledge of the branching pattern of the nerve and the consequences of a lesion of a particular branch is important in diagnostic problem solving. Figure 2.13 gives a schematic overview of the branching pattern of this nerve. The facial nerve is a mixed nerve; it contains motor fibers that supply striated muscle fibers, sensory fibers that carry taste from the anterior two-third of the tongue and some sensation, and parasympathetic fibers.

The facial nerve emerges from the brain stem and leaves the skull via the internal auditory meatus. Next, a small nerve is branched off (the stapedius nerve), that supplies the stapedius muscle (a small muscle that is attached to the ear drum, regulating its tension). The facial nerve proceeds its way through the facial canal (in the temporal bone), next branching off the chorda tympani, a nerve mostly consisting of parasympathetic fibers that supply the submandibular and sublingual glands. The facial nerve leaves the facial canal through the stylomastoid foramen. Finally, it splits up in a number of branches that supply the superficial musculature of the face and scalp (e.g. orbicularis oris et oculi, buccinator, platysma).

A bit simplified, we distinguish the following five levels of facial nerve lesions (consult again Figure 2.13):

Level 1: A lesion outside the stylomastoid foramen produces signs such as drooping of the mouth. The patient cannot whistle, wink, close the eye or wrinkle the forehead. When the patient attempts to close the eye, the eye bulb will turn upward (*Bell's sign*).

Level 2: A lesion of the nerve in its course through the facial canal will result in all the signs as present in a level 1 lesion, but in addition there is reduced salivation (lesion of the chorda tympani) and loss of taste in the anterior two-thirds of the tongue.

Level 3: All signs of a level 2 lesion are present, but in addition the stapedius nerve is affected, causing hyperacusis (due to paralysis of the stapedius muscle).

Level 4: A lesion of the geniculate ganglion is usually due to herpes zoster, in which case herpetic lesions are visible on the ear drum and external auditory canal (*Ramsay Hunt syndrome*). Typically, a patient will experience pain in and behind the ear.

Level 5: A lesion in the internal auditory meatus is usually associated with acoustic nerve (cranial nerve VIII) involvement, because the last nerve also runs through this canal. In addition to the signs mentioned for lower level lesions, deafness will be present.

This completes the description of the neurological knowledge involved in diagnosing facial nerve lesions.

In formalizing this knowledge using first-order predicate logic, we start by completing the axiomatization of the anatomical relationships by giving a domain-specific fill-in for the *Connected* predicate. The atom $Connected(x, y)$ is intended to mean that the facial nerve runs from level x up to level y:

$$\begin{array}{l} Connected(stylomastoid\_foramen, chorda\_tympani) \\ Connected(chorda\_tympani, stapedius\_nerve) \\ Connected(stapedius\_nerve, geniculate\_ganglion) \\ Connected(geniculate\_ganglion, internal\_auditory\_meatus) \end{array}$$

Note that we have employed anatomical terms to denote the various levels.

To relate anatomical structures and signs that may arise due to facial nerve lesion, we have to express that the signs associated with a lesion at a certain level x includes all the signs of a lesion at a lower level y:

$$\forall x \forall y (Lesion(x) \wedge Connected(y, x) \rightarrow Lesion(y))$$

This completes our axiomatization of the knowledge that forms the basis of logical anatomical reasoning.

We next specify the relationship between a lesion at a certain level and the specific anatomical structures that will be affected by this lesion, expressed by the unary predicate symbol *Affected*. We use a bi-implication, because given a lesion at a certain level we may want to know which structures will be affected by this lesion; on the other hand, the observation of malfunction of certain structures may be interpreted as evidence for a lesion at a certain level (in Chapter 6 will see that it is more natural to use abduction for diagnostic reasoning using causal knowledge):

$$\begin{array}{l} (Lesion(stylomastoid\_foramen) \leftrightarrow \\ \quad (Affected(orbicularis\_oris) \wedge \\ \quad\quad Affected(orbicularis\_oculi) \wedge \end{array}$$

$$\begin{array}{l} \quad Affected(buccinator) \wedge \\ \quad Affected(frontalis\_muscle) \wedge \\ \quad Affected(platysma))) \end{array}$$

$$\begin{array}{l} (Lesion(chorda\_tympani) \leftrightarrow \\ \;(Affected(sensory\_taste\_fibers) \wedge \\ \quad Affected(sublingual\_gland) \wedge \\ \quad Affected(submaxillary\_gland))) \end{array}$$

$$(Lesion(stapedius\_nerve) \leftrightarrow Affected(stapedius\_muscle))$$

$$(Lesion(geniculate\_ganglion) \leftrightarrow Affected(sensory\_fibers\_ear))$$

$$(Lesion(internal\_auditory\_meatus) \leftrightarrow Affected(acoustic\_nerve))$$

Finally, paralysis of certain muscles and disturbed sensation will give rise to specific signs and complaints in the patient. This knowledge is again expressed using a collection of bi-implications:

$$\begin{array}{l} (Affected(orbicularis\_oris) \leftrightarrow (Sign(mouth\_droops) \wedge \\ \;Sign(cannot\_whistle))) \end{array}$$

$$(Affected(orbicularis\_oculi) \leftrightarrow (Sign(cannot\_close\_eyes) \wedge Sign(Bell)))$$

$$(Affected(buccinator) \leftrightarrow Sign(flaccid\_cheeks))$$

$$(Affected(frontalis\_muscle) \leftrightarrow Sign(cannot\_wrinkle\_forehead))$$

$$(Affected(platysma) \leftrightarrow Sign(paresis\_superficial\_neck\_musculature))$$

$$\begin{array}{l} ((Affected(sublingual\_gland) \wedge Affected(submaxillary\_gland)) \leftrightarrow \\ \;Complaint(dry\_mouth)) \end{array}$$

$$(Affected(sensory\_taste\_fibers) \leftrightarrow Complaint(taste\_loss\_anterior\_part\_tongue))$$

$$(Affected(stapedius\_muscle) \leftrightarrow Complaint(hyperacusis))$$

$$\begin{array}{l} (Affected(sensory\_fibers\_ear) \leftrightarrow \\ \;(Complaint(pain\_behind\_ear) \wedge \\ \quad Complaint(pain\_within\_ear) \wedge \\ \quad Sign(herpetic\_lesions))) \end{array}$$

$$(Affected(acoustic\_nerve) \leftrightarrow Complaint(deafness))$$

Let T be the logical theory given above. For example, after automatic translation to clausal form, using hyperresolution we are now capable to derive:

$$T \cup \{Lesion(stapedius\_nerve)\} \cup \{\neg Sign(x)\} \cup \{\neg Complaint(y)\} \vdash_{\mathcal{P}} \Box$$

where for x we have *mouth_droops*, *cannot_whistle*, *cannot_close_eyes*, *Bell*, *flacid_cheeks*, *cannot_wrinkle_forehead* and *paresis_superficial_neck_musculature*; for y we have *hyperacusis*, *dry_mouth* and *taste_loss_anterior_part_tongue*. Note that all results but the complaint *hyperacusis* have been derived using the anatomical axioms for the *Connected* predicate. Reasoning from signs and complaints to the level of a facial nerve lesion is also possible (essentially employing the bi-implication), but here we need a meta-level primitive that selects from the unit clause concerning *Lesion* the one specifying knowledge regarding the highest level of the lesion.

In our formalization of the anatomical reasoning model, we did not make a distinction between left- and right-sided lesions of the facial nerve. The extension of the logical theory to include this distinction is straightforward.

Causal reasoning

Although in the previous section we primarily focussed on the logical specification of anatomical relations, implicitly other reasoning models were also involved. Some of the formulas presented in the previous section express some relationship between cause and effect of nerve damage, thus representing *causal knowledge*. The reasoning about such cause–effect relationships is known as *causal reasoning*. In the present section, we shall study the logic of causal reasoning in medicine, taking the logical specification of physiological processes as a point of departure.

The representation of causal knowledge in logic is rather straightforward; it may be represented by means of a collection of logical implications of the following form:

$$\mathit{cause} \rightarrow \mathit{effect}$$

where both *cause* and *effect* are conjunctions of literals. Most literals refer to the *state* of some *parameter*; the states of all parameters together describe the entire physiological process. As an example of a parameter consider the level of a substance in the blood; the actual level of the substance stands for the parameter's state.

A state is either *numeric* or *qualitative*. An example of a numeric state (of parameter level of *sodium* in the *blood*) is expressed by the following unit clause:

$$\mathit{conc}(\mathit{blood}, \mathit{sodium}) = 125$$

In clinical practice, numeric parameters are often changed to qualitative states. In the above case, we get:

$$\mathit{conc}(\mathit{blood}, \mathit{sodium}) = \mathit{decreased}$$

There are several common types of causal reasoning in medicine. We shall study the *negative feedback process* and its logical specification in some detail.

In terms of cause–effect relationships, the global specification of a negative feedback process leads to the following logical theory T (to simplify matters, we have assumed that a cause consists of a single literal):

$$\begin{array}{l} s \rightarrow r_1 \\ r'_1 \rightarrow r_2 \\ \quad\vdots \\ r'_{n-1} \rightarrow r_n \\ r'_n \rightarrow \neg s \end{array}$$

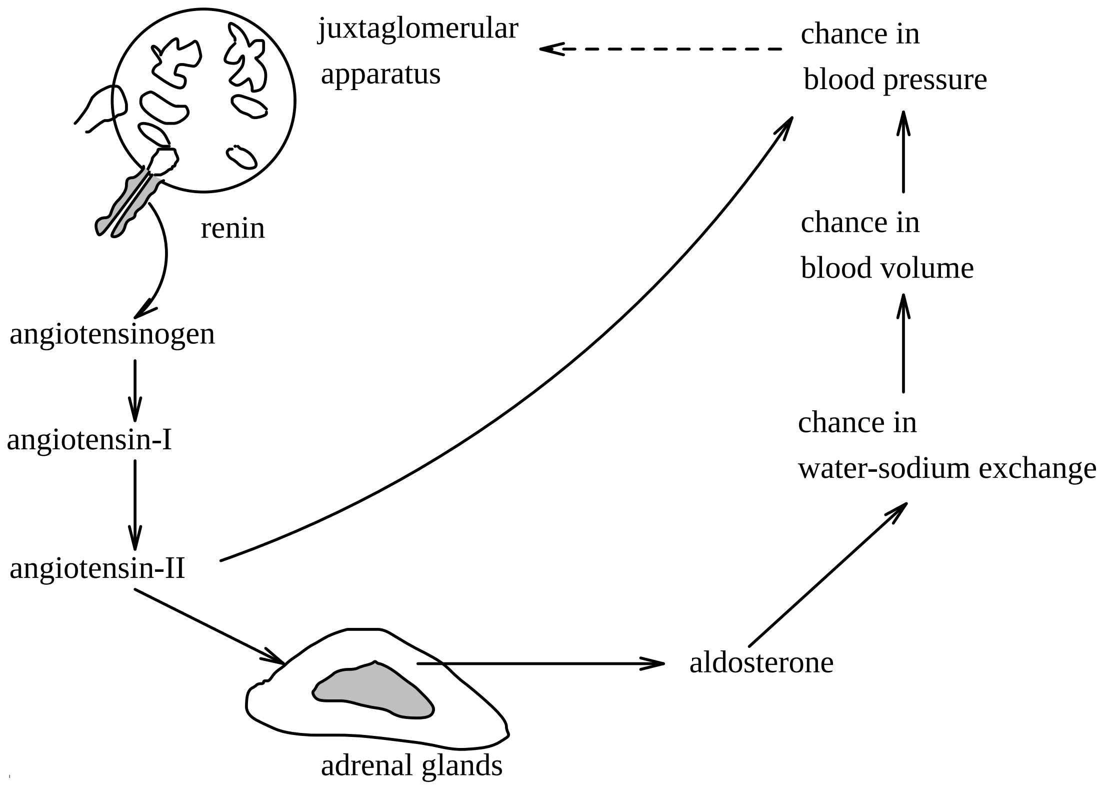


Figure 2.14: The renin–angiotensin–aldosterone system.

where s, r_i, r'_i, $1 \leq i \leq n$, $n \geq 1$ are literals in first-order logic; the literals r_i, r'_i are similar, in the sense that substitution of terms for variables occurring in these literals can make them syntactically equal. Note that we have $T \models \neg s$; in words: the negative feedback is a semantic consequence of the process description.

To investigate the applicability of this approach to formalizing causal reasoning in medicine, we have chosen a particular example of a negative feedback process from the literature, viz. the renin–angiotensin–aldosterone system. We start by giving a brief summary of the medical knowledge involved.

The regulation of the blood pressure is accomplished by the collaborative effort of a number of control systems in the human body. One of these control systems is the *renin–angiotensin–aldosterone system*. Figure 2.14 gives a pictorial overview of this control system. We review the regulatory factors involved.

The proteolytic enzyme *renin* is released by cells of the *juxtaglomerular apparatus* in the kidneys. A decrease in the mean renal arterial pressure increases the renin secretion, and an increase in the mean renal arterial pressure leads to a decrease in the renin secretion. Renin acts on an α_2-globulin (*angiotensinogen*) that circulates in the blood, liberating the decapeptid *angiotensin-I*. In turn, the octapeptid *angiotensin-II* is liberated from *angiotensin-I* by *angiotensin converting enzyme* (ACE) that is produced in the lungs.

Angiotensin-II is a powerful arteriolar constrictor; administration of angiotensin-II leads to an increase in arterial blood pressure. In addition, it stimulates the secretion of *aldosterone*, a hormone produced by the adrenal cortex; an increase in angiotensin-II levels in the blood increases aldosterone levels in the blood. Aldosterone stimulates active reabsorption of Na^+

from the urine and the secretion of K^+ to the urine. Water moves with the reabsorbed Na^+ to the blood, which causes an increase in blood volume. This in turn leads to an increase in blood pressure. Finally, an increase in blood pressure inhibits the secretion of renin. This completes our description.

When considering the physiological process described above in terms of a causal model, we have to analyse its behaviour in terms of causes and effects. We start this formalization by introducing a number of predicate and function symbols and constants that will be used to represent parameters and states. For the representation of the level of a substance in the blood we employ the binary function symbol *conc*. The unary function symbol *pressure* stands for blood pressure. The conversion of one substance into another substance by some enzyme, will be represented by the binary function symbol *conversion*. Finally, we distinguish two constants: *decreased* and *increased* to express the states of various parameters in a qualitative way. Step by step, the text given above will now be translated into a logical theory T.

A decrease in the blood pressure yields an increase in renin blood levels:

$$\mathit{pressure}(\mathit{blood}) = \mathit{decreased} \rightarrow \mathit{conc}(\mathit{blood}, \mathit{renin}) = \mathit{increased}$$

The relationship between renin levels in the blood and conversion of angiotensinogen into angiotensin-I is expressed as follows:

$$\forall v(\mathit{conc}(\mathit{blood}, \mathit{renin}) = v \rightarrow \mathit{conversion}(\mathit{angiotensinogen}, \mathit{angiotensin\_I}) = v)$$

where the universally quantified variable v stands for *increased* or *decreased*.

The relationship between decreased or increased conversion of angiotensinogen into angiotensin-I is represented by means of the following logical implication:

$$\begin{aligned}&\forall v(\mathit{conversion}(\mathit{angiotensinogen}, \mathit{angiotensin\_I}) = v \rightarrow \\ &\quad \mathit{conc}(\mathit{blood}, \mathit{angiotensin\_I}) = v)\end{aligned}$$

A change v in the blood level of angiotensin-I leads to an inverse change in the ACE levels, and a similar change in angiotensin-II levels:

$$\begin{aligned}&\forall v(\mathit{conc}(\mathit{blood}, \mathit{angiotensin\_I}) = v \rightarrow \\ &\quad (\neg(\mathit{conc}(\mathit{blood}, \mathrm{ACE}) = v) \wedge \mathit{conversion}(\mathit{angiotensin\_I}, \mathit{angiotensin\_II}) = v))\end{aligned}$$

$$\begin{aligned}&\forall v(\mathit{conversion}(\mathit{angiotensin\_I}, \mathit{angiotensin\_II}) = v \rightarrow \\ &\quad \mathit{conc}(\mathit{blood}, \mathit{angiotensin\_II}) = v)\end{aligned}$$

Angiotensin-II produces arterial vasoconstriction and an increase in aldosterone levels:

$$\begin{aligned}&\mathit{conc}(\mathit{blood}, \mathit{angiotensin\_II}) = \mathit{increased} \\ &\rightarrow \\ &(\mathit{Vasoconstriction}(\mathit{arteries}, \mathit{peripheral}) \wedge \\ &\ \mathit{conc}(\mathit{blood}, \mathit{aldosterone}) = \mathit{increased})\end{aligned}$$

Arterial vasoconstriction produces an increase in bloodpressure:

$$\mathit{Vasoconstriction}(\mathit{arteries}, \mathit{peripheral}) \rightarrow \mathit{pressure}(\mathit{blood}) = \mathit{increased}$$

An increase in the aldosterone levels results in an increase of blood sodium and a decrease of the potassium levels:

$$conc(blood, aldosterone) = increased \rightarrow$$
$$(conc(blood, sodium) = increased \wedge conc(blood, potassium) = decreased)$$

The reabsorption of sodium is accompanied by the reabsorption of water, causing an increase in blood volume; more in general, a change in sodium level causes a change in blood volume:

$$\forall v\ (conc(blood, sodium) = v \rightarrow volume(blood) = v)$$

The change in blood volume causes a similar change in cardiac output:

$$\forall v(volume(blood) = v \rightarrow output(heart) = v)$$

A change in cardiac output causes the same change in blood pressure:

$$\forall v(output(heart) = v \rightarrow pressure(blood) = v)$$

A change in bloodpressure causes an inverse change in renin levels:

$$\forall v(pressure(blood) = v \rightarrow \neg(conc(blood, renin) = v))$$

Finally, we need to express that 'increased' and 'decreased' are different notions:

$$\neg(increased = decreased)$$

This completes our formalization of the causal knowledge concerning the renin–angiotensin–aldosterone system.

After (automatic) conversion of this logical theory T to clausal form, a resolution-based theorem prover is capable of deriving in six steps

$$T \cup \{pressure(blood) = decreased\} \vdash \Box$$

the last step being the derivation of

$$pressure(blood) = increased$$

among others via the intermediate derivation of

$$Vasoconstriction(arteries, peripheral)$$

yielding a contradiction with

$$pressure(blood) = decreased$$

2.9 Logic as a representation formalism

Compared to other knowledge representation formalisms in artificial intelligence, logic has the great advantage of having a clear syntax and semantics. A logical deductive system in principle offers a set of inference rules, which is sound and complete: each formula derived using such a set of inference rules has a meaning that is unique in terms of the meaning of the formulas it was derived from. So, logic offers a starting point for studying the foundations of knowledge representation and manipulation.

First-order logic in its pure form however has hardly ever been used as a knowledge representation formalism in intelligent systems. This is partly due to the difficulty of expressing domain knowledge in logical formulas. When in a specific problem domain the knowledge is not available in a form 'close' to logic, a lot of energy has to be invested into converting intelligent knowledge to logical formulas, and in this process often valuable information is lost. Moreover, the type of logic that has been dealt with in this chapter, which is sometimes called *standard logic*, is not suitable for encoding all types of knowledge. For example, reasoning about time, and reasoning about reasoning strategies to be followed, often called meta-inference, cannot be represented directly in first-order predicate logic. Moreover, in standard logic it is not possible to handle incomplete and uncertain information, or to deal adequately with exceptional cases to general rules. Currently however, a lot of research is going on concerning *non-standard logics* for expressing such concepts in a formal way.

Exercises

(2.1) Consider the interpretation $v : PROP \to \{\mathit{true}, \mathit{false}\}$ in propositional logic, which is defined by $v(P) = \mathit{false}$, $v(Q) = \mathit{true}$ and $v(R) = \mathit{true}$. What is the truth value of the formula $((\neg P) \wedge Q) \vee (P \to (Q \vee R))$ given this interpretation v?

(2.2) For each of the following formulas in propositional logic determine whether it is valid, invalid, satisfiable, unsatisfiable or a combination of these, using truth tables:

(a) $P \vee (Q \to \neg P)$
(b) $P \vee (\neg P \wedge Q \wedge R)$
(c) $P \to \neg P$
(d) $(P \wedge \neg Q) \wedge (\neg P \vee Q)$
(e) $(P \to Q) \to (Q \to P)$

(2.3) Suppose that $F_1, \ldots, F_n$, $n \geq 1$, and G are formulas in propositional logic, such that the formula G is a logical consequence of $\{F_1, \ldots, F_n\}$. Construct the truth table of the implication $F_1 \wedge \cdots \wedge F_n \to G$. What do you call such a formula?

(2.4) Prove the following statements using the laws of equivalence for propositional logic:

(a) $P \to Q \equiv \neg P \to \neg Q$
(b) $P \to (Q \to R \equiv (P \wedge Q) \to R$
(c) $(P \wedge \neg Q) \to R \equiv (P \wedge \neg R) \to Q$
(d) $P \vee (\neg Q \vee R) \equiv (\neg P \wedge Q) \to R$

(2.5) Prove that the proposition $((P \to Q) \to P) \to P$, known as Peirce's law, is a tautology, using the laws of equivalence in propositional logic and the property that for any propositions π and ϕ, the formula $\pi \vee \neg\phi \vee \phi$ is a tautology.

(2.6) In each of the following cases, we restrict ourselves to a form of propositional logic only offering a limited set of logical connectives. Prove by means of the laws of equivalence that every formula in full propositional logic can be translated into a formula only containing the given connectives:

(a) the connectives $\neg$ and $\vee$.

Table 2.6: Meaning of Sheffer stroke.

F	G	$F\|G$
true	*true*	*false*
true	*false*	*true*
false	*true*	*true*
false	*false*	*true*

(b) the connective $|$ which is known as the Sheffer stroke; its meaning is defined by the truth table given in Table 2.6.

(2.7) Consider the following formula in first-order predicate logic: $\forall x(P(x) \vee Q(y))$. Suppose that the following structure

$$S = (\{2,3\}, \varnothing, \{A : \{2,3\} \to \{\mathit{true}, \mathit{false}\}, B : \{2,3\} \to \{\mathit{true}, \mathit{false}\})$$

is given. The predicates A and B are associated with the predicate symbols P and Q, respectively. Now, define the predicates A and B, and a valuation v in such a way that the given formula is satisfied in the given structure S and valuation v.

(2.8) Consider the following statements. If a statement is correct, then prove its correctness using the laws of equivalence; if it is not correct, then give a counterexample.

(a) $\forall x P(x) \equiv \neg\exists x \neg P(x)$

(b) $\forall x \exists y P(x,y) \equiv \forall y \exists x P(x,y)$

(c) $\exists x(P(x) \to Q(x)) \equiv \forall x P(x) \to \exists x Q(x)$

(d) $\forall x(P(x) \vee Q(x)) \equiv \forall x P(x) \vee \forall x Q(x)$

(2.9) Transform the following formulas into the clausal form of logic:

(a) $\forall x \forall y \exists z(P(z,y) \wedge (\neg P(x,z) \to Q(x,y)))$

(b) $\exists x(P(x) \to Q(x)) \wedge \forall x(Q(x) \to R(x)) \wedge P(a)$

(c) $\forall x(\exists y(P(y) \wedge R(x,y)) \to \exists y(Q(y) \wedge R(x,y)))$

(2.10) For each of the following sets of clauses, determine whether or not it is satisfiable. If a given set is unsatisfiable, then give a refutation of the set using binary resolution; otherwise give an interpretation satisfying it:

(a) $\{\neg P \vee Q, P \vee \neg R, \neg Q, \neg R\}$

(b) $\{\neg P \vee Q \vee R, \neg Q \vee S, P \vee S, \neg R, \neg S\}$

(c) $\{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$

(d) $\{P \vee \neg Q, Q \vee R \vee \neg P, Q \vee P, \neg P\}$

(2.11) Let E be an expression and let σ and θ be substitutions. Prove that $E(\sigma\theta) = (E\sigma)\theta$.

(2.12) For each of the following sets of expressions, determine whether or not it is unifiable. If a given set if unifiable, then compute a most general unifier:

(a) $\{P(a,x,f(x)), P(x,y,x)\}$
(b) $\{P(x,f(y),y), P(w,z,g(a,b))\}$
(c) $\{P(x,z,y), P(x,z,x), P(a,x,x)\}$
(d) $\{P(z,f(x),b), P(x,f(a),b), P(g(x),f(a),y)\}$

(2.13) Use binary resolution to show that each one of the following sets of clauses is unsatisfiable:

(a) $\{P(x,y) \vee Q(a,f(y)) \vee P(a,g(z)), \neg P(a,g(x)) \vee Q(a,f(g(b))), \neg Q(x,y)\}$
(b) $\{append(nil,x,x), append(cons(x,y),z,cons(x,u)) \vee \neg append(y,z,u), \neg append(cons(1,cons(2,nil)),cons(3,nil),x)\}$
(c) $\{R(x,x), R(x,y) \vee \neg R(y,x), R(x,y) \vee \neg R(x,z) \vee \neg R(z,y), R(a,b), \neg R(b,a)\}$

Remark. The first three clauses in exercise (c) define an equivalence relation.

(2.14) Consider the set of clauses $\{\neg P, P \vee Q, \neg Q, R\}$. We employ the set-of-support resolution strategy. Why do we not achieve a refutation if we set the set of support initially to the clause R?

(2.15) Develop a logic knowledge base for a problem domain you are familiar with.

Chapter 3

Production Rules and Inference

In the early 1970s, Alan Newell and Herbert Simon introduced the notion of a *production system* as a psychological model of human behaviour. In this model, part of the human knowledge is being represented in separate units called *productions* or *production rules*. These units contain information concerning actions a person has to take upon the perception of certain stimuli from the environment. Such actions may affect a person's view on the environmental reality, on the one hand because previous assumptions may have to be revised, and on the other hand because possibly new phenomena have to be explained. The model of Newell and Simon closely resembles the two-process theory of memory in cognitive psychology, where two different mechanisms for the storage of incoming sensory information are distinguished: the short-term memory, and the long-term memory, respectively. The short-term memory only contains a limited amount of rapidly decaying information. It corresponds to the part of a production system in which input and derived data are kept. The long-term memory is for permanent storage of information, and corresponds to the rule base of a production system in which the production rules are specified. The production-rule formalism has been employed by many other researchers in addition to Newell and Simon. Most of them, however, view the production-rule formalism merely as a formal language for expressing certain types of knowledge. The formalism has for example been used in the Heuristic DENDRAL system for predicting the molecular structure of compounds, as has been discussed in chapter 1. Part of the knowledge necessary for the purpose of this system has been encoded by means of production rules. The greatest success of the formalism, however, came with the building of the MYCIN and EMYCIN systems, in which the suitability of production rules for building diagnostic intelligent systems was convincingly shown. Another successful system, more directly employing the work of Newell and Simon, is OPS5, and its successor CLIPS, which will be discussed in Chapter 8.

Many present-day intelligent systems use the production-rule formalism as a knowledge representation scheme. Practical experience with production rules has proven this formalism to be particularly suitable in solving classification problems in which the available knowledge takes the form of rules of thumb. In other types of applications, such as design and planning, production rules have been applied with success as well. The suitability of the production system approach for building certain types of intelligent systems not only depends on the production-rule formalism itself, but also on the type of inference method employed for rule-based reasoning.

In the present chapter, we closely look at a number of important notions from production systems. Section 3.1 discusses the various schemes for representing knowledge a production system offers. We proceed by discussing the two basic reasoning methods for systems with production rules in Section 3.2. To conclude, Section 3.3 pays attention to limitations of production rules.

3.1 Knowledge representation in a production system

A production system offers a number of formalisms for representing expert knowledge. The most important of these, of course, is the production-rule formalism, in which the actual problem-solving knowledge is expressed. The entire set of production rules in a production system is called its *rule base*. In addition to the production-rule formalism, a production system provides a means for defining the objects referred to in the production rules, called the *domain declaration* in this book. The rule base and the domain declaration together constitute the *knowledge base* of the production system. These and other schemes for representing knowledge will be discussed in detail in the subsequent sections.

3.1.1 Variables and facts

During a consultation of the knowledge base of a production system, information is constantly being added, removed, or modified as a result of the application of production rules, of data entered by the user, or as a result of querying some database. The facts that become known to the system during a consultation, are stored in a so-called *fact set*, also known as the *global database* or *working memory* of the system.

Factual information can be represented in a number of ways. One simple way is to represent facts by means of *variables* which can take either a single constant or a set of constants as a value. Note that the set of all variables defined in a production system together with their possible values, presents a picture of the information which is relevant in the field modelled in the system.

In general, two types of variable are discerned:

- *single-valued variables*, that is, variables which can take at most one constant value at a time, and
- *multi-valued variables*, that is, variables which can take a set of constants for a value.

Single-valued variables are used to represent information which in the case under consideration is unique; multi-valued variables are used for representing a collection of interrelated facts.

EXAMPLE 3.1

In a intelligent system, a variable with the name *capabilities* may be used for storing information about the capabilities of a certain person. This variable has to be multi-valued, because people may have more than one capability at the same time. An example of a single-valued variable is the *gender* of a person.

Properties of variables, such as whether they are single- or multi-valued, and usually also information concerning the values a variable is allowed to take, are all described in the domain declaration of a knowledge base. The following definition provides a formal description of such a domain declaration.

Definition 3.1 *Let τ denote a nonempty set of constants, called a* type. *A* typed variable declaration *is an expression of one of the following forms:*

- $x^s : \tau$*, where* x^s *is a single-valued variable;*
- $x^m : 2^\tau$*, where* x^m *is a multi-valued variable.*

Untyped variable declarations *are expressions of the form* x^s*, in the single-valued case, or* x^m*, in the multi-valued case. A set* D *of variable declarations for all variables occurring in the knowledge base is called the* domain declaration *of the knowledge base.*

Examples of types are the set of integer numbers, denoted by **int**, the set of real numbers, denoted by **real**, and finite sets of constants such as $\{\mathit{intelligent}, \mathit{attractive}, \mathit{outspoken}\}$. Note that a domain declaration is similar to a variable declaration part in, for instance, Pascal. It restricts the values a variable may take.

A variable together with the value(s) it has adopted during a consultation is called a *fact*.

Definition 3.2 *A* fact *is a statement having one of the following forms:*

- $x^s = c$*, where* $c \in \tau$ *if* x^s *is a single-valued variable declared as* $x^s : \tau$*;*
- $x^m = C$*, where* $C \subseteq \tau$ *if* x^m *is a multi-valued variable declared as* $x^m : 2^\tau$*.*

A fact set *has the following form:*

$$\{x_1^s = c_1, \ldots, x_p^s = c_p, x_1^m = C_1, \ldots, x_q^m = C_q\}$$

where c_i *are constants and* C_j *are sets of constants. A variable may only occur once in a fact set.*

EXAMPLE 3.2 ______

Consider the following domain declaration

$$\begin{array}{rcl} D & = & \mathit{gender} : \{\mathit{female}, \mathit{male}\}, \\ & & \mathit{age} : \mathbf{int}, \\ & & \mathit{personality} : 2^{\{\mathit{intelligent}, \mathit{outspoken}, \mathit{arrogant}\}}, \\ & & \mathit{hire} : \{\mathit{yes}, \mathit{no}\} \end{array}$$

of a knowledge base of an intelligent system. The following facts are typical elements of a fact set after a specific consultation:

$$\begin{array}{lcl} \mathit{gender} & = & \mathit{male} \\ \mathit{age} & = & 27 \\ \mathit{personality} & = & \{\mathit{intelligent}, \mathit{outspoken}\} \\ \mathit{hire} & = & \{\mathit{yes}\} \end{array}$$

The statement 'x^s = *unknown*' (or 'x^m = *unknown*', respectively) is used to indicate that the variable x^s (or x^m, respectively) has not been assigned an actual value; x^s (or x^m) is then called *unknown*. The constant *unknown* has a special meaning: it expresses that the inference engine has not been able to derive one or more values for the variable. Since its meaning goes beyond (that is, *meta*) the contents of the fact set and the knowledge base, the constant *unknown* is called a *meta-constant*.

It must be emphasized that a fact set is not a part of a knowledge base, but instead is a separate component of the system. A fact set comprises information which is specific for a particular consultation, whereas a knowledge base only contains *declarations* of variables and therefore does not specify consultation-dependent values. In the following, we will frequently assume that suitable variable declarations are present in the domain declaration of a knowledge base, without explicitly referring to them.

3.1.2 Conditions and conclusions

At the beginning of this chapter, we mentioned that production rules are used for representing problem-solving knowledge from a specific problem domain. The major part of this type of knowledge takes the form of *heuristic rules* or rules of thumb, which, as we shall see, are the informal, real-life analogies of production rules. In a heuristic rule, several conditions and conclusions are interrelated, as follows:

```
if
    certain conditions are fulfilled,
then
    certain conclusions may be drawn.
```

EXAMPLE 3.3

An example of a heuristic rule, taken from the domain of the assessment of applicants for a job, is the following:

```
if
    the person's age less than 60, and
    the person is intelligent, and
    the person is not arrogant
then
    the person will be hired
```

In the process of knowledge engineering, such heuristic rules have to be transformed into their formal counterparts, that is, into production rules. A production rule just like a heuristic rule, consists of a number of conditions and conclusions. In a production rule, which unlike a heuristic rule is a formal statement, the conditions and conclusions comprise the following elements:

- symbolic and numeric constant values
- variables

- predicates and actions

An important part of the translation process therefore concerns the identification of the variables and constants that are relevant in the heuristic rules.

EXAMPLE 3.4

Consider the preceding heuristic rule once more. In the first condition a variable *age* may be identified, the second and third conditions concerns a person's *personality*, and, finally, the conclusion concerns the question whether the firm is going to *hire* the person. The following constant values may be introduced to represent the information further comprised in the heuristic rule: 60, *intelligent*, *arrogant* and *yes*.

Several syntactic forms have been devised for the representation of production rules. In the present book we employ the syntax described in the following definition.

Definition 3.3 *A* production rule *is a statement having the following form:*

$$\begin{array}{lcl}
\langle\text{production rule}\rangle & ::= & \textbf{if } \langle\text{antecedent}\rangle \textbf{ then } \langle\text{consequent}\rangle \textbf{ fi} \\
\langle\text{antecedent}\rangle & ::= & \langle\text{disjunction}\rangle \ \{\textbf{and } \langle\text{disjunction}\rangle\}^* \\
\langle\text{disjunction}\rangle & ::= & \langle\text{condition}\rangle \ \{\textbf{or } \langle\text{condition}\rangle\}^* \\
\langle\text{consequent}\rangle & ::= & \langle\text{conclusion}\rangle \ \{\textbf{also } \langle\text{conclusion}\rangle\}^* \\
\langle\text{condition}\rangle & ::= & \langle\text{predicate}\rangle\textbf{(}\langle\text{variable}\rangle\textbf{,}\langle\text{constant}\rangle\textbf{)} \\
\langle\text{conclusion}\rangle & ::= & \langle\text{action}\rangle\textbf{(}\langle\text{variable}\rangle\textbf{,}\langle\text{constant}\rangle\textbf{)} \\
\langle\text{predicate}\rangle & ::= & \textit{same} \mid \textit{notsame} \mid \textit{greaterthan} \mid \ldots \\
\langle\text{action}\rangle & ::= & \textit{add} \mid \textit{remove} \mid \ldots
\end{array}$$

In the production-rule formalism it is assumed that the **or** operator has a higher precedence than the **and** operator. Note that the nesting of conjunctions and disjunctions is limited; this is typical for production systems.

A condition is built from a *predicate* and two associated arguments: a variable and a constant. By means of its predicate, a condition expresses a comparison between the specified constant value and the actual value(s) the specified variable has adopted. In the context of production systems, a predicate is a function which upon evaluation returns either the truth value *true* or the value *false*. The way predicates are evaluated is illustrated by means of the following example.

EXAMPLE 3.5

Let F be the following fact set:

$$F = \{\mathit{age} = 70, \mathit{personality} = \{\mathit{intelligent}, \mathit{arrogant}\}\}$$

where *age* is a single-valued variable and *personality* is a multi-valued one. Now consider the following condition:

$$\mathit{same}(\mathit{personality}, \mathit{intelligent})$$

The predicate *same* returns upon evaluation the truth value *true* if *intelligent* is one of the constants in the set of constants adopted by the multi-valued variable *personality*; otherwise, the value *false* is returned. So, in the present example, the evaluation of the condition will return the value *true*. In case of a single-valued variable, *same* tests whether or not the constant specified as its second argument is equal to the constant which the variable mentioned as its first argument has adopted. Given the present fact set, the condition

$$\mathit{lessthan}(\mathit{age}, 60)$$

therefore yields the value *false* upon evaluation.

A conclusion is built from an *action* and two associated arguments. An action can be considered to operate on a variable. The most frequently applied action is *add*, which adds the constant specified as its second argument to the value set of the multi-valued variable mentioned in its first argument; in case of a single-valued variable, the action *add* assigns the constant value from its second argument to the specified variable.

EXAMPLE 3.6

Consider the fact set F:

$$F = \{\mathit{personality} = \{\mathit{arrogant}\}, \mathit{gender} = \mathit{male}, \mathit{age} = 25\}$$

in which *personality* is the only multi-valued variable. The action *add* in the conclusion

$$\mathit{add}(\mathit{personality}, \mathit{outspoken})$$

adds the constant *outspoken* to the set of constant values the variable *personality* already has adopted. So, after evaluation of this conclusion we have obtained the following fact set F':

$$F' = \{\mathit{personality} = \{\mathit{arrogant}, \mathit{outspoken}\}, \mathit{gender} = \mathit{male}, \mathit{age} = 25\}$$

A production rule is built from such conditions and conclusions.

EXAMPLE 3.7

The heuristic rule informally introduced above can now be translated into the production-rule formalism, for example as follows:

```
if
    lessthan(age, 60) and
    same(personality, intelligent) and
    notsame(personality, arrogant)
then
    add(hire, yes)
fi
```

To conclude, the following example discusses a production rule comprising more than one conclusion.

EXAMPLE 3.8

The following heuristic rule:

```
if
    the person is quiet, and
    the person is calm
then
    it is likely the person is introvert, and
    it is conceivable that the person is stable
```

may be expressed in the production-rule formalism as follows:

```
if
    same(expression, quiet) and
    same(emotionality, calm)
then
    add(person, introvert) also
    add(person, stable)
fi
```

The only action we have considered up till now is the action *add*. If this action is the only one specified in the consequent of a production rule, then the rule closely resembles a logical implication, in which the conditions of the rule appear on the left of the implication symbol and the conclusions are specified on the right of it. This interpretation, however, is no longer valid when actions other than *add* have been specified in the consequent of a rule. Consider, for example, the action *remove* which upon execution cancels the assignment of a specific constant to a single-valued variable, or deletes a constant from the set of constants of a multi-valued variable. A production rule in which this action has been specified cannot possibly be viewed as a logical implication. More about the correspondence between the production-rule formalism and logic will be said in Section 3.1.4.

In Table 3.1 the semantics of some frequently used predicates and actions are described. From the special meaning of the constant *unknown* as discussed in the preceding section, we have that the predicates *known* and *notknown* mentioned in the table have a special meaning as well: they express knowledge concerning the derivability of values for variables, which again goes beyond the contents of the fact set and the rule base of a production system. We call such predicates *meta-predicates*. In Section 3.2.1 we will turn again to the meanings of the predicates and the effects the different actions have on a fact set.

3.1.3 Object-attribute-value tuples

The production-rule formalism introduced in the preceding section does not provide a means for specifying formal relationships between the variables of concern. In many problem do-

Table 3.1: Some predicates and actions, and their meaning

Example	*Semantics for single-valued variables*	*Semantics for multi-valued variables*
$\mathit{same}(x, c)$	$x^s = c$	$c \in x^m$
$\mathit{notsame}(x, c)$	$x^s \neq c$ and $x^s \neq \mathit{unknown}$	$c \notin x^m$ and $x^m \neq \mathit{unknown}$
$\mathit{lessthan}(x, c)$	$x^s < c$	-
$\mathit{greaterthan}(x, c)$	$x^s > c$	-
$\mathit{known}(x)$	$x^s \neq \mathit{unknown}$	$x^m \neq \mathit{unknown}$
$\mathit{notknown}(x)$	$x^s = \mathit{unknown}$	$x^m = \mathit{unknown}$
$\mathit{add}(x, c)$	$x^s \leftarrow c$	$x^m \leftarrow x^m \cup \{c\}$
$\mathit{remove}(x, c)$	$x^s \leftarrow \mathit{unknown}$	if $x^m = \{c\}$ then $x^m \leftarrow \mathit{unknown}$ else $x^m \leftarrow x^m \setminus \{c\}$

mains to be modelled however, one can often discern separate subdomains, or *objects*, which are interrelated in some way. Each subdomain then is described by a number of properties or *attributes* which are specific for that subdomain. The idea is illustrated in the following example.

EXAMPLE 3.9

In the heuristic rule from Example 3.8, a description is given of the personality of a person. Personality traits are usually described by several factors which are specific for the person. We now may view *traits* as a separate object, and *character* and *emotional* as attributes belonging to that object.

In production systems, objects are often used to explicitly group the properties which are mentioned in the heuristic rules. The objects themselves are often even further exploited for directing the inference process. If we extend the production-rule formalism to include the specification of an object, predicates and actions have to be changed from binary, in the previous case of variable-value pairs, to ternary operators. We call a tuple consisting of an object, an attribute and a constant value, an *object-attribute-value tuple*, or *o-a-v triple.* The following simple extension to the original syntax definition of production rules is thus obtained:

$$
\begin{aligned}
\langle\text{condition}\rangle &::= \langle\text{predicate}\rangle(\langle\text{object}\rangle,\langle\text{attribute}\rangle,\langle\text{constant}\rangle)\\
\langle\text{conclusion}\rangle &::= \langle\text{action}\rangle(\langle\text{object}\rangle,\langle\text{attribute}\rangle,\langle\text{constant}\rangle)
\end{aligned}
$$

Note that an object-attribute pair always explicitly indicates that the mentioned attribute belongs to the specified object. The object-attribute pairs in an object-attribute-value tuple just act like the variables discussed in the foregoing sections. The attributes of an object are declared as being either *single-valued* or *multi-valued*, in a way resembling the declaration of variables. In the sequel, we shall write $o.a^s$ to denote the single-valued attribute a^s belonging to the object o; we use $o.a^m$ for the multi-valued case.

EXAMPLE 3.10

The following production rule

```
if
    same(person, gender, male) and
    same(person, behaviour, quiet) and
    same(personality, emotional, stable)
then
    add(personality, introvert, yes)
fi
```

concerns the objects *person* and *personality*. Attributes of the object *personality* referred to in the rule are *emotional* and *introvert*. The attributes of the object *person* referred to in the rule are *gender* and *behaviour*.

The extension of the definitions of predicates and actions with the specification of an object enables us to express relationships between objects and their attributes. However, it is still not possible to explicitly express relationships between objects amongst themselves in a natural way. Yet it may be desirable in an intelligent system to have this type of information available as well. Therefore, in addition to the rule base, often an *object schema* is present in an intelligent system; it is usually added to the domain declaration of a knowledge base. An object schema defines the interrelationships between the objects, and the relationships between the objects and their associated attributes. Figure 3.1 shows a portion of the object schema describing some features of a person. An object is represented by an ellipse. A solid line is used to represent a relationship between two objects. In Figure 3.1 the object *personality* is called a *subobject* of the object *person*. A dashed line indicates a relationship between an object and an associated attribute. We note that an object schema is frequently used for storing meta-information, for example directives to the inference engine to handle certain attributes in a special way.

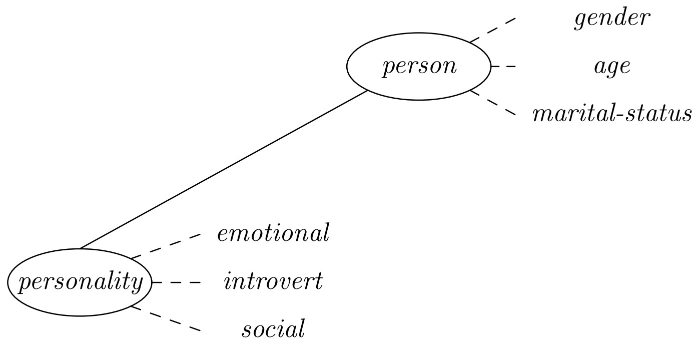


Figure 3.1: An object scheme.

3.1.4 Production rules and first-order predicate logic

In the preceding sections, we have informally discussed the meanings of predicates and actions in a production system. In general, we can say that the semantics of production rules in a production system is described in terms of a specific inference method for applying the rules. The only semantics for a production system available therefore is procedural in nature. This is contrary to first-order logic which also has a neat declarative semantics. There is, however, a strong relationship between the production-rule formalism and the formalism of first-order logic, as we already briefly touched upon in Section 3.1.1. In most rule bases, at least part of the production rules can be translated into first-order predicate logic in a straightforward and natural way. This is fortunate, since it enables developing a rule base without precise knowledge of the working of the inference engine: the declarative readings of the corresponding logical formulas can thus be exploited.

Let us start by looking at the correspondence between conditions and conclusions in production rules on the one hand and literals in first-order logic on the other hand. Without loss of generality, we assume that the production system makes use of object-attribute-value tuples. As we discussed above, the predicates *same* and *notsame* test whether the specified attribute of some object has, respectively has not, the specified constant as a value or in its set of values. Again, we distinguish between multi-valued and single-valued attributes. A multi-valued attribute may be viewed as a relation A defined on the cartesian product of a set of objects and a set of constants, that is, $A \subseteq O \times V$, where O is the set of objects, and V is the set of constant values. Recall that in first-order predicate logic, predicate symbols may be employed for representing such relations. Conditions of the form $\mathit{same}(o, a^m, v)$, and $\mathit{notsame}(o, a^m, v)$, containing a multi-valued attribute, may therefore be translated into first-order logic in the literals $a(o, v)$, and $\neg a(o, v)$, respectively. In the single-valued case, we have to take into account that the attribute may adopt at most one value at a time. Single-valuedness is best expressed in first-order logic by using function symbols. Single-valued attributes can therefore be viewed as functions a from the set of objects O to the set of values V, that is, $a : O \to V$. Note that for expressing the meaning of the predicate *same*, we have the equality predicate $=$ at our disposal. The following translation is now rather straightforward: the condition $\mathit{same}(o, a^s, v)$ is translated into the literal $a(o) = v$, and the condition $\mathit{notsame}(o, a^s, v)$ is translated into $\neg a(o) = v$. Many other predicates used in production rules can be translated in much the same way. Table 3.2 summarizes some of these translations. Note that the semantics of first-order logic implies that the unit clause $a(o, v)$ in the presence of the unit clause $\neg a(o, v)$, leads to an inconsistency. In addition, the unit clauses $a(o) = v$ and $a(o) = w$, where $v \neq w$, are inconsistent in the presence of the equality axioms. As can be seen from Table 3.2, the action *add* is treated the same

Table 3.2: Translation of conditions and actions into first-order logic.

Example	*Logic representation for single-valued attributes*	*Logic representation for multi-valued attributes*
$\mathit{same}(o, a, v)$	$a(o) = v$	$a(o, v)$
$\mathit{notsame}(o, a, v)$	$\neg a(o) = v$	$\neg a(o, v)$
$\mathit{equal}(o, a, v)$	$a(o) = v$	–
$\mathit{lessthan}(o, a, v)$	$a(o) < v$	–
$\mathit{greaterthan}(o, a, v)$	$a(o) > v$	–
$\mathit{add}(o, a, v)$	$a(o) = v$	$a(o, v)$

way as the predicate *same* is; this reflects the transition from a procedural to a declarative semantics. The meta-predicates *known* and *notknown* are not included in Table 3.2, since it is not possible to express meta-information in standard first-order logic. A similar remark can be made concerning the action *remove*. It is noted that special non-standard logics in which such meta-predicates and non-monotonic actions can be expressed, have been developed. The subject of non-standard logic, however, goes beyond the scope of the present book.

Further translation of a production rule into a logical formula is now straightforward. The general translation scheme is as follows:

$$\begin{array}{lll}
\textbf{if} & & \\
\quad c_{1,1} \ \textbf{or} \ c_{1,2} \ \textbf{or} \ \ldots \ \textbf{or} \ c_{1,m} \ \textbf{and} & & ((c'_{1,1} \vee c'_{1,2} \vee \ldots \vee c'_{1,m}) \wedge \\
\quad \ldots & & \ldots \\
\quad c_{n,1} \ \textbf{or} \ c_{n,2} \ \textbf{or} \ \ldots \ \textbf{or} \ c_{n,p} & \Rightarrow & (c'_{n,1} \vee c'_{n,2} \vee \ldots \vee c'_{n,p})) \\
\textbf{then} & & \rightarrow \\
\quad a_1 \ \textbf{also} \ a_2 \ \textbf{also} \ \ldots \ \textbf{also} \ a_q & & (a'_1 \wedge a'_2 \wedge \ldots \wedge a'_q) \\
\textbf{fi} & &
\end{array}$$

where conditions $c_{i,j}$ and actions a_k are translated into literals $c'_{i,j}$ and a'_k, respectively, as prescribed by Table 3.2 From the table it can readily be seen that the kind of production rules we have looked at, are translated into *ground* logical implications.

The following example illustrates the translation scheme.

EXAMPLE 3.11

Consider the following production rule:

if
  greaterthan(*blood*, *systolic-pressure*, 140) **and**
  greaterthan(*heart*, *rate*, 100) **and**
  same(*person*, *skin*, *pale*) **or**
  same(*eyes*, *iris*, *wide*)
then
  add(*person*, *condition*, *rage*)
fi

Translation of this rule into first-order logic yields the following implication:

$$\begin{array}{l}
(\textit{systolic-pressure}(\textit{blood}) > 140 \wedge \\
\ \textit{rate}(\textit{heart}) > 100 \wedge \\
\ (\textit{skin}(\textit{person}, \textit{pale}) \vee \\
\ \ \textit{eyes}(\textit{iris}, \textit{wide}))) \\
\rightarrow \\
\textit{condition}(\textit{person}, \textit{rage})
\end{array}$$

The formalism of first-order logic is in certain ways more flexible than the production-rule formalism is. For instance, it is not restricted to the use of *o-a-v* triples only; it allows, for example, to specify the literal $\textit{systolic-pressure}(\textit{person}, \textit{blood}) > 140$, instead of the first literal in the implication shown above. The latter literal is more in correspondence with the intended meaning of the original heuristic rule.

To conclude, the fact set of a production system may be translated into the logic formalism much in the same way as the rule base is. A fact $o.a^s = v$ concerning a single-valued attribute a^s is translated into a unit clause $a(o) = v$. Recall that in the presence of the equality axioms single-valuedness is guaranteed. Now consider the multi-valued case. A fact concerning a multi-valued attribute a^m is translated into a set of unit clauses $a(o, v_i)$ for each v_i in the set of constant values a^m has adopted. However, it is not sufficient to only add these positive clauses: it is typical for production systems that values of multi-valued attributes not explicitly entered into the fact set, are taken implicitly by the system as being not true. This behaviour has been copied from human problem solving. For example, a medical doctor usually only records in a patient report the symptoms and signs that have actually been observed in the patient; all information not explicitly recorded for the specific patient is implicitly assumed to be negative. This aspect of problem solving is reflected in the meaning of the *notsame* predicate. Note that this way of dealing with negations is quite different from the meaning of a negative literal in first-order logic which only holds in a model in which it has actually been satisfied. For a correct meaning of the *notsame* predicate therefore we have to add explicitly unit clauses $\neg a(o, v_i)$ for the remaining constants v_i occurring in the type τ of a^m, as soon as at least one positive unit clause $a(o, v_j)$, $j \neq i$, occurs in the fact set (in case of an untyped variable, we add unit clauses $\neg a(o, v_i)$ for the remaining constants v_i mentioned in the rule base). The explicit addition of negative unit clauses is called *negation by absence*. It is a special case of the closed world assumption mentioned before in Chapter 2.

EXAMPLE 3.12

Consider the following domain declaration:

$$D = \{disorder^m : 2^{\{flu, hepatitis\}}, age^s : \mathbf{int}\}$$

and the fact set $F = \{disorder^m = \{flu\}, age^s = 70\}$. We obtain the following translation into first-order logic:

$disorder(patient, flu)$
$\neg disorder(patient, hepatitis)$
$age(patient) = 70$

3.2 Inference in a production system

Several inference methods have been devised for dealing with production rules. An inference method in a production system explicitly exploits the difference between facts and rules: it operates on the fact set, which can be looked upon as the global working memory for the production rules. Note that in logic, such an explicit separation between facts (unit clauses) and rules (implications) generally is not made, although it should be noted that several inference methods, such as SLD resolution, make use of a similar distinction.

Roughly speaking, an inference method selects and subsequently applies production rules from a rule base. In applying the selected production rules, it executes the actions specified in their conclusions. Execution of such actions may cause facts to be added to, to be modified in, or to be deleted from the fact set. In this chapter we discuss the addition and deletion of facts: a discussion of the modification of facts as a result of executing actions will be postponed until Chapter 8, where we shall discuss the language OPS5 and CLIPS. Figure 3.2 shows the general idea of inference in a production system. The manipulation of production rules and facts is depicted in the figure by means of arrows.

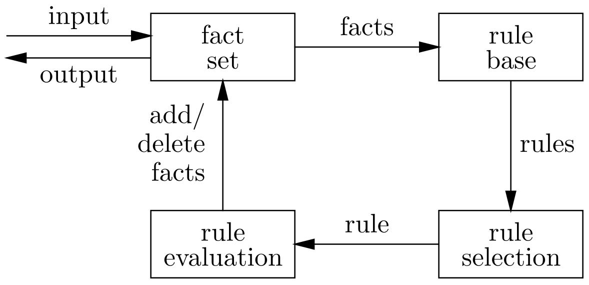


Figure 3.2: Global architecture of a production system.

The distinction between the two basic forms of inference mentioned in chapter 1 is essential when considering production systems; they yield entirely different reasoning strategies. Before discussing the two basic inference methods, top-down and bottom-up inference, in detail, we introduce them informally with the help of a simplified example, in which we abstract from the syntactical structure of conditions, conclusions, and facts. To simplify matters further, we suppose that all conditions of the form c succeed upon evaluation in the presence of the fact c. Now, consider Table 3.3.

Table 3.3: Production system before and after execution.

State	*Component*	*Top-down Inference*	*Bottom-up Inference*
Initial	*Goal*	g	–
	Facts	$\{a\}$	$\{a\}$
	Rules	R_1 : **if** b **then** g **fi**	R_1 : **if** b **then** g **fi**
		R_2 : **if** g **then** c **fi**	R_2 : **if** g **then** c **fi**
		R_3 : **if** a **then** b **fi**	R_3 : **if** a **then** b **fi**
Final	*Facts*$'$	$\{a, b, g\}$	$\{a, b, c, g\}$

- *Top-down inference* starts with a statement of one or more *goals* to be achieved. In our example, we have just the single goal g. A goal may match with a conclusion of one or more production rules present in the rule base. All production rules thus matching with a certain goal are selected for application. In the present case, the only rule selected is R_1. Each one of the selected production rules is subsequently applied by first considering the conditions of the rule as the new subgoals to be achieved. Roughly speaking, if there are facts present in the fact set which match with these new subgoals, then these subgoals are taken as been achieved; subgoals for which no matching facts can be found, are matched against the conclusions of the production rules from the rule base. Again, matching production rules are selected for application. In our example, we have from the rule R_1 the new subgoal b, which in turn causes the selection of the production rule R_3. This process is repeated recursively. Note that in top-down inference, production rules are applied in a backward manner. When all the subgoals, that is, the conditions of a selected production rule, have been achieved, then the actions in the conclusions of the rule are executed, possibly causing changes in the fact set. Since the subgoal a of the selected production rule R_3 matches with the fact a, we have that the condition of the rule is fulfilled. Subsequently, its action is executed, yielding the new fact b. This new fact in turn fulfills the condition b of rule R_1, which led to the selection of R_3. The inference process is terminated as soon as the initially specified goals have been achieved. Note that only production rules relevant for achieving the initially given goals are applied. This explains why the rule R_2 in table 3.3 has not been used.

- *Bottom-up inference* starts with a fact set, in our example $\{a\}$. The facts in the fact set are matched against the conditions of the production rules from the rule base. If for a specific production rule all conditions are fulfilled, then it is selected for application. The rule is applied by executing the actions mentioned in its conclusions. So, in our example the rule R_3 will be applied first. The application of the selected production rules is likely to result in changes in the fact set, thus enabling other production rules to be applicable. In the present case, after the application of rule R_3, we have obtained the new fact set $\{a, b\}$, which results in the rule R_1 now being applicable. The fact g added to the fact set as a result of executing the action of rule R_1, results in the subsequent application of R_2. We therefore conclude that the final fact set is equal to $\{a, b, c, g\}$. The inference process is terminated as soon as all applicable production rules have been processed.

Note the difference in the resulting fact sets in table 3.3 obtained from applying top-down and bottom-up inference, respectively. As a consequence of their different inference behaviour, top-down and bottom-up inference are suitable for developing different kinds of intelligent systems. Top-down inference is often used in inference engines of diagnostic intelligent systems in which the inference process is controlled by a specific goal and a small amount of data. Bottom-up inference is most suitable for applications in which the interpretation of a vast amount of data is important, and in which there are no preset goals. In the following section, we first address top-down inference before discussing bottom-up inference in Section 3.2.4.

3.2.1 Top-down inference and production rules

As we discussed before, top-down inference usually is incorporated in diagnostic intelligent systems. If it employs the variable-value representation, such a system tries to derive facts concerning one or more preset goal variables. In a typical medical diagnostic intelligent system for example, one could think of a multi-valued variable *diagnosis*, for which we want to establish for a specific patient the set of values (that is, the possible disorders). So, in general, top-down inference starts with a set of *goals* $\{G_1, G_2, \ldots, G_m\}$, $m \geq 1$, essentially being goal variables. For this purpose, in the domain-declaration part of a knowledge base, the variable declarations are extended to include a specification of whether or not they are goals. In the sequel, we shall use x_g^s to denote a single-valued goal variable, and x_g^m to denote a multi-valued goal variable. The subset of all facts concerning the goal variables in the fact set after applying the inference algorithm is called a *solution*.

The top-down inference algorithm used in this book for establishing the goals G_i from the initial set of goals, informally amounts to the following. Suppose that we are given a fixed rule base $\{R_1, R_2, \ldots, R_n\}$, $n \geq 1$, of production rules R_i. For deriving values for a multi-valued variable, the inference engine will try and apply production rules as long as there are production rules available which can extend the set of constants for the goal variable by adding new ones to it. In the single-valued case, the inference engine proceeds until a single value has been derived for the variable, and then it terminates. If the system has not been able to derive values for the variable from the rule base, for instance because applicable production rules are absent, then on some occasion it will turn to the user and ask for additional information. For this purpose a distinction is made between variables for which values may be asked from the user, these variables are called *askable variables*, and variables which may not be asked. The askability of a variable is again denoted in the domain declaration of the variables, this time by means of a subscript a; so, we have x_a^m for askable multi-valued variables and x_a^s for askable single-valued ones.

EXAMPLE 3.13

In a medical intelligent system intended for diagnosing a patient's disorder, it is rather undesirable that the user is asked to specify the patient's disorder when the system has not been able to attain a diagnosis. Therefore, the variable *disorder* should not be askable. On the other hand, *symptom* and *gender* are typical examples of askable variables: these can usually not be derived using the production rules.

It will be evident that goal variables should never be askable.

The entire process of deriving values for a variable by applying the production rules, and possibly asking the user for values for it, is called *tracing* the variable. In the sequel, we abstract from the distinction between single- and multi-valued variables in the description of the top-down inference algorithm, since the basic structure of the algorithm is the same for both types of variable. The algorithm for tracing a variable pictured in the foregoing is described by the following procedure:

```
procedure TraceValues(variable)
    Infer(variable);
    if not established(variable) and askable(variable) then
        Ask(variable)
    fi
end
```

The function call established(variable) is used to examine whether or not a value for the variable variable has been obtained from the rule base by means of the procedure call Infer(variable); askable(variable) is used to determine whether the variable concerned is askable.

For deriving values for a variable from the rule base, called *inferring* the variable, the procedure Infer is invoked. In the procedure Infer, a subset $\{R_{i_1}, R_{i_2}, \ldots, R_{i_k}\}$ of the production rules is selected; a rule R_j will be selected if the name of the given variable occurs in one of the conclusions of the rule, in other words, if the given variable and the variable in the conclusion *match*. The thus selected rules are then *applied*. The procedure for selecting production rules and subsequently applying them is shown below:

```
procedure Infer(variable)
    Select(rule-base, variable, selected-rules);
    foreach rule in selected-rules do
        Apply(rule)
    od
end
```

The actual selection of the relevant production rules from the rule base is described in the following procedure:

```
procedure Select(rule-base, variable, selected-rules)
    selected-rules ← ∅;
    foreach rule in rule-base do
        matched ← false;
        foreach concl in consequent(rule) while not matched do
            pattern ← variable(concl);
            if Match(variable, pattern) then
                selected-rules ← selected-rules ∪ {rule};
                matched ← true
            fi
        od
    od
end
```

The set of selected production rules is called the *conflict set*. In the **foreach** statement in the Infer procedure this conflict set is traversed: the rules from this set are applied one by one by means of a call to the procedure Apply, which will be described shortly. Note that this way the rules are applied exhaustively, that is, all rules concluding on the variable which is being traced, are applied.

Neither the order in which the production rules from the conflict set are applied, nor the order in which the conditions and the conclusions of the rules are evaluated, is fixed as yet. If the order in applying the selected rules has not been fixed, we speak of *nondeterminism of the first kind*. The evaluation order of the conditions and the conclusions in a rule not being fixed is called *nondeterminism of the second kind*. Nondeterminism of the first kind is resolved by using a so-called *conflict-resolution strategy*, which imposes some order on the rules from the conflict set. The simplest conflict-resolution strategy is, of course, just to apply the production rules in the order in which they have been selected from the rule base (which is then viewed as a sequence of production rules instead of as a set). More sophisticated conflict-resolution strategies order the conflict set using some context-sensitive criterion. An example of such a strategy is ordering the rules according to the number of conditions not yet fulfilled; this way, solutions which are 'close' to the information already available to the system generally prevail over more remote solutions. From the user's point of view, a system provided with a context-sensitive conflict-resolution strategy behaves much more intelligently, since likely solutions are explored before unlikely ones. Nondeterminism of the second kind is usually handled by evaluating the conditions and conclusions of a selected production rule in the order of their appearance. However, more sophisticated techniques are also possible. Sophisticated conflict-resolution strategies and evaluation ordering methods are seldom used in intelligent systems using top-down inference, since the goal-directed nature of top-down inference is itself an ‘intelligent’ control strategy, rendering additional ones less necessary. As a consequence, most systems employing top-down inference use the simplest strategies in solving the two types of nondeterminism, that is, they apply production rules, and evaluate conditions and conclusions in the order of their specification. This particular strategy will be called *backward chaining*.

The application of a selected production rule commences with the evaluation of its conditions. If upon evaluation at least one of the conditions is found to be *false*, then the rule is said to *fail*. If, on the other hand, all conditions evaluate to be *true*, then the rule is said to *succeed*. The application of a production rule is described in the following procedure:

```
procedure Apply(rule)
    EvalConditions(rule);
    if not failed(rule) then
        EvalConclusions(rule)
    fi
end
```

The procedure Apply first evaluates the condition part of the rule by calling EvalConditions. If this evaluation ends in success, then it evaluates the conclusions of the rule by means of EvalConclusions.

Let us take a closer look at the procedure EvalConditions which checks whether all conditions in the antecedent of the production rule yield the truth value *true* upon evaluation. Beginning with the first condition, the procedure traces the variable occurring in the condition by means of a recursive call to TraceValues. Subsequently, the test specified by means of the predicate in the condition is executed. In the EvalConditions procedure presented below, we have assumed, for simplicity's sake, that the antecedent of a production rule only comprises a conjunction of conditions.

```
procedure EvalConditions(rule)
    foreach condition in antecedent(rule) do
        var ← variable(condition);
        TraceValues(var); indirect recursion
        ExecPredicate(condition);
        if condition failed then
            return
        fi
    od
end
```

It should be noted that there are many ways to optimize the last procedure, several of which will be discussed below. The ExecPredicate procedure executes the test denoted by the predicate which has been specified in the condition under consideration. It compares the value the mentioned variable has with the constant specified in its second argument (if any). This test yields either the truth value *true* or *false*. We have already seen an example of the execution of such a predicate in the preceding section.

If all conditions of the production rule have been evaluated and have yielded the value *true*, then the rule succeeds, and its conclusions are subsequently evaluated. The evaluation of the conclusion part of a successful production rule merely comprises the execution of the actions specified in its conclusions:

```
procedure EvalConclusions(rule)
    foreach conclusion in consequent(rule) do
        ExecAction(conclusion)
    od
end
```

We have mentioned before that executing the action *add* results in the assignment of a constant value to a single-valued variable, or in the addition of a constant value to the set of values of a multi-valued variable. In Section 3.1.2, we also briefly discussed the action *remove*. Execution of this action results in deleting the specified constant value from the fact concerning the variable. Execution of this action can therefore disrupt the monotonicity of the reasoning process, thus rendering it difficult to reconstruct the inference steps which have been carried out. Furthermore, it is quite conceivable that the action is executed on a variable which has not been traced as yet, in which case the continuation of the inference is undefined. Therefore, in many intelligent systems this action is not allowed, in particular not in those systems employing backward chaining.

It may happen that a certain variable is specified in one of the conditions as well as in one of the conclusions of a production rule. Such a rule is called a *self-referencing production rule*. When applying a self-referencing rule during a top-down inference process as discussed above, the rule may occasion infinite recursion. For the moment, we therefore do not allow self-referencing rules in a rule base. In the following, we shall return to these rules once more in discussing some optimizations of the inference algorithm.

The procedures discussed in the foregoing together constitute the entire top-down inference algorithm. The following example demonstrates the behaviour of a system employing this form of inference.

EXAMPLE 3.14

Let $D = \{x_a^m, y^m, z_a^m, v_g^m, w_a^s\}$ be the domain declaration of some knowledge base. As can be seen, all variables except y and v are askable; v is the only goal variable. Suppose that we initially have the following fact set:

$$F = \{w = 5\}$$

Now consider the following production rules:

R_1 : **if** $same(x, a)$ **and** $same(x, b)$ **then** $add(z, f)$ **fi**
R_2 : **if** $same(x, b)$ **then** $add(z, g)$ **fi**
R_3 : **if** $same(x, d)$ **and** $greaterthan(w, 0)$ **then** $add(z, e)$ **fi**
R_4 : **if** $same(x, c)$ **and** $lessthan(w, 30)$ **then** $add(v, h)$ **fi**
R_5 : **if** $same(y, d)$ **and** $lessthan(w, 10)$ **then** $add(v, i)$ **fi**
R_6 : **if** $known(x)$ **and** $notsame(z, e)$ **then** $add(y, d)$ **fi**

The backward-chaining algorithm starts with the selection of the two production rules R_4 and R_5, since the goal variable v appears in their respective conclusions. The production rule R_4 will be the first one to be applied. Since there are no production rules concluding on the variable x occurring in the first condition of rule R_4, the user is asked to enter values for x. We suppose that the user answers by entering $x = \{c\}$. Evaluation of the first condition therefore yields the truth value *true*. It follows that R_4 succeeds, since the evaluation of the second condition, $lessthan(w, 30)$, yields the value *true* as well. The evaluation of the conclusion of the rule results in the addition of the fact $v = \{h\}$ to the fact set. Next, rule R_5 is applied. The first condition of this rule mentions the variable y. Since the variable y occurs in the conclusion of rule R_6, this rule is the next to be applied in order to obtain a value for y. The first condition of R_6 upon evaluation yields the truth value *true*. Evaluation of the second condition of rule R_6 ultimately results in a request to the user to supply values for the variable z, since the production rules R_1, R_2 and R_3 fail to infer values for it. When the user, for instance, provides the answer $z = \{i, j\}$, rule R_6 will succeed and the fact $y = \{d\}$ will be added to the fact set. We recall that rule R_6 was invoked during to the evaluation of the first condition of rule R_5. From the new fact $y = \{d\}$ we have that the first condition of rule R_5 yields the truth value *true* upon evaluation. Since the second condition is fulfilled as well, the action specified in the conclusion of the rule is executed: the value i is inserted into the fact concerning the variable v. We conclude that the following fact set F' has been obtained:

$$F' = \{x = \{c\}, y = \{d\}, z = \{i, j\}, v = \{h, i\}, w = 5\}$$

So, the solution arrived at is $\{v = \{h, i\}\}$.

An analysis of the *search space* generated by top-down inference can be instructive when developing optimizations of the algorithm. The search space of the top-down inference algorithm discussed in the foregoing is largely determined by the initial set of goals and the rule base, and has the form of a tree. We start the analysis by taking the backward-chaining strategy as a starting-point, and shall introduce several refinements to that algorithm. By means of the following example, we demonstrate how the search space is structured.

EXAMPLE 3.15 ___

Let $D = \{x_a^m, y^m, z_g^s, u_a^m, v^m, w_a^m\}$ be the domain declaration of a production system. Note that the single-valued variable z is the only goal variable. The fact set initially is empty. Consider the following set of production rules:

R_1 : **if** $same(w, a)$ **and** $same(x, b)$ **then** $add(v, c)$ **fi**
R_2 : **if** $same(w, d)$ **and** $same(v, c)$ **then** $add(y, e)$ **fi**
R_3 : **if** $same(v, c)$ **then** $add(z, k)$ **fi**
R_4 : **if** $same(x, j)$ **and** $same(y, e)$ **then** $add(z, h)$ **fi**
R_5 : **if** $same(u, f)$ **and** $same(x, g)$ **then** $add(z, i)$ **fi**

The inference engine starts with the construction of the conflict set: $\{R_3, R_4, R_5\}$. The rule R_3 is the first to be applied. The variable v mentioned in the first condition of rule R_3 occurs in the conclusion of rule R_1. This is the only rule in the new conflict set. So, rule R_1 is the next rule to be evaluated. Suppose that rule R_1 fails; as a consequence, no value will be inferred for v. The reader can easily verify that given this set of production rules the variables w, x, and u will be asked from the user. The search

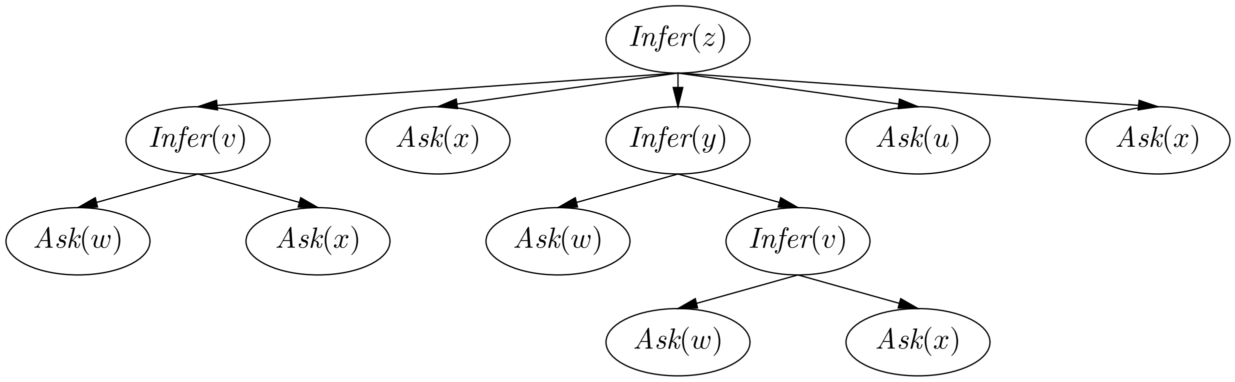


Figure 3.3: A search space generated using backward chaining.

space generated from the set of goals and the rules using backward chaining, takes the form of a tree, as shown in Figure 3.3. The label *Infer* indicates that the variable is inferred from the rule base; *Ask* indicates that the variable is asked from the user. Now note that the mere presence of a fact concerning a specific variable in the fact set, does not guarantee that the variable has actually been traced by exhaustively applying the production rules: the fact may have been entered into the fact set as a side-effect of the application of a production rule having more than one conclusion when tracing another variable. Therefore, the figure shows that the variable v has to be traced twice, even if a fact concerning v already occurs in the fact set. Furthermore, it indicates that several variables have to be asked from the user more than once.

In the preceding example, we showed that the process of tracing a specific variable may be repeated unnecessarily: it will be evident that optimization is required here. We introduce the notion of a 'traced' variable. A variable is marked as *traced* as soon as the process of tracing the variable has been performed, independent of whether it has yielded a value for the variable or not. We now modify the TraceValues procedure as follows:

```
procedure TraceValues(variable)
    Infer(variable);
    if not established(variable) and askable(variable) then
        Ask(variable)
    fi;
    traced(variable) ← true
end
```

Recall that in the foregoing, this procedure was invoked from the EvalConditions procedure. The last-mentioned procedure is now modified in such a way that the TraceValues procedure is invoked for a given variable only if the variable has not yet been marked as traced. This simple refinement has a dramatic effect on the structure of the search space, as is shown in the following example.

EXAMPLE 3.16

Consider once again the production rules from the preceding example, and also the search space for the set of goals and the rule base shown in Figure 3.3. If we exploit the refinement discussed in the foregoing and mark a variable as traced as soon as it has been traced exhaustively, then several inference steps and several questions to the user will have become superfluous. The resulting search space is considerably smaller than the one depicted in Figure 3.3, since now it suffices to represent each vertex only once. The resulting search space therefore has the form of a graph; it is depicted in Figure 3.4. Note that it indicates the dependencies between the variables during the inference.

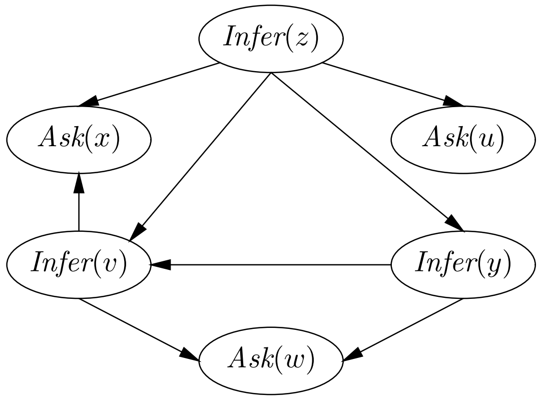


Figure 3.4: A search graph for backward chaining.

If a production rule contains two or more conclusions concerning different variables, then the rule may be applied more than once by the top-down inference algorithm. However, if we assume that production rules do not contain actions removing or modifying facts, then it suffices to apply such a rule only once: applying such a rule a second time cannot result in the addition of new facts to the fact set. To prevent a rule from being applied more than once, each rule will be marked as been *used* as soon as the inference algorithm applies it.

```
procedure Infer(variable)
    Select(rule-base, variable, selected-rules);
    foreach rule in selected-rules with not used(rule) do
        used(rule) ← true;
        Apply(rule)
    od
end
```

Moreover, marking a production rule before it is actually applied, has the further advantage of preventing infinite recursion in case of self-referencing rules.

In the foregoing we have introduced two refinements of the basic backward-chaining (and top-down) algorithm: the marking of a variable after its examination by the inference engine as 'traced', and the marking of a production rule as 'used' upon its application. The last refinement we pay attention to is the so-called *look-ahead facility*. This facility yields under certain circumstances a remarkable increase in efficiency, by pruning the search space in an effective way. The general structure of the search space, however, is not changed. Let us first study the following example illustrating the need for the look-ahead facility.

EXAMPLE 3.17

Consider the production rule:

```
if
    $same(x, a)$ and
    $same(y, b)$ and
    $notsame(z, c)$ and
    $lessthan(v, 40)$ and
    $greaterthan(w, 90)$
then
    $add(u, 6)$
fi
```

in which v and w are the only single-valued variables. Now, suppose that the fact $v = 80$ is present in the fact set. This single fact provides us with enough information to deduce that this rule will certainly fail due to its fourth condition. However, the backward-chaining algorithm as introduced in the foregoing, will detect the failure only when at least one (and possibly all three) of the variables x, y, and z from the first three conditions has been traced.

The look-ahead facility now amounts to examining all conditions from a selected production rule before it is actually applied. If a condition is encountered which already fails beforehand using the information from the fact set, then the entire rule will fail; the inference engine proceeds with the next rule. A possible algorithm for the look-ahead facility is shown below:

```
function LookAhead(rule)
    foreach condition in antecedent(rule) do
        var $\leftarrow$ variable(condition);
        if traced(var) then
            ExecPredicate(condition);
            if condition failed then
                return(false)
            fi
        fi
    od;
    return(true)
end
```

Note that in the look-ahead facility only conditions specifying a traced variable are examined; it just skips the other conditions. If look ahead returns with success, then the top-down inference algorithm continues as usual. This function LookAhead is called from the procedure

Infer just before calling Apply. The procedure Apply then is invoked only if LookAhead has succeeded.

In the following diagram, the most important procedure and function calls are shown, indicating the level at which they are introduced in the top-down inference algorithm:

```
TraceValues
   Infer
      Select
      LookAhead
         ExecPredicate
      Apply
         EvalConditions
            TraceValues
            ExecPredicate
         EvalConclusions
            ExecAction
   Ask
```

This diagram depicts once more the indirect recursive call to the procedure TraceValues from EvalConditions.

Note that only relatively slight modifications of the discussed algorithm are needed to extend it to an object-attribute-value representation. We only have to add a procedure for tracing the *goal attributes* of an object, a notion similar to the notion of a goal variable:

```
procedure Activate(object)
   foreach attr in attributes(object) do
      if goal(attr) then
         TraceValues(object, attr)
      fi
   od
end
```

The other procedures only need slight alteration by adding the object as an extra argument to the procedures, and by taking for a variable an attribute of the given object.

3.2.2 Bottom-up inference and production rules

Broadly speaking, bottom-up inference with production rules differs from top-down inference only by being controlled by the fact set instead of by goals and subgoals. As in top-down inference, each time a set of relevant production rules is selected from a rule base; the resulting set of applicable production rules in bottom-up inference again is called the conflict set. Recall that in top-down inference, only production rules having a particular (sub)goal variable in one of their conclusions were included in the conflict set. In bottom-up inference, however, a production rule is entered into the conflict set if its conditions are fulfilled using the information from the given fact set. In applying the rules from the conflict set, there is also a difference between the two forms of inference. In top-down inference all production rules from the conflict set are applied exhaustively. In bottom-up inference, however, generally only one of them is applied. This difference arises from the action *remove* being applied frequently in production systems employing bottom-up inference, in contrast with top-down inference systems. It will be evident that the evaluation of this action may have as an effect that certain conditions, which were true before evaluation of that specific action took place, do not longer hold after its evaluation. As a consequence, all rules in the conflict set have to be reconsidered, since some may specify conditions which fail upon evaluation using the altered fact set. Furthermore, the changes in the fact set may render other rules being successful, which should then be added to the conflict set. Therefore, after applying a single rule from a specific conflict set, a new conflict set is selected. In practice, however, many of the rules previously present in the conflict set will appear again in the new conflict set in the next inference cycle. The entire process is repeated again and again, until some predefined termination criterion is met; a frequently employed criterion is the emptiness of the set of applicable rules. The inference is started just by the presence of initial facts in the fact set. The general approach of bottom-up inference in a production system is described in the following procedure:

```
procedure Infer(rule-base, fact-set)
    rules ← Select(rule-base, fact-set);
    while rules ≠ ∅ do
        rule ← ResolveConflicts(rules);
        Apply(rule);
        rules ← Select(rule-base, fact-set)
    od
end
```

The function Select is applied for selecting the applicable production rules from the rule base. The function ResolveConflicts subsequently chooses from the resulting conflict set a single rule for application: this function implements a conflict-resolution strategy, several of which will be discussed below. The selected production rule is then applied by means of a call to the procedure Apply. In bottom-up inference, the procedure Apply just evaluates the actions of the conclusions specified in the consequent of the rule.

The selection of the relevant production rules from the rule base is described in the following function Select. Again, for ease of exposition, we have assumed that the antecedent of a production rule only comprises a conjunction of conditions:

```
function Select(rule-base, fact-set)
    selected-rules ← ∅;
    foreach rule in rule-base do
        failed ← false;
        foreach cond in antecedent(rule) while not failed do
            EvalCondition(cond, fact-set, failed)
        od;
        if not failed then
            selected-rules ← selected-rules ∪ {rule}
        fi
    od;
    return(selected-rules)
end
```

The procedure EvalCondition evaluates a single condition of a production rule. It returns failure if the variable mentioned in the condition does not occur in the fact set, or if the specified predicate upon evaluation yields the truth value *false*; otherwise the procedure returns success. Note that this evaluation differs from the evaluation described for top-down inference, since here the evaluation of a condition does not lead to the generation of a new subgoal.

The basic bottom-up inference algorithm is very simple. However, much more is still to be said about incorporating control strategies into the basic bottom-up inference scheme. The order in which the rule base is traversed nor the order in which the conditions and conclusions of a selected rule are evaluated has been fixed in the procedure Select. As in Section 3.2.1, we again call the non-fixed order in which production rules are selected and applied, nondeterminism of the first kind, and the non-fixed order in which conditions and conclusions are evaluated, nondeterminism of the second kind. Nondeterminism of the first kind is again resolved by means of a conflict-resolution strategy. These strategies are much more often applied in systems with bottom-up inference than in systems using top-down inference. If rules, and conditions and conclusions of rules, are evaluated in the order in which they have been specified we speak of *forward chaining*; this is the simplest possible form of conflict-resolution. So, in forward chaining the first successful production rule encountered will be selected for application. Note that the choice of the order in which the conditions of the rules from the rule base are evaluated in bottom-up inference has no effect on the resulting behaviour of the system. Only the order in which the rules are applied, and the order in which the conclusions of the rules are evaluated, are of importance.

Many conflict-resolution strategies have been developed for bottom-up inference; here we only discuss three more of them in addition to the earlier mentioned forward chaining. The main reason for augmenting the basic inference algorithm with a conflict-resolution strategy which is more sophisticated than simple forward chaining, is to obtain a more context-sensitive and problem-directed reasoning behaviour, that is, to better control the inference. Since bottom-up inference lacks the 'intelligent' goal-directed nature of top-down inference, conflict-resolution strategies evidently are much more important in bottom-up inference than in top-down inference. Possible conflict resolution strategies differing from forward chaining are:

- conflict resolution by *prioritization*, which for selecting a rule for application uses priorities of the production rules which have been indicated explicitly by the knowledge engineer;
- conflict resolution by *specificity*, which causes the system to prefer more strongly stated production rules over weaker ones;
- conflict resolution by *recency*, which uses the most recently derived facts in selecting a production rule for application, thus causing the system to pursue a single line of reasoning.

Conflict resolution by production rule prioritization has the same advantages as forward chaining: it is easy to implement and use, while being effective in many applications. However, an obvious disadvantage of this strategy is the burden it places on the knowledge engineer who has to impose an ordering on the production rules from the rule base explicitly.

Conflict resolution by specificity is based on some measure of specificity for production rules, such as for example the number of tests specified in the conditions of the rules. A production rule R_1 is considered to be more specific than a production rule R_2 if R_1 contains at least the same conditions as R_2.

EXAMPLE 3.18

Consider the following two production rules:

R_1 : **if** same(programming-language, symbol-manipulation)
then add(language, AI) **fi**

R_2 : **if** same(programming-language, symbol-manipulation)
and same(syntax, parentheses)
then add(language, Lisp) **fi**

Production rule R_2 is more specific than production rule R_1, since it contains the condition from R_1 as well as some additional conditions. Furthermore, the conclusion of R_2 is more specific than the one from R_1. It will be obvious that success of R_2 will yield a stronger result than success of rule R_1. A specificity strategy will therefore choose R_2 from the conflict set R_1, R_2 for application.

The use of a specificity strategy increases the extensibility of the rule base: a rule base can easily be enlarged by adding new, more specific rules to it without our having to worry too much about older rules, since more specific production rules prevail over the more general ones. Note that most humans exhibit a similar behaviour. For example, when a person encounters his friend in the street, he will not be inclined to think that this other person is a mammal, but he will think instead that it is his friend John: he just applies the most specific knowledge he has.

The last conflict-resolution strategy we pay attention to, the recency strategy, is undoubtedly the most complicated of the ones we have mentioned. This conflict-resolution strategy requires that each fact in the fact set is supplemented with a so-called time tag, a unique number indicating the 'time' the fact was derived. In the following definition, the notion of a fact is redefined for the case of bottom-up inference using the recency strategy. We will only deal with single-valued variables; the extension to multi-valued ones is straightforward.

Definition 3.4 *A* fact *is an expression of the following form:*

$$t : x^s = c$$

where $t \in \mathbb{N}$ is a time tag *which uniquely identifies the fact; x^s is a single-valued variable, and c is a constant. A* fact set *has the following form:*

$$\{t_1 : x_1^s = c_1, \ldots, t_n : x_m^s = c_m\}$$

Constants and variables may now occur more than once in the fact set. However, a time-tag variable pair is unique. Each fact added to the fact set as a result of applying a production rule, is assigned a new time tag $t + 1$ where t is the last assigned one.

EXAMPLE 3.19

Consider the following fact set:

$$\{1 : x = a, 2 : x = b, 3 : y = c, 4 : z = d\}$$

The variable x occurs twice amongst these facts, with different time tags. This should be interpreted as follows: at time 1 the variable x has taken the value a, and at time 2 it has obtained the value b. Therefore, x has two values, one at time 1 and another one at time 2.

There are various ways in which time tags may be interpreted in the representation of facts. Time tags for example may be taken to monitor progress in time of some parameter.

EXAMPLE 3.20

Consider the following fact set:

$$\{1 : \mathit{temp} = 36.2, 2 : \mathit{temp} = 37, 3 : \mathit{temp} = 38\}$$

Herein, time tags are used to indicate the change of the body temperature of some person in time; each time tag for example indicates a day.

We recall that here we introduced time tags in order to enable conflict resolution by recency; this is the main usage of time tags. After the applicable production rules have been selected from the rule base, it is possible to order the conflict set using the time tags associated with the individual facts from the fact set. With each rule in the conflict set a sequence of time tags is associated, where each time tag originates from a fact matching with a condition of the specific rule. These time tags are then sorted in decreasing order. Each thus obtained sequence of time tags is padded with as many as zero time tags as required to make all sequences of equal length. This way, the production rules in the conflict set may be compared to each other.

EXAMPLE 3.21

Consider the following fact set:

$$\{1 : x = a, 2 : x = b, 3 : y = c, 4 : z = d, 5 : w = e, 6 : z = f\}$$

Now, suppose that the conflict set consists of the following three production rules:

R_1: **if** $\mathit{same}(z, f)$ **then** $\mathit{add}(x, e)$ **fi**
R_2: **if** $\mathit{same}(x, b)$ **and** $\mathit{same}(z, d)$ **then** $\mathit{add}(y, f)$ **fi**
R_3: **if** $\mathit{same}(x, a)$ **and** $\mathit{same}(y, c)$ **and** $\mathit{same}(w, e)$ **then** $\mathit{add}(x, d)$ **fi**

Rule R_3 has the largest number of conditions, namely three. So, with each rule we associate a sequence of time tags having length three:

R_1:	6	0	0
R_2:	4	2	0
R_3:	5	3	1

The production rules in the conflict set are now ordered according to the lexicographical order of their associated sequences of time tags: in the ordering, a rule R_1 precedes a rules R_2 if the sequence of time tags associated with R_1, read from left to right, is larger than the one associated with rule R_2. The order relation between members of the conflict set is denoted by the symbol $\geq$. The relation $\geq$ is a total ordering, and has therefore the following four properties:

- *Reflexivity:* for each production rule R we have $R \geq R$;
- *Transitivity:* for each three production rules R_1, R_2, and R_3, satisfying $R_1 \geq R_2$ and $R_2 \geq R_3$, we have that $R_1 \geq R_3$;
- *Anti-symmetry:* for each pair of rules satisfying $R_1 \geq R_2$ and $R_2 \geq R_1$, we have that $R_1 = R_2$;
- *Totality:* for each pair of production rules R_1 and R_2, we have either $R_1 \geq R_2$ or $R_2 \geq R_1$.

EXAMPLE 3.22

Consider the following sequences of time tags, belonging to four production rules R_1, R_2, R_3 and R_4, respectively:

R_1:	6	0	0
R_2:	4	3	2
R_3:	5	3	1
R_4:	6	1	0

For these rules we have that $R_4 \geq R_1 \geq R_3 \geq R_2$.

This ordering of the rules from a conflict set enables us to give an algorithm for conflict resolution based on recency; it is described in the following function ResolveConflicts:

```
function ResolveConflicts(rules)
    if rules = ∅ then return(∅)
    else
        r ← Max-Time-tag-Subset(rules);
        if r is singleton then return(r)
        else
            result ← ResolveConflicts(r)
        fi;
        if result = ∅ then return(first(r))
        else
            return(result)
        fi
    fi
end
```

The function Max-Time-tag-Subset called from ResolveConflicts selects from the conflict set rules the rules with the highest time tag. If the resulting set r contains more than one element, then, after the earlier examined time tags have been skipped, the function ResolveConflicts is called recursively for this set r. Note that contrary to what has been described before, the conflict set is not ordered entirely before it is examined: each time only a subset of the rules relevant for conflict resolution is selected. The presented algorithm does not always yield a single production rule, since it is possible to have two or more (different) production rules, having the same sequence of time tags. In this case, on arbitrary grounds the first specified production rule is returned as a result.

As has been discussed above, a variable specified in a condition of a production rule may have more than one occurrence in the fact set, although with different time tags. As a consequence, a condition may match with more than one fact. So, a production rule may be applied more than once, using different facts.

EXAMPLE 3.23

Consider the following fact set:

$$F = \{1 : x = a\}$$

and the conflict set consisting of the following two production rules:

R_1: **if** $\mathit{same}(x, a)$ **then** $\mathit{add}(y, b)$ **fi**
R_2: **if** $\mathit{same}(y, b)$ **then** $\mathit{add}(y, b)$ **fi**

Then, the application of rule R_1 in the first inference step results in the following modified fact set:

$$F' = \{1 : x = a, 2 : y = b\}$$

Subsequent application of rule R_2 yields the following fact set:

$$F'' = \{1 : x = a, 2 : y = b, 3 : y = b\}$$

Rule R_2 can now be applied again. In this example, the inference will not terminate; rule R_2 will be applied forever.

In the preceding example, we have shown that a production rule may be applied more than once, using different facts. It is therefore necessary to specify in the conflict set all possible *applications* of a rule, instead of the rule itself. For this purpose, we introduce the notion of a rule instance.

Definition 3.5 *Let F be a fact set, and R a production rule. Let $M \subseteq F$ be a subset of facts, such that each element $f \in M$ matches with a condition of R, and each condition of the production rule R matches with an element from M. Then, the pair (R, M) is called a* rule instance*.*

In other words, a rule instance consists of a production rule and the facts matching with its conditions. It will be evident that although production rules may be applied several times, it is undesirable that rule instances are applied more than once. Note that the conflict set should now be taken as a set of rule instances. The basic algorithm for bottom-up inference discussed in the foregoing has to be altered for dealing with such rule instances. Recall that so far, we have treated four procedures which together constitute the bottom-up inference algorithm:

- the procedure Infer, which described the global inference process;
- the function Select, which was used for the selection of applicable rules from the rule base;
- the function ResolveConflicts, which specified the conflict-resolution method;
- the procedure Apply, which applied the once selected production rule.

First, we reconsider the procedure Infer. Here, we have to record the rule instances which have been applied, to prevent rule instances from being applied more than once:

```
procedure Infer(fact-set, rule-base)
    applied-instances $\leftarrow \varnothing$;
    instances $\leftarrow$ Select(rule-base, applied-instances, fact-set);
    while instances $\neq \varnothing$ do
        instance $\leftarrow$ ResolveConflicts(instances);
        Apply(instance);
        applied-instances $\leftarrow$ applied-instances $\cup$ {instance};
        instances $\leftarrow$ Select(rule-base, applied-instances, fact-set)
    od
end
```

The function Select now has to generate rule instances from the production rules in the rule base instead of the production rules themselves:

```
function Select(rule-base, applied, fact-set)
    selected-instances $\leftarrow \varnothing$;
    foreach rule in rule-base do
        failed $\leftarrow$ false;
        rule-instances $\leftarrow$ (rule, $\varnothing$);
        foreach cond in antecedent(rule) while not failed do
            ModifyInstances(cond, rule-instances, fact-set, failed)
        od;
        if not failed then
            selected-instances $\leftarrow$ selected-instances $\cup$
                                    (rule-instances $\setminus$ applied)
        fi
    od;
    return(selected-instances)
end
```

Note that the second argument of Select now contains a set of rule instances. The procedure ModifyInstances called from Select, evaluates a given condition cond using the fact set. Each different matching fact gives rise to the creation of a new rule instance. Of course, if a condition fails then no rule instance will be created. Since the rule instances are built recursively, evaluation of subsequent conditions of a production rule may lead to discarding rule instances under construction from the set rule-instances:

```
procedure ModifyInstances(condition, rule-instances, fact-set, failed)
    relevant-facts ← EvalCondition(condition, fact-set);
    failed ← relevant-facts = ∅;
    new-instances ← ∅;
    if not failed then
      foreach fact in relevant-facts do
        foreach rule-inst in rule-instances do
          new-instances ← new-instances ∪ Add(rule-inst, fact)
        od
      od
    fi;
    rule-instances ← new-instances
end
```

After the conflict set has been created, the next step in Infer is to select a single rule instance from it by conflict resolution. This is achieved by means of the procedure ResolveConflicts which already has been discussed. Finally, the procedure Apply is called to evaluate the conclusions of the selected rule instance. Recall that evaluation of the action *add* adds a new fact to the fact set, which is assigned a new time tag. Evaluation of the action *remove* deletes a fact from the fact set; the fact to be deleted is selected either by explicitly referring to its time tag, or simply by matching.

We conclude this section with an example.

EXAMPLE 3.24

Consider the following fact set F:

$$F = \{1 : x = a, 2 : x = b, 3 : y = 4\}$$

Furthermore, let us have the following set of production rules:

R_1: **if** $\mathit{same}(x, a)$ **and** $\mathit{same}(x, b)$ **then** $\mathit{add}(z, e)$ **fi**
R_2: **if** $\mathit{same}(z, e)$ **and** $\mathit{same}(w, g)$ **then** $\mathit{add}(z, f)$ **fi**
R_3: **if** $\mathit{lessthan}(y, 10)$ **and** $\mathit{same}(x, a)$ **or** $\mathit{same}(x, b)$ **then** $\mathit{add}(w, g)$ **fi**

The given fact set F gives rise to the creation of the following rule instances, together constituting the conflict set:

$(R_1, \{1 : x = a, 2 : x = b\})$
$(R_3, \{3 : y = 4, 1 : x = a\})$
$(R_3, \{3 : y = 4, 2 : x = b\})$

Note that two rule instances of R_3 have been created. Using the recency conflict-resolution strategy, the second instance of R_3 is selected for evaluation, since the time tag of its second matching fact is larger than the time tag of the second matching fact of the first instance of R_3. Evaluation of the instance $(R_3, \{3 : y = 4, 2 : x = b\})$ causes the fact $4 : w = g$ to be added to the fact set, resulting in:

$$F' = \{1 : x = a, 2 : x = b, 3 : y = 4, 4 : w = g\}$$

The inference is now repeated; the instance $(R_3, \{3 : y = 4, 2 : x = b\})$ however is no longer applicable.

The algorithm discussed in this section provides a more or less complete description of the bottom-up inference method. However, an inference engine implementing this algorithm will be quite inefficient, since at every inference step all instances are created all over again. A first step towards improving the efficiency of the algorithm is to save the instances between two consecutive inference steps. Such an algorithm already exists, it is called the *rete algorithm*, and has been developed by C.L. Forgy as part of the system OPS5. We shall return to this rete algorithm and OPS5 in Chapter 8.

3.3 Production rules as a representation formalism

In this chapter we have discussed several forms of knowledge representation and inference used in production systems. Various attempts in the past in using production rules for building intelligent systems have proven the production system approach to be a flexible one, and is suitable for many problem areas. In fact, many of the intelligent systems mentioned in chapter 1 are examples of rule-based systems. In addition, several large intelligent systems have been and still are being developed using the techniques discussed in this chapter. However, some disadvantages and restrictions of the formalism have also been recognized:

- Descriptive knowledge cannot be represented in the formalism in a natural way. An example of descriptive knowledge has been given in chapter 1 where we described the cardiovascular system. We shall see in the following chapter that the frame formalism is much more suitable for representing this type of knowledge.

- The different types of knowledge encountered in a problem area, such as problem-dependent knowledge, problem-independent knowledge, and knowledge used for exerting control on the inference process (often called meta-knowledge) have to be expressed using one and the same formalism and therefore cannot be distinguished explicitly.

- The production rule formalism has a strong operational flavour. As a consequence, some knowledge of the underlying execution model of the inference engine is required for adequately representing a problem domain in a knowledge base. Compare this situation with the one in logic, where no knowledge concerning inference is required for specifying a correct knowledge base; familiarity with the declarative semantics of logic suffices in this case.

- A more involved application generally leads to a large rule base, which is difficult to develop and maintain. So, for developing large applications the necessity of partitioning a large rule base into smaller modules arises. However, the production rule formalism offers no direct means for explicitly indicating and exploiting such a modularization.

Some of these problems may be solved by combining production rules with some other formalism, such as for example a frame formalism, or with other programming paradigms such as object-oriented programming. Chapter 8 discusses several solutions that have been proposed and incorporated in actual systems.

Exercises

(3.1) Consider the knowledge base containing the following domain declaration

$$D = \{x_a^s, y^s, z_a^s, w_g^m\}$$

and the following rule base:

$\{R_1$: **if** *lessthan*$(z, 20)$ **and** *notknown*(y) **then** *add*(w, b) **fi**,
R_2: **if** *same*(x, c) **and** *known*(y) **then** *add*(w, d) **fi**,
R_3: **if** *notsame*(x, b) **and** *greaterthan*$(z, 100)$ **then** *add*(y, f) **fi**$\}$

The variables x_a^s, y^s and z_a^s are single-valued, and the variable w_g^m is multi-valued. As can be seen, w_g^m is a goal variable, and the variables x_a^s and z_a^s are askable. Furthermore, let the following fact set be given:

$$F = \{x = c, z = 5\}$$

(a) Determine the fact set F' which results from applying backward chaining on this set of production rules. Which production rules will have succeeded, and which ones have failed?

(b) Suppose that the following self-referencing production rule is added to the three rules listed above:

R_4: **if** *notknown*(y) **then** *add*(y, A) **fi**

Again we employ backward chaining, starting with the same initial fact set F as given above. Which fact set F'' do we now obtain?

(3.2) Has the choice of the conflict-resolution strategy employed in top-down inference with the look-ahead facility, any effect on the values inferred for a multi-valued goal variable? Motivate your answer. Now answer the same question for an arbitrary (subgoal) variable.

(3.3) Consider the following knowledge base, containing the domain declaration

$$D = \{x^s, y^s, z^s, u^s\}$$

and the rule base:

$\{R_1$: **if** *same(x,a)* **and** *known*(y) **then** *add*(y, b) **fi**,
R_2: **if** *same*(x, c) **and** *lessthan*$(z, 15)$ **then** *add*(u, d) **fi**,
R_3: **if** *same*(y, b) **and** *lessthan*$(z, 5)$ **then** *add*(u, f) **fi**$\}$

Furthermore, consider the following fact set:

$$F = \{1 : x = a, 2 : y = b, 3 : z = 10, 4 : x = c\}$$

Bottom-up inference is employed to derive new facts from the given rule base and the facts present in the fact set.

(a) Give all rule instances created by matching the initial fact set F and the rule base given above. This set of instances is the conflict set.

(b) Order the conflict set obtained in (a), using conflict resolution by recency. Which one of the rule instances will then be selected for application?

(c) Give the fact set which results after evaluation of the rule instance chosen in (b). Will the inference eventually terminate? Motivate your answer by giving the successive changes which take place in the fact set.

(3.4) Consider the following knowledge base, containing the domain declaration

$$D = \{x^s, y^s, z^s\}$$

and the rule base:

$\{R_1$: **if** $\mathit{same}(x, a)$ **and** $\mathit{equal}(y, 10)$ **then** $\mathit{add}(z, b)$ **fi**,
R_2: **if** $\mathit{same}(z, c)$ **and** $\mathit{lessthan}(y, 20)$ **then** $\mathit{add}(x, a)$ **fi**$\}$

Furthermore, consider the following fact set:

$$F = \{1 : x = A, 2 : z = C, 3 : y = 10, 4 : z = C\}$$

Bottom-up inference is employed for inferring new facts from the given rule base and the fact set.

(a) Which rule instances will be created in the first inference step, and which one of these will be selected for application if we apply conflict resolution by recency? Give the new fact set obtained after evaluation of the chosen rule instance. Will the inference eventually terminate?

(b) Suppose that we add the following production rule to the ones shown above, before consulting the knowledge base.

R_3: **if** $\mathit{same}(z, b)$ **then** $\mathit{add}(z, c)$ **fi**

We start with the same inititial fact set F as in (a). Give the fact set that eventually results. Will the inference terminate? Explain your answer by comparing the results with those of (a).

(3.5) Recall that two types of nondeterminism are distinguished in production systems due to the need to specify: (1) the order in which applicable rules are selected from the rule base, and (2) the order in which conditions and conclusions are evaluated. If we take a particular conflict-resolution strategy in bottom-up inference, which choice(s) for resolving nondeterminism do(es) influence the behaviour of the system?

Chapter 4

Frames and Inheritance

Representing knowledge in graph-like structures has a rich tradition in philosophy and psychology. At the end of the nineteenth century, the philosopher Charles S. Peirce used a graph-like notation for the representation of logical sentences. This approach to representing human knowledge has been further pursued since by many researchers, yielding explicit psychological models of human memory and intellectual behaviour. In particular the area of natural language processing has contributed much to the research on the representation of information in graph-like structures, there called *semantic nets* or *associative nets*; in fact, the earliest use of graph-based representations in computers was for machine translation. In the early 1960s, Ross Quillian for example used the semantic net formalism for representing meanings of English words in terms of associative links to other words, yielding a dictionary-like representation; he developed a program for finding relationships between words by traversing the net. Through this work, Ross Quillian has given a major impetus to the research on graph-based representations and their use in AI systems; he is generally credited with the development of the semantic net in its original form.

For handling more complicated problem domains and for dealing with more sophisticated forms of inference, the semantic net formalism as devised by Ross Quillian soon proved to be too limited. Much of the later work on semantic nets therefore has been directed towards more structured formalisms, again mostly for natural language processing. Semantic nets have seldom been used for building expert systems. Nevertheless, we shall briefly discuss some characteristics of the formalism, since the semantic net is often viewed as a precursor of the frame formalism, which is much more frequently applied within expert systems.

The basic idea underlying the notion of frames has already been posed at the beginning of this century by the psychologist Otto Selz. He considered human problem solving as the process of filling in the gaps of partially completed descriptions. The present notion of frames was introduced half-way the 1970s by Marvin Minsky for exerting semantic control in a pattern-recognition application. Since its introduction, however, the frame formalism has been employed in several other kinds of knowledge-based systems as well. The general idea of a frame-based system is that all knowledge concerning individuals or classes of individuals including their interrelationships, is stored in a complex entity of representation, called a *frame*. Instead of the term frame, the terms *unit*, *object*, and *concept* are also often used in literature. A set of frames representing the knowledge in a domain of interest, is organized hierarchically in what is called a *taxonomy*. Such a taxonomy forms the basis of a method of automated reasoning called *inheritance*. The frame formalism and its associated inheritance are the primary topics of this chapter. To prepare for a thorough treatment of these subjects, we shall first discuss the semantic net formalism briefly in Section 4.1.

4.1 Semantic Nets

A *semantic net* is usually depicted as a labelled directed graph, consisting of vertices and labelled arcs between vertices; such a graph is sometimes further restricted by requiring its being acyclic. Several disciplines have influenced the original idea of a semantic net as it was introduced in the 1960s: each discipline has brought its own interpretation of the vertices and arcs, and each discipline has adapted the notion of the semantic net in certain ways to arrive at a more structured formalism suitable for its own purposes. As a consequence, there is hardly any consensus as to what a semantic net is, nor is there any consensus as to what meaning should be ascribed to the basic elements of such a semantic net. Since the semantic net formalism is seldom used in expert systems, we will introduce it in a simple form, just to give the reader an idea about what graph-based representations are like.

4.1.1 Vertices and labelled arcs

We have mentioned before that a semantic net is usually depicted as a labelled, directed graph. Each vertex in the graphical representation of a semantic net is taken to represent a *concept*. The arcs of the graph represent binary relations between concepts. Let us give some informal examples of how knowledge is represented in a semantic net.

EXAMPLE 4.1

Consider the following statement concerning the human body:

'The heart is part of the cardiovascular system'

This statement comprises two concepts: the concept 'heart' and the concept 'cardiovascular system'. These concepts are related in the sense that the first concept, the 'heart', forms an anatomical part of the second concept, the 'cardiovascular system'. This knowledge is represented by means of the graph shown in Figure 4.1. The concepts are depicted by ellipses, labelled *heart* and *cardiovascular system*; the relation between the concepts is represented by means of an arc labelled *part-of*.

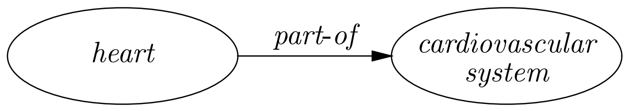


Figure 4.1: A graphical representation of a semantic net.

EXAMPLE 4.2

In the semantic net depicted in Figure 4.2, two different kinds of relation are used in representing information concerning the cardiovascular system of the human body: the 'part-of' relation and the 'is-a' relation.

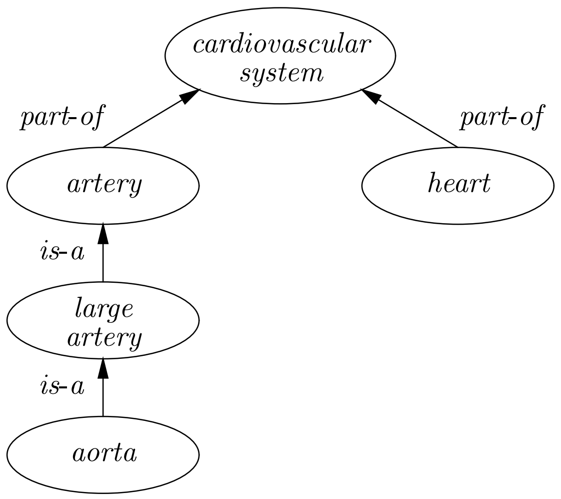


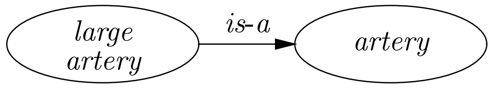
Figure 4.2: Some information concerning the cardiovascular system in a semantic net.

In the preceding example we encountered the is-a relation. This is a quite common relation between concepts. It reflects the two different senses in which a concept can be used; in this book, the term concept is used to denote either an *individual object* or a *class of objects*. The is-a relation may be used as follows:

- To express that a class of objects is a subclass of another class of objects, such as in the statement

  'A large artery is an artery'.

  This statement is depicted in a semantic net as follows:

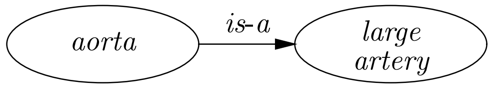
  

  In this case, the is-a part of the statement defines a *set inclusion relation*.

- To express that a specific object is a member of a certain class of objects, such as in the statement

  'The aorta is a large artery'.

  This statement is depicted as follows:

  Here, the is-a part of the statement defines a *membership relation* between an element and a set of elements.

In the early semantic net formalism, no explicit distinction was made between the different uses of the is-a relation, called the *is-a link* in semantic net terminology: individual objects

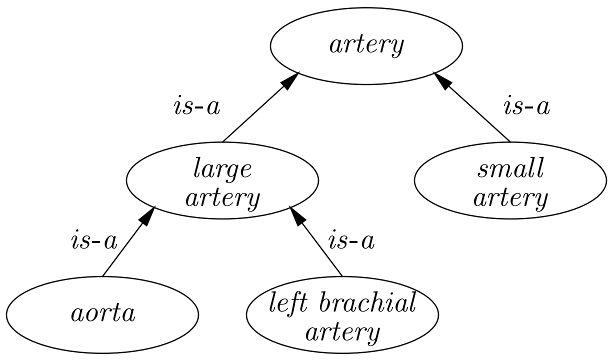


Figure 4.3: Counting specific objects.

and classes of objects were handled identically. The following example illustrates that this may lead to problems.

EXAMPLE 4.3

In Figure 4.3 some information concerning the arteries of the cardiovascular system is represented. Now consider a knowledge-based system comprising the information as shown. Suppose that we ask this system on how many specific arteries information is available. If we assume that the system 'knows' that information on individual objects is contained in the leaves of the net, the system will answer 3: the *aorta*, the *left brachial artery*, and the *small artery*. The system is not able to distinguish between the *small artery* representing a class of objects, and the individual objects *aorta* and *left brachial artery*.

The preceding example gives us a valid reason for distinguishing between different types of is-a link. From now on, we distinguish between the *subset-of link* and the *member-of link*.

Before proceeding, we define the notion of a semantic net more formally.

Definition 4.1 *A* semantic net *S is a labelled graph $S = (V(S), A(S), \lambda)$ where $V(S)$ is the set of* vertices *of S and $A(S) \subseteq V(S) \times V(S)$ is the set of* arcs *of S; λ is the* labelling function *$\lambda : A(S) \rightarrow L(S)$ associated with S where $L(S)$ is the set of* arc labels.

EXAMPLE 4.4

Consider the semantic net from Figure 4.1 once more. This net is defined by $S = (V(S), A(S), \lambda)$ where

$$V(S) = \{heart, cardiovasculari\text{-}system\}$$
$$A(S) = \{(heart, cardiovascular\text{-}system)\}$$
$$\lambda(heart, cardiovascular\text{-}system) = part\text{-}of$$

In the preceding, we have defined a semantic net as a mere syntactical object: it has no meaning as yet. In order to assign a meaning to a semantic net, we have to define a proper interpretation for it. Note the analogy with a logical formula being a syntactical object and its interpretation (see the Sections 2.1 and 2.2).

We start by giving an example of a possible interpretation for the subset-of link. We assign to the relation defined by the subset-of links the meaning of the usual set inclusion relation $\subseteq$. The relation $\subseteq$ has the following properties:

- *Reflexivity*: for each X, we have $X \subseteq X$.
- *Anti-symmetry*: for each X, Y, if $X \subseteq Y$ and $Y \subseteq X$, then $X = Y$.
- *Transitivity*: for each X, Y, Z, if $X \subseteq Y$ and $Y \subseteq Z$, then $X \subseteq Z$.

Any binary relation having these properties is called a *partial order*. With each vertex $x \in V(S)$ taking part in a subset-of link — note that from this observation we have that the vertex represents a class of objects — we associate a set of elements $I(x)$ from a (semantic) domain of discourse D, that is, $I(x) \subseteq D$. We now may interpret $\lambda(x, y) = \textit{subset-of}$ as $I(x) \subseteq I(y)$. From the reflexivity of the set inclusion relation we have that we may add to or delete from a semantic net arcs of the form (x, x) for which $\lambda(x, x) = \textit{subset-of}$:

$$\begin{tikzcd} x \arrow[loop above, "\textit{subset-of}"] \end{tikzcd}$$

This is called a *trivial cycle*. From the transitivity of the set inclusion relation it furthermore follows that if we have $\lambda(x, y) = \textit{subset-of}$ and $\lambda(y, z) = \textit{subset-of}$ in a specific net, then we may add $\lambda(x, z) = \textit{subset-of}$ to the net without changing its meaning: the two nets

$$\begin{tikzcd} x \arrow[r, "\textit{subset-of}"] & y \arrow[r, "\textit{subset-of}"] & z \end{tikzcd}$$

and

$$\begin{tikzcd} x \arrow[r, "\textit{subset-of}"] \arrow[rr, bend right, "\textit{subset-of}"'] & y \arrow[r, "\textit{subset-of}"] & z \end{tikzcd}$$

therefore express the same information.

Similar to the interpretation of the subset-of link, vertices $u \in V(S)$ taking part in the left-hand side of a member-of link — from this we have that u represents an individual object — have associated an element $I(u)$ from the domain D, that is, $I(u) \in D$. The relation defined by the member-of links now is assigned the meaning of the usual membership relation $\in$; that is, we interpret $\lambda(u, v) = \textit{member-of}$ as $I(u) \in I(v)$.

It will be evident that for a semantic net to have a neat semantics we have to define a proper interpretation for each type of link used in the net. Especially if no restrictions have been imposed on the types of links, this will be a cumbersome endeavour. It is no wonder therefore that since the introduction of the semantic net idea, researchers have sought after more restricted special-purpose net formalisms. Here, we do not pursue the subject of the declarative semantics of a semantic net any further.

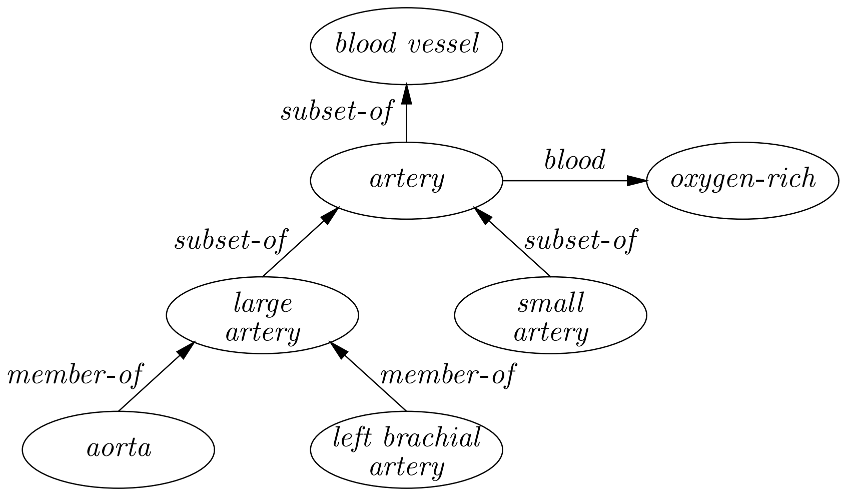


Figure 4.4: Inheritance.

4.1.2 Inheritance

The subset-of and member-of links of a semantic net may be exploited to derive new information from it, that is, they may be used as the basis for an inference engine. We illustrate the use of these links in reasoning with the help of an example.

EXAMPLE 4.5

Consider the semantic net shown in Figure 4.4. Among others, the following two statements are represented in the net:

'A large artery is an artery'
'An artery is a blood vessel'

From these two statements we may derive the statement

'A large artery is a blood vessel'

exploiting the transitivity of the relation defined by the subset-of links. Furthermore, the statement

'The aorta is an artery'

can be derived from the net using the semantics of both the member-of and subset-of link.

Exploiting the semantics of the member-of and subset-of links in the manner discussed informally in the preceding example forms the basis of a reasoning mechanism called (*property*) *inheritance*: a concept *inherits* the properties of the concepts 'higher' in the net through these member-of and subset-of links. The general idea is demonstrated in the following example.

EXAMPLE 4.6

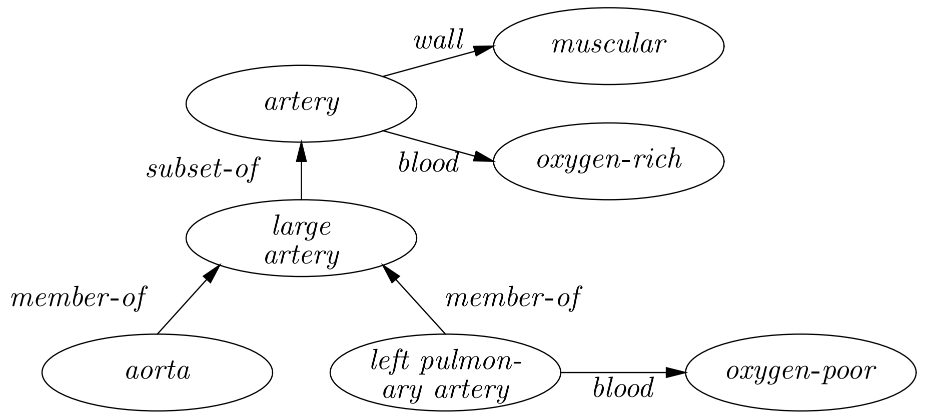


Figure 4.5: An exception to a general property.

Consider the semantic net shown in Figure 4.4 once more. Using property inheritance, we may derive from it the following statement:

'The aorta contains oxygen-rich blood'

The concept 'aorta' has inherited the property 'contains oxygen-rich blood' from the concept 'artery' which is found higher in the net.

In Section 4.3 we shall discuss the principle of inheritance more formally.

The semantic net is a natural formalism for expressing knowledge in which the basic concepts are organized in a hierarchical manner. Several problems, however, arise from the rigidity of the principle of inheritance as introduced above. We give two examples to illustrate some of these problems.

EXAMPLE 4.7

Consider Figure 4.5, again showing some information concerning the arteries. Among other information, it has been specified that arteries in general have muscular walls and transport oxygen-rich blood. An exception to the latter property of arteries is for example the left pulmonary artery which is an artery containing oxygen-poor blood.

Using the member-of and subset-of links shown in the net the aorta inherits the properties of the arteries: the aorta has a muscular wall and transports oxygen-rich blood. Using a similar argument, the left pulmonary artery inherits these two properties as well. The left pulmonary artery, however, transports oxygen-poor blood! So, the property that arteries transport oxygen-rich blood should not be inherited by the left pulmonary artery. When employing the principle of inheritance discussed so far, the inheritance of this property cannot be prevented. A possible solution to this problem is to store the information that an artery contains oxygen-rich blood explicitly with each artery for which this property holds. This is shown in Figure 4.6. A major drawback of this solution is that the general property has been lost.

EXAMPLE 4.8

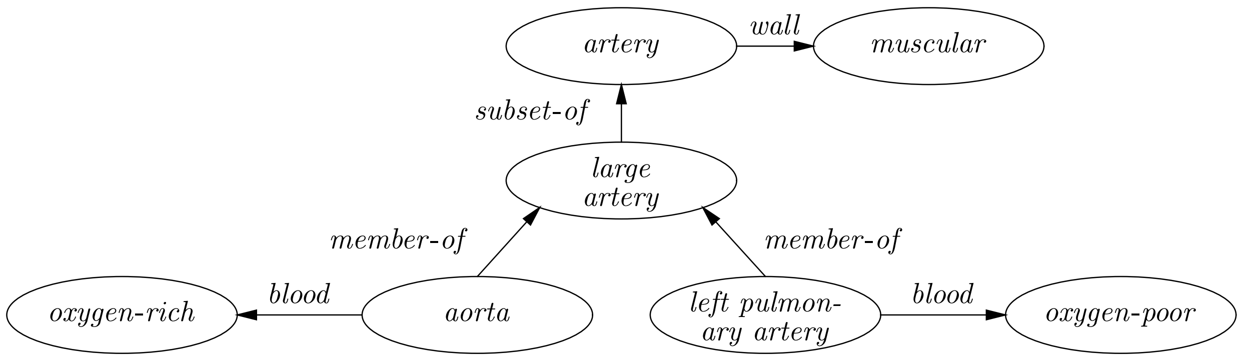


Figure 4.6: Loss of a general property.

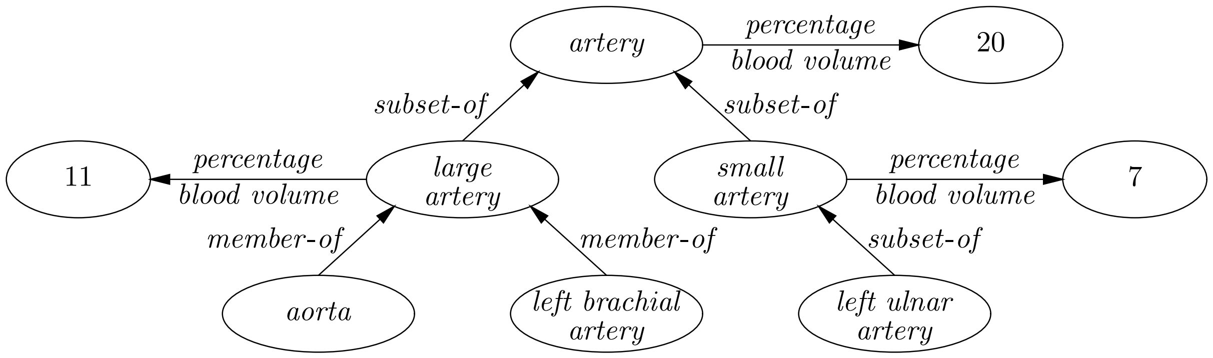


Figure 4.7: Inheritance of properties relevant to a class as a whole.

In the foregoing examples we have discussed properties which are relevant to individual objects. In the semantic net shown in Figure 4.7 some information has been stored that is relevant to a class of objects as a whole and not to the individuals belonging to it. For example, in the net we have represented the information that the large arteries together contain approximately 11% of all the blood the human body contains. This information is only relevant to the class as a whole and not to a single large artery. So, this property should not be inherited by the aorta and the left brachial artery. Furthermore, the information that all arteries together contain 20% of the total blood volume should not be inherited by the class of the small arteries: the latter class only contains 7% of the total blood volume. Again, inheritance cannot be prevented.

It has been mentioned before that the semantic net formalism has undergone many changes since its introduction. The resulting formalisms on the one hand are more restricted: only a limited number of predefined link-types is allowed, each having a clear semantics. On the other hand, many new features have been added to the formalism. In particular the principle of inheritance has been revised in order to make inheritance of properties more flexible. Furthermore, some facilities for representing procedural knowledge have been added to the semantic net. Many of these extensions bear a close resemblance to features of frames. Therefore, we shall not discuss these features in relation to semantic nets here, but only in relation to the frame formalism in Section 4.2.

4.1.3 The extended semantic net

Before we turn our attention to knowledge representation in frames, we conclude this section with a discussion of an interesting type of semantic net: the *extended semantic net*. The extended semantic net was developed by A. Deliyanni and R.A. Kowalski as an alternative representation formalism for the clausal form of logic with a restriction to binary predicate symbols. It should be noted that the restriction to binary predicates is not an essential one: any atom containing an n-placed predicate symbol can be replaced by a conjunction of atoms involving binary predicates only. If $n > 2$, $n + 1$ new predicates are needed to represent the original information; if $n = 1$, only a single new predicate is required.

EXAMPLE 4.9

Consider the three-placed predicate symbol *Bloodpressure*, which is supposed to have the following intended meaning:

$$\mathit{Bloodpressure}(x, y, z) = \text{‘the mean blood pressure in } x \text{ lies between } y \text{ mmHg and } z \text{ mmHg’}$$

The clause

$$\mathit{Bloodpressure}(\mathit{artery}, 40, 80) \leftarrow$$

for example, can be replaced by the following four clauses

$$\begin{array}{l} \mathit{Info}(\mathit{fact}, \mathit{bloodpressure}) \leftarrow \\ \mathit{Subject}(\mathit{fact}, \mathit{artery}) \leftarrow \\ \mathit{Lowerbound}(\mathit{fact}, 40) \leftarrow \\ \mathit{Upperbound}(\mathit{fact}, 80) \leftarrow \end{array}$$

in which only binary predicate symbols have been used to express the same information. We have introduced the new constants *fact* and *bloodpressure*; the new binary predicate symbols should be read as

$$\begin{array}{ll} \mathit{Info}(w, \mathit{bloodpressure}) & = \text{‘}w \text{ is information about blood pressure’} \\ \mathit{Subject}(w, x) & = \text{‘}x \text{ is the subject of the information } w\text{’} \\ \mathit{Lowerbound}(w, y) & = \text{‘}y \text{ is the lower bound of } w\text{’} \\ \mathit{Upperbound}(w, z) & = \text{‘}z \text{ is the upper bound of } w\text{’} \end{array}$$

EXAMPLE 4.10

Consider the unary predicate symbol *Artery* with the following intended meaning:

$$\mathit{Artery}(x) = \text{‘}x \text{ is an artery’}$$

The clause

$$\mathit{Artery}(\mathit{aorta}) \leftarrow$$

may be replaced by the clause

$$\mathit{Isa}(\mathit{aorta}, \mathit{artery}) \leftarrow$$

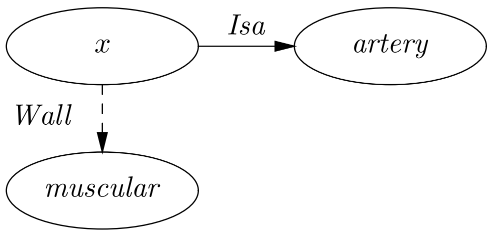


Figure 4.8: The extended semantic net for the clause $Wall(x, muscular) \leftarrow Isa(x, artery)$.

We have mentioned before that the extended semantic net provides an alternative syntax for the clausal form of logic: the arguments to the predicate symbols occurring in a set of clauses are taken as the vertices, and the binary predicate symbols themselves are taken as the labels of the arcs of a directed graph. The direction of the arc expresses the order of the arguments to the predicate symbol. The conclusions and conditions of a clause are represented by different types of arcs: conditions are denoted by pulled arcs and conclusions are indicated by dashed arcs.

EXAMPLE 4.11

The extended semantic net shown in Figure 4.8 represents the clause

$$Wall(x, muscular) \leftarrow Isa(x, artery)$$

A particular constant may occur in more than one clause. If we represent all occurrences of a constant by means of a single vertex, then it is not always apparent in the representation discussed above to which clauses a particular vertex belongs. This is why an extended semantic net representing a set of clauses is divided into a number of subnets, each representing a single clause.

EXAMPLE 4.12

The following set of clauses

$$\{Wall(x, muscular) \leftarrow Isa(x, artery),\ Isa(y, artery) \leftarrow Isa(y, large\text{-}artery)\}$$

has been represented in Figure 4.9. The net is partitioned into two subnets: for each clause from the given clause set we have one corresponding subnet. Note that if a similar situation arises concerning a variable, we can simply rename the variables to avoid the problem.

A pleasant consequence of the syntactical correspondence between the clausal form of logic and the extended semantic net is that the inference rules that are defined for the clausal form of logic can be applied for manipulation of arcs and vertices in an extended semantic net.

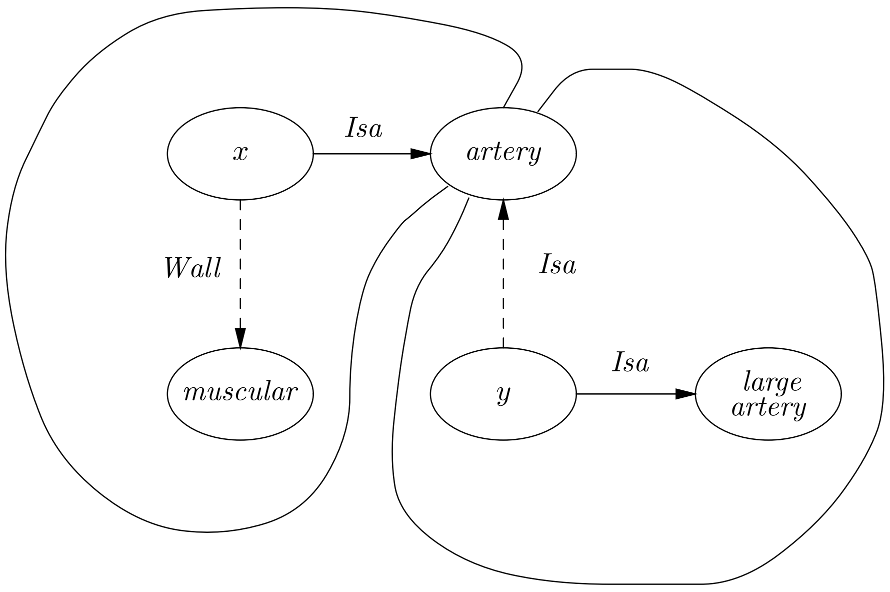


Figure 4.9: Partition of an extended semantic net into subnets.

4.2 Frames and single inheritance

In a frame-based system all knowledge relevant to a concept is stored in a complex entity of representation called a *frame*. Frames provide a formalism for explicitly grouping all knowledge concerning the properties of individual objects or classes of objects. Within a frame, part of the properties is specified as reference information to other, more general frames. This reference information is represented by means of is-a links which are quite similar in concept to the is-a links in a semantic net. This way, the knowledge in a domain of interest is organized hierarchically in what is called a *frame hierarchy*, *frame taxonomy*, or *taxonomy* for short. A taxonomy often is depicted graphically as a directed, acyclic graph once more bearing a close resemblance to a semantic net. However, in the graph representation of a frame taxonomy only the is-a links are shown (as was true for the semantic net, trivial cycles are not shown in the graph since they do not represent additional information); the knowledge concerning an individual object or a class of objects itself is part of the internal structure of the vertices of the graph. In contrast with the semantic net, in a frame representation different components are distinguished, all having a special status allowing them to be treated explicitly as such. For example, the is-a links in a frame taxonomy are represented and treated in a way different from other components.

4.2.1 Tree-shaped frame taxonomies

As has been mentioned above, frames are organized in a taxonomy in which the vertices represent frames and in which every arc denotes an is-a link between two frames. In the frame formalism which will be used in this book, two types of frames are distinguished:

- *class frames*, or *generic frames*, which represent knowledge concerning classes of objects;
- *instance frames*, which represent knowledge concerning individual objects.

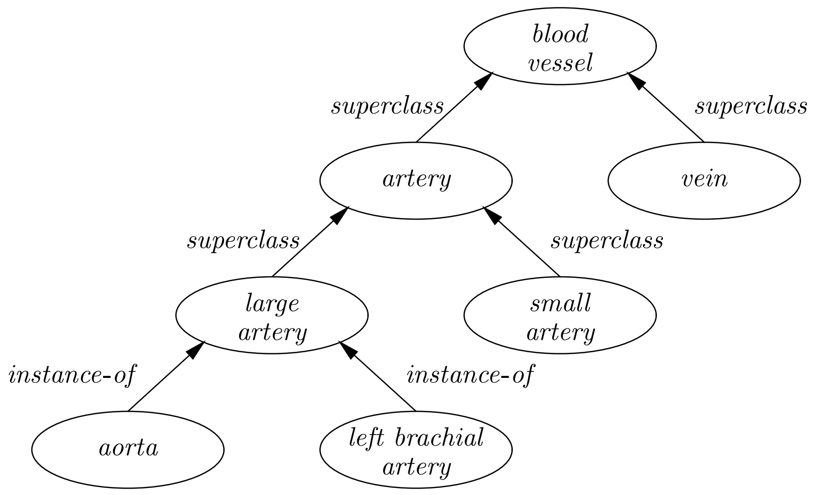


Figure 4.10: A tree-shaped taxonomy.

Class frames have much in common with the record datatype as, for example, provided in the Pascal programming language, and instances are similar to filled-in record variables.

Since there are two types of frames, we also distinguish two types of is-a links by means of which a frame indicates its relative position in the frame taxonomy:

- an *instance-of link*, which is an is-a link between an instance frame and a class frame;
- a *superclass link*, which is an is-a link between two class frames defining a partial order on the set of class frames in a frame taxonomy.

These is-a links are similar in meaning to the member-of and subset-of links, respectively, distinguished for semantic nets in the previous section. Their formal meaning will be discussed in the following.

In the present section, we consider frame taxonomies that can be represented as trees. Section 4.3 discusses the more general graph-shaped taxonomies. An example of a tree-shaped taxonomy is shown in Figure 4.10. In this figure, a frame is represented as an ellipse; the internal structure of a frame is not shown. In a tree-shaped frame taxonomy capturing knowledge concerning a given domain, the root of the tree represents the most general description of the domain: the other frames represent descriptions of concepts that are more specific. The descendants of a certain frame in the taxonomy therefore are often called *specializations* of that frame. The ancestors of a frame in the taxonomy are called its *generalizations*. When we restrict the discussion to classes of objects only, specializations are generally called *subclasses* and generalizations are called *superclasses*. We shall use the terms *superframe* and *subframes* for the parent and children of a given frame, respectively.

EXAMPLE 4.13

Consider the frame taxonomy shown in Figure 4.10 once more. The vertex representing the frame with the name *blood vessel* is the father (and therefore ancestor) of the vertices representing the frames *artery* and *vein*. So, the frame with the name *blood vessel* is the generalization of the frames *artery* and *vein*; it equally is a generalization of the frame with the name *aorta*. The frame with the name *small artery* is a specialization of the *artery* frame.

A frame representing an individual object cannot be specialized any further. Therefore, in a tree-shaped frame taxonomy an instance is always a leaf of the tree.

EXAMPLE 4.14

Consider Figure 4.10 once more. The frames with the names *aorta* and *left brachial artery* cannot be specialized any further, since these frames represent individual objects and therefore are instances. Except for the vertices representing these two frames, the tree has another two leaves: the frames *small artery* and *vein*. These two frames are generic: the descriptions given in these frames may be further specialized. Note that although an instance is always a leaf of the tree, not every leaf is an instance.

We shall now turn our attention to the internal structure of a frame. We assume that each frame in a taxonomy has a unique name. The information specific to the concept represented by a frame is laid down in so-called *attributes* or *slots*. So, attributes offer a means for representing the properties of individual objects or classes of objects. In the following definition, we shall present a syntax of a language for the representation of frames; from then on we shall be able to be more precise in discussing frames and their formal meaning.

Definition 4.2 *A* frame *is a statement having the following form:*

⟨frame⟩	::=	⟨class⟩ | ⟨instance⟩

⟨class⟩	::=	**class** ⟨class-name⟩ **is**
		    superclass ⟨super-specification⟩;
		    ⟨class-attributes⟩
		end

⟨instance⟩	::=	**instance** ⟨instance-name⟩ **is**
		    instance-of ⟨super-specification⟩;
		    ⟨instance-attributes⟩
		end

⟨super-specification⟩	::=	⟨class-name⟩ | **nil**
⟨class-attributes⟩	::=	⟨declaration⟩ {; ⟨declaration⟩}* | ⟨empty⟩
⟨instance-attributes⟩	::=	⟨attribute-value-pair⟩ {; ⟨attribute-value-pair⟩}* | ⟨empty⟩
⟨declaration⟩	::=	⟨attribute-type-pair⟩ | ⟨attribute-value-pair⟩
⟨attribute-type-pair⟩	::=	⟨attribute-name⟩ : ⟨type⟩
⟨attribute-value-pair⟩	::=	⟨attribute-name⟩ = ⟨value⟩

$\langle$type$\rangle$ ::= **int** | **real** | **string** | $\langle$set$\rangle$ | $\langle$class-name$\rangle$

$\langle$value$\rangle$ ::= $\langle$elementary-constant$\rangle$ | $\langle$instance-name$\rangle$

$\langle$empty$\rangle$ ::=

A $\langle$super-specification$\rangle$ *equal to the special symbol* **nil** *is used to indicate that the frame concerned is the root of the tree-shaped taxonomy. As a type, a* $\langle$set$\rangle$ *consists of elementary constants and instance names, separated by comma's and enclosed in curly brackets. An elementary constant is either a real or integer constant, or a string of non-blank characters, that is, an instance of one of the predefined (or standard) classes* **real**, **int**, *and* **string**. *The* $\langle$instance-name$\rangle$ *value of an attribute refers to a uniquely defined instance in the taxonomy.*

For the moment we assume that an attribute-type or attribute-value pair for an attribute only occurs once in a frame taxonomy. Later on we shall drop this restriction. In the preceding definition, we have stated for ease of exposition that a class frame is the root of a tree-shaped taxonomy if it has a super-specification equal to **nil**, where it actually is a subclass of the most general class **nil**. This more accurate interpretation of the symbol **nil**, however, is only important in frame taxonomies in which more than one most general class frame not equal to **nil** occurs; if we did not consider **nil** as the most general class, then the graph representation of such taxonomy would be a forest of trees instead of just a tree.

As can be seen in the preceding definition, the definition of an instance frame is composed of the specification of the class to which the instance belongs followed by a collection of attribute-value pairs. Together they give a description of an individual concept in the domain of discourse. Let us give an example of such an instance.

EXAMPLE 4.15

We consider the left brachial artery which is one of the arteries in the human body. It is known that the left brachial artery has an approximate diameter of 0.4 cm, that it is localized in the upper arm, and that it contains oxygen-rich blood. All this information is captured by the following instance frame:

instance *left-brachial-artery* **is**
  instance-of *artery*;
  diameter = 0.4;
  location = *arm*;
  blood = *oxygen-rich*
end

We have used the attributes *diameter*, *location*, and *blood* for the representation of the mentioned properties of the individual concept 'left brachial artery'. Note that all three attributes have been assigned actual values. The values 0.4 and *oxygen-rich* are assumed to be elementary constants. The value *arm* of the attribute *location* is an instance of the class frame *limb*:

instance *arm* **is**
  superclass *limb*;
  position = *superior*
end

The value *superior* again is an elementary constant.

The information specified in the attribute parts of instance frames has to accord with the following rules. All attributes occurring in the instances of a class frame must have been declared in the attribute part of that class or in one of its generalizations; the values which have been filled in for the attributes in the instance must be of the appropriate attribute type as defined by the classes in the taxonomy. Note that these rules provide part of the meaning of the instance-of link.

EXAMPLE 4.16

Consider the instance *left-brachial-artery* from the preceding example once more. The class to which *left-brachial-artery* belongs is defined as follows:

class *artery* **is**
  superclass *blood-vessel*;
  location : {*arm*,*head*,*leg*,*trunk*}
end

This class frame provides a type declaration for the attribute *location*: it indicates that the *location* attribute is only allowed to take a value from the set {*arm*,*head*,*leg*,*trunk*}. Note that the value *arm* given for the *location* attribute in the *left-brachial-artery* instance is indeed of the correct type. Not all attributes occurring in the instance have been declared in the class frame *artery*; so, the *diameter* and *blood* attributes must have been declared in some of the generalizations of the *artery* class. In the previous example we have mentioned that the instance frame *arm* belongs to the class *limb*. This class frame for example is defined as follows:

class *limb* **is**
  superclass nil;
  position : {*inferior*,*superior*}
end

From the superclass specification **nil** we have that this class is the root of a tree-shaped taxonomy.

In the preceding example, the class frames we considered had attribute-type pairs only in their declaration part. However, the syntax definition indicates that also attribute-value pairs are allowed in the declaration part of a class frame. The following example illustrates this idea.

EXAMPLE 4.17

Consider the previous examples once more. Since most arteries contain oxygen-rich blood, there is no purpose in repeating this information for all individual arteries separately. In this case, it appears to be convenient to fix the value *oxygen-rich* for the attribute *blood* in advance in the class frame *artery*. We then obtain the following alternative definition for the *artery* frame:

```
class artery is
    superclass blood-vessel;
    location : {arm,head,leg,trunk};
    blood = oxygen-rich
end
```

The instance frame *left-brachial-artery* may now be simplified to

```
instance left-brachial-artery is
    instance-of artery;
    diameter = 0.4;
    location = arm
end
```

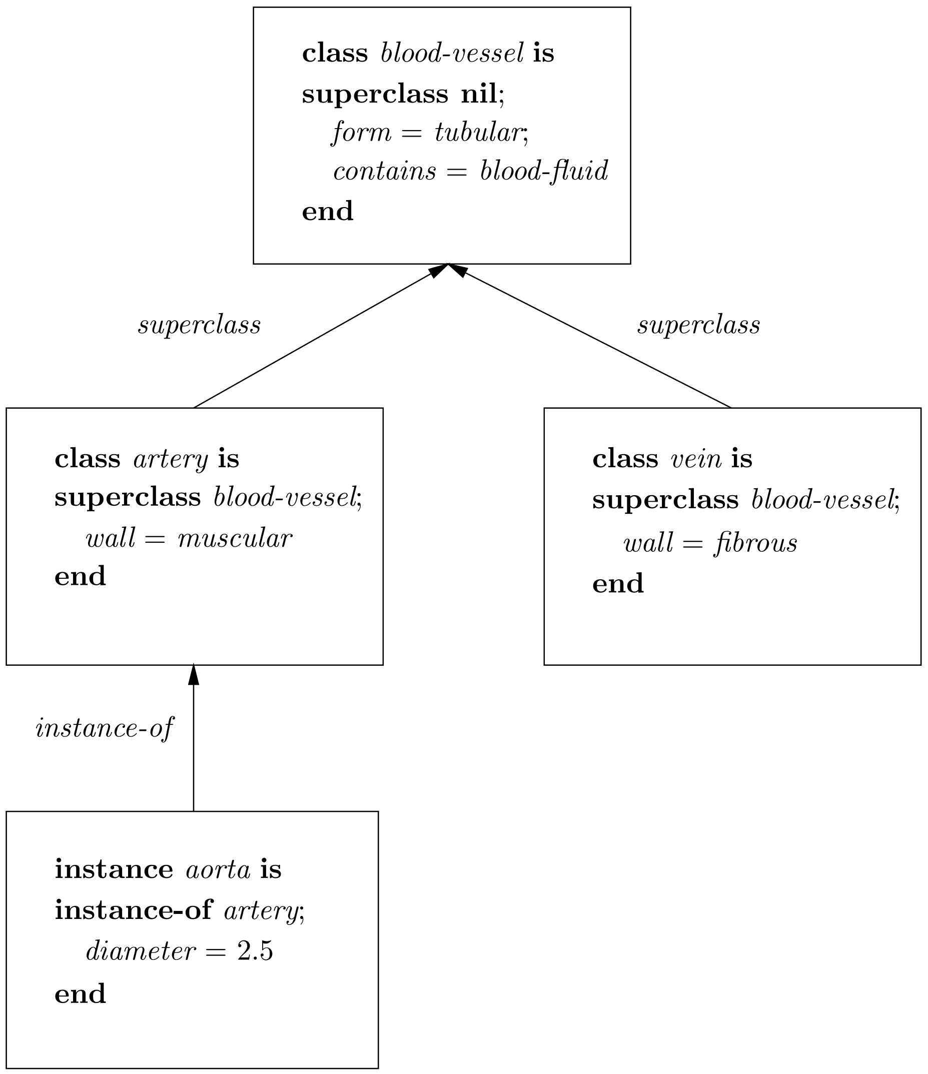
without affecting the intended meaning.

Although only informally, we have now fully described the meaning of the instance-of link. We turn our attention to the superclass link. Recall that we have distinguished two different types of attribute information in a frame taxonomy: information about attribute types and information about attribute values. Accordingly, in assigning a meaning to the superclass link we have to distinguish between these different types of attribute information. First of all, the superclass link defines a partial order on the class frames in a taxonomy and may be applied for reasoning about attribute values much in the same way we have seen for semantic nets. Secondly, however, the superclass link may be viewed as defining a relation which restricts the semantic contents of the frame taxonomy as we have shown in the preceding example. This is essentially a higher-order relation. These two different ways of interpreting the superclass link are best treated separately. We shall therefore first study the meaning of attribute values in a tree-shaped taxonomy and show how it is possible to reason about such attribute values. From now on, we shall, for ease of exposition, completely disregard the fact that classes may contain type information until Section 4.2.4 in which we shall return to attribute types.

We have mentioned before that the semantic net formalism we have briefly discussed in Section 4.1 may be viewed as a precursor of the frame formalism. We take a closer look at the relationship between the two formalisms. This relationship can be examined more readily if we depict a frame taxonomy in a graph as we have done with the semantic net. The following example shows the general idea.

EXAMPLE 4.18

Figure 4.11 shows four frames in a tree-shaped taxonomy. The frames are represented as boxes; the internal structure of each of the frames is depicted. The arcs in the graph represent the instance-of and superclass links between the frames. Note that the frames themselves already indicate their position in the taxonomy explicitly; the graphical representation of the taxonomy therefore contains redundant information. From a graph representing a frame taxonomy we can easily derive an equivalent semantic net. Figure 4.12 shows the semantic net equivalent to the taxonomy depicted in Figure 4.11. Note that although the corresponding semantic net essentially comprises the same information as the original frame taxonomy, the apparent modularity of the taxonomy has been lost.



class *blood-vessel* **is**
superclass nil;
  form = *tubular*;
  contains = *blood-fluid*
end

class *artery* **is**
superclass *blood-vessel*;
  wall = *muscular*
end

class *vein* **is**
superclass *blood-vessel*;
  wall = *fibrous*
end

instance *aorta* **is**
instance-of *artery*;
  diameter = 2.5
end

Figure 4.11: A tree-shaped taxonomy showing the internal structure of the frames.

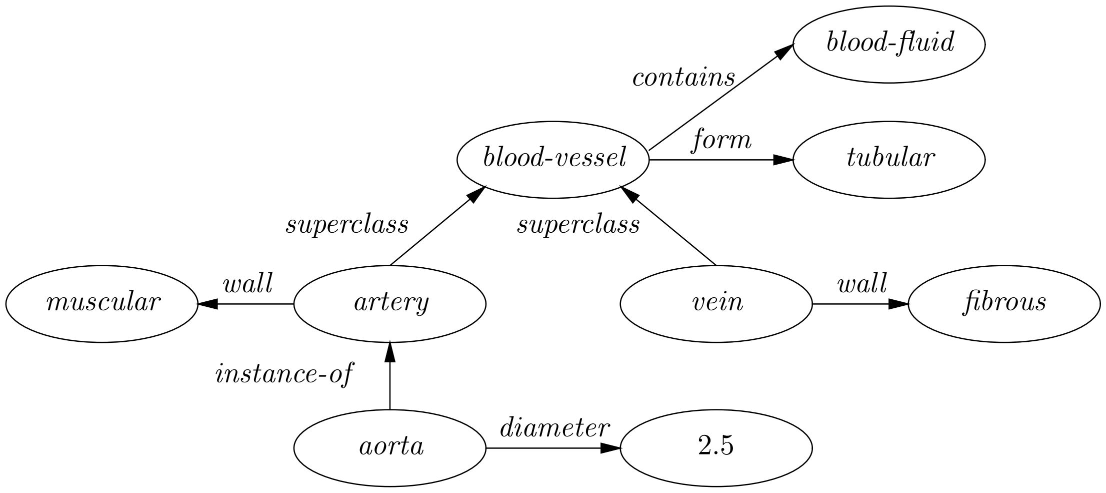


Figure 4.12: The semantic net corresponding to the taxonomy shown in Figure 4.11.

Based on the frame formalism defined above we shall discuss the meaning that can be associated with the frame formalism. The discussion takes the following example for a starting point.

EXAMPLE 4.19

Suppose that we want to represent in a frame the following information concerning the vascular system: the aorta is an artery having a diameter of 2.5 cm. Using our frame formalism, this information may be represented as follows:

```
instance aorta is
    instance-of artery;
    diameter = 2.5
end
```

The information that an artery is a blood vessel having a muscular wall is represented in the following class frame:

```
class artery is
    superclass blood-vessel;
    wall = muscular
end
```

To conclude, the following class frame shows that a blood vessel is tubular in form and contains blood:

```
class blood-vessel is
    superclass nil;
    form = tubular;
    contains = blood-fluid
end
```

The last frame furthermore indicates that the *blood-vessel* frame is the root of the taxonomy concerned.

The information that is specified in a frame taxonomy can also be expressed in first-order predicate logic, roughly by complying with the following directions:

- take the names of the instances as constants;
- take the names of the class frames as unary predicate symbols;
- translate an instance-of link into a predicate symbol having a constant for an argument;
- translate a superclass link into a logical implication;
- take the attribute names as unary function symbols;

- translate an attribute-value pair into an equality between a function term and a constant.

EXAMPLE 4.20

Assuming a suitable interpretation, the following formulas represent the same information as the frames in the foregoing example do:

$$
\begin{array}{l}
\mathit{artery}(\mathit{aorta}) \\
\mathit{diameter}(\mathit{aorta}) = 2.5 \\
\\
\forall x(\mathit{artery}(x) \rightarrow \mathit{blood\text{-}vessel}(x)) \\
\forall x(\mathit{artery}(x) \rightarrow \mathit{wall}(x) = \mathit{muscular}) \\
\\
\forall x(\mathit{blood\text{-}vessel}(x) \rightarrow \mathit{form}(x) = \mathit{tubular}) \\
\forall x(\mathit{blood\text{-}vessel}(x) \rightarrow \mathit{contains}(x) = \mathit{blood\text{-}fluid})
\end{array}
$$

The semantics of first-order predicate logic may now be exploited to define a semantics for the frame formalism. Under the assumption that each attribute only occurs once in the taxonomy, we may ascribe a meaning based on first-order predicate logic to the set of frames of this taxonomy using the following general translation scheme:

$$
\begin{array}{lll}
\textbf{class } C \textbf{ is} & & \\
\quad \textbf{superclass } S; & & \forall x(C(x) \rightarrow S(x)) \\
\quad a_1 = b_1; & \Rightarrow & \forall x(C(x) \rightarrow a_1(x) = b_1) \\
\quad \vdots & & \vdots \\
\quad a_n = b_n & & \forall x(C(x) \rightarrow a_n(x) = b_n) \\
\textbf{end} & &
\end{array}
$$

$$
\begin{array}{lll}
\textbf{instance } I \textbf{ is} & & \\
\quad \textbf{instance-of } C; & & C(I) \\
\quad a_1 = b_1; & \Rightarrow & a_1(I) = b_1 \\
\quad \vdots & & \vdots \\
\quad a_n = b_n & & a_n(I) = b_n \\
\textbf{end} & &
\end{array}
$$

Under the mentioned assumption we have that there always exists an interpretation I of the thus obtained logical formulas which is a model. In the next section we shall study the case in which the restriction that an attribute-value pair for an attribute occurs only once in a taxonomy has been dropped.

We conclude this section with a discussion of the inference method associated with the frame formalism. We start by examining the derivations that can be made from the corresponding logical formulas by means of a sound and complete collection of inference rules.

EXAMPLE 4.21

Consider the formulas given in the previous example once more. From the formulas

$$artery(aorta)$$
$$\forall x(artery(x) \rightarrow wall(x) = muscular)$$

we can derive the formula

$$wall(aorta) = muscular$$

using modus ponens. Similarly, from the set of formulas in the preceding example the following formula can be derived:

$$blood\text{-}vessel(aorta)$$

When closely examining these derivations, we see that the information holding for arteries in general is explicitly said to hold for the aorta in particular: since we know that the aorta is an artery, the aorta *inherits* this information from the arteries. Similarly, the aorta inherits the information specific to blood vessels.

In the foregoing example we have demonstrated the reasoning behaviour with logical formulas representing the information stored in a given frame taxonomy. This reasoning behaviour is modelled in an inference method for frames called *single inheritance.* In case of a tree-shaped taxonomy, we speak of *single* inheritance to stress the fact that each frame has at most one superframe. In contrast, the inference method associated with more general, graph-shaped taxonomies is called *multiple* inheritance; we shall discuss multiple inheritance in Section 4.3. Informally speaking, in single inheritance all information that holds for a particular frame is determined by traversing the taxonomy from the frame itself to the root of the taxonomy, that is, the most general frame, and successively collecting the attributes with their associated value that are found in the encountered frames. This may be viewed as exploiting the transitivity property of the superclass relation. This procedure terminates as soon as the information in the root of the taxonomy has been processed. The function shown below describes the recursive inference procedure:

```
function Inherit(frame, attribute-value-pairs)
  if frame = nil then
    return(attribute-value-pairs)
  end;
  attribute-value-pairs ← attribute-value-pairs ∪ AttributePart(frame);
  return(Inherit(Superframe(frame), attribute-value-pairs))
end
```

The parameters `frame` and `attribute-value-pairs` take as values a frame name and a collection of attribute-value pairs, respectively. If the parameter `frame` equals **nil**, then either the taxonomy is empty or the root of the taxonomy has been reached: in both cases all attribute-value pairs holding for the frame concerned have been collected in the second argument `attribute-value-pairs`. If the parameter `frame` differs from the value **nil**, then all attribute-value pairs specified in the frame `frame` are extracted from it using the function `AttributePart` and added to `attribute-value-pairs`. The information holding for the superframe of the given frame `frame` is subsequently determined by means of a recursive call to the `Inherit` function.

EXAMPLE 4.22

Consider Figure 4.11 again. In the instance frame with the name *aorta* the attribute-value pair

> *diameter* = 2.5

has been specified. Using the `Inherit` function described in the foregoing, the instance inherits the attribute-value pair

> *wall* = *muscular*

from its superframe *artery*. From the superframe *blood-vessel* of the frame *artery*, the instance inherits the following two attribute-value pairs:

> *contains* = *blood-fluid*
> *form* = *tubular*

4.2.2 Exceptions

In the previous section we have introduced a semantics for the frame formalism based on first-order predicate logic. To this end, we assumed that attributes occurred only once in a frame taxonomy. This assumption, however, renders the frame formalism not flexible enough for coping with all practical applications. In this section we therefore abandon this rather restrictive assumption and investigate the problems that arise from doing so; we will assume, however, that in a given frame an attribute can only take one value at a time. Allowing attributes to occur more than once in a frame taxonomy increases the expressive power of the formalism: it has become possible to state *exceptions* to information that holds in general but for some special cases. The following example shows the way an exception may be represented; it furthermore discusses the consequence of the introduction of exceptions into the formalism with respect to its semantics.

EXAMPLE 4.23

We have said in the preceding section that most arteries contain oxygen-rich blood. The following class frame captures this knowledge:

> **class** *artery* **is**
>   **superclass** *blood-vessel*;
>   *blood* = *oxygen-rich*
> **end**

However, it is known that the left and right pulmonary arteries are exceptions to this property of arteries: the pulmonary arteries have almost all properties arteries have but, opposed to arteries in general, they transport oxygen-poor blood. Restricting the discussion to the left pulmonary artery only, this information has been specified in the following instance frame:

```
instance $\textit{left-pulmonary-artery}$ is
    instance-of $\textit{artery}$;
    $\textit{blood} = \textit{oxygen-poor}$
end
```

We now have expressed that the value *oxygen-poor* of the attribute *blood* is an exception to the value *oxygen-rich* of the attribute *blood* that has been specified in the superframe *artery* of the instance: informally speaking, the 'general' value has been surpassed. Note that the attribute *blood* is no longer unique in the taxonomy.

Applying the general translation scheme for converting these two frames into formulas in first-order predicate logic, we obtain the following set of formulas:

$$\textit{artery}(\textit{left-pulmonary-artery})$$
$$\textit{blood}(\textit{left-pulmonary-artery}) = \textit{oxygen-poor}$$

$$\forall x(\textit{artery}(x) \to \textit{blood-vessel}(x))$$
$$\forall x(\textit{artery}(x) \to \textit{blood}(x) = \textit{oxygen-rich})$$

This set of logical formulas is inconsistent, since by means of modus ponens we can derive the following logical consequences:

$$\textit{blood}(\textit{left-pulmonary-artery}) = \textit{oxygen-poor}$$
$$\textit{blood}(\textit{left-pulmonary-artery}) = \textit{oxygen-rich}$$

The inconsistency now follows from the equality axioms (these are assumed to be implicitly present). We assume that the unique name assumption holds, that is, symbols (function symbols, predicate symbols, and constants) with different names are assumed to be different. Now observe that in any model for the logical formulas shown above the constants *oxygen-rich* and *oxygen-poor* are equal. This, however, contradicts the unique name assumption.

In the foregoing example we have demonstrated that in the frame formalism exceptions are represented by locally surpassing attribute values. Furthermore, it has been shown that in case we allow multiple occurrences of attributes the translation of the frame formalism into first-order predicate logic may render an inconsistent set of formulas; it is not possible to fully capture the notion of exceptions by standard first-order predicate logic. The meaning of the frame formalism allowing for exceptions, however, can be described using a non-standard logic, such as for example the non-monotonic logic developed by D. McDermott and J. Doyle, or by the default logic developed by R. Reiter. We do not enter into these theories in detail; we merely give a sketch of their respective general idea.

We first consider the *non-monotonic logic* of McDermott and Doyle. In non-monotonic logic, first-order predicate logic is extended with a special *modal operator* M. The truth of a formula $M(f(x) = c)$ now means that the formula $f(x) = c$ is *possibly* true; in other words, it is not possible to derive from the given set of formulas, formulas $f(x) = d$ with $d \neq c$. In our example, the formula $\textit{blood}(\textit{left-pulmonary-artery}) = \textit{oxygen-poor}$ must be true in all models for our set of logical formulas. It therefore is undesirable that the formula

$\mathit{blood}(\mathit{left\text{-}pulmonary\text{-}artery}) = \mathit{oxygen\text{-}rich}$ can be derived, since this would lead to an inconsistency. Using the modal operator M we can block the derivation of the latter formula. The new formulas representing the given information now are as follows:

$$\begin{array}{l} \mathit{artery}(\mathit{left\text{-}pulmonary\text{-}artery}) \\ \mathit{blood}(\mathit{left\text{-}pulmonary\text{-}artery}) = \mathit{oxygen\text{-}poor} \\ \\ \forall x(\mathit{artery}(x) \rightarrow \mathit{blood\text{-}vessel}(x)) \\ \forall x(\mathit{artery}(x) \wedge M(\mathit{blood}(x) = \mathit{oxygen\text{-}rich}) \rightarrow \mathit{blood}(x) = \mathit{oxygen\text{-}rich}) \end{array}$$

Informally speaking, these formulas state that for a constant e the formula $\mathit{blood}(e) = \mathit{oxygen\text{-}rich}$ can only be derived if no other formula $\mathit{blood}(e) = c$ with $c \neq \mathit{oxygen\text{-}rich}$ can be derived. So, the formula $\mathit{blood}(\mathit{left\text{-}pulmonary\text{-}artery}) = \mathit{oxygen\text{-}rich}$ no longer is a logical consequence of the above-given set of formulas.

The *default logic* developed by R. Reiter equally provides a way of handling exceptions but from a different perspective than non-monotonic logic does. In default logic, special inference rules, called *defaults*, are added to first-order predicate logic. The translation of the frame formalism into default logic now yields a set of logical formulas and a set of defaults. In the present case, we obtain the following set of logical formulas

$$\begin{array}{l} \mathit{artery}(\mathit{left\text{-}pulmonary\text{-}artery}) \\ \mathit{blood}(\mathit{left\text{-}pulmonary\text{-}artery}) = \mathit{oxygen\text{-}poor} \\ \\ \forall x(\mathit{artery}(x) \rightarrow \mathit{blood\text{-}vessel}(x)) \end{array}$$

and the following default

$$\frac{\mathit{artery}(x) : \mathit{blood}(x) = \mathit{oxygen\text{-}rich}}{\mathit{blood}(x) = \mathit{oxygen\text{-}rich}}$$

A default consists of a *prerequisite*, in our case the formula $\mathit{artery}(x)$, and a set of so-called *justifications*, here the formula $\mathit{blood}(x) = \mathit{oxygen\text{-}rich}$; these are specified above the line. It furthermore contains a *consequent*, here $\mathit{blood}(x) = \mathit{oxygen\text{-}rich}$, specified below the line. In this example, the default expresses that given the satisfiability of the prerequisite $\mathit{artery}(x)$ for some x in the domain and given that there are no formulas which contradict the justification $\mathit{blood}(x) = \mathit{oxygen\text{-}rich}$, then the consequent $\mathit{blood}(x) = \mathit{oxygen\text{-}rich}$ may be derived. So, in the present case $\mathit{blood}(\mathit{left\text{-}pulmonary\text{-}artery}) = \mathit{oxygen\text{-}rich}$ cannot be derived. This is precisely what we wanted to achieve.

We conclude this section by introducing an inheritance procedure that respects the intuitive meaning of a frame formalism allowing for exceptions. It is obvious that the inheritance procedure described in the previous section cannot be applied in case attributes occur more than once in a taxonomy: this procedure might come up with conflicting information. However, only a minor modification of the procedure suffices to let it cope with exceptions. The general idea of the alteration of the inheritance procedure is as follows. Just before an attribute-value pair is added to the set of collected attribute-value pairs, it is examined whether the attribute name concerned already occurs in this set: in that case, the attribute value has been surpassed by an exception somewhere lower in the taxonomy. An attribute-value pair is only then added to the set of collected attribute values if the attribute name is not present as yet in this set. The following function describes the altered inheritance procedure more formally:

```
function Inherit(frame, attribute-value-pairs)
   if frame = nil then
     return(attribute-value-pairs)
   end;
   pairs ← AttributePart(frame);
   attribute-value-pairs ← attribute-value-pairs ∪
             NewAttributes(pairs, attribute-value-pairs);
   return(Inherit(Superframe(frame), attribute-value-pairs))
end
```

The function `NewAttributes` is used to delete from `pairs` those attribute-value pairs of which the attribute name already occurs in `attribute-value-pairs`.

The intuitive idea of this new inheritance function is that the value which holds for an attribute is given in the frame itself or in the nearest frame higher in the taxonomy providing a value for the attribute.

4.2.3 Inheritance and attribute facets

In our treatment of inheritance of attribute values in the preceding sections, we did not pay any attention to the way in which these values were obtained. In many practical applications, however, it may be important to know whether an attribute value for a given instance has been obtained by inheritance or has been explicitly specified in some way: in the latter case, the user is likely to have more confidence in the accurateness of the value than in the former case where the value has only been stated for an entire class of instances. Furthermore, it often is desirable to be able to compute attribute values based on the values of some other attributes which have been obtained during a consultation. The frame formalism discussed in the preceding sections is not able to cope with such situations. It is not surprizing therefore that most frame formalisms that are employed in systems which are actually used in practical applications, offer special language constructs, called *facets*, for the purpose of handling the situations mentioned above. In this section, we shall discuss some of the facets that are most frequently met in literature.

A facet may be viewed as a property associated with an attribute. The most common facet is the *value facet* referring to the actual value of the attribute. The value stored in a value facet of an attribute is assumed to have been established with absolute certainty. Since it is often difficult to specify with certainty in advance the values attributes of instances of a class will adopt, the initial values for class attributes are often specified in *default facets*. These default values may be overridden as the consultation proceeds. Note that our algorithm for single inheritance with exceptions already exhibited this behaviour; the difference, however, is that when facets are used, an inherited default attribute value is still marked as being a default value. In general, it depends on the characteristics of the problem area which part of the attribute values will be specified in a default facet and which part is specified in a value facet. The values specified in the default facets of attributes in a frame taxonomy together offer a typical picture of the domain of discourse.

The third facet we discuss is the *demon facet*, or *demon* for short. A demon is a procedure that will be invoked at a particular time during the manipulation of the frame in which it has been specified. The condition under which a demon is activated depends upon the type of the demon. An *if-needed demon* is activated the moment an attribute value is needed but not yet known for the attribute it is attached to. An *if-added demon* is activated the moment a value is entered into the value facet of the attribute concerned. An *if-removed demon* is invoked the moment a value is removed from the value facet of the attribute it is attached to. This way of integrating procedural and declarative knowledge is called *procedural attachment*. The frame formalism gains enormously in expressive power by the incorporation of demons. Using demons and attribute values, it for example is possible to represent local state changes due to computation: the state of the computation at a certain moment during a consultation is described by the values the attributes have at that moment.

So far, we have discussed inheritance as the only method for frame manipulation. It will be evident, however, that inheritance alone does not render a full inference engine: inheritance accounts for only a small portion of the inference engine of most frame-based systems. In many systems, demons are used to influence the overall control exerted in manipulating the frame taxonomy: an if-added demon for instance may be used to direct the control to a particular frame as a side-effect. However, great care must be taken in applying such techniques: when such side-effects are applied very often, the behaviour of the system will become difficult to fathom. This, however, is not true for every use of demons. An if-needed demon for instance can be an algorithm for asking the user for further information or for calculating a value for example in handling time-dependent information.

EXAMPLE 4.24

Consider a real-time expert system for controlling some ongoing process that has the possibility to read off several gauges indicating the status of this process. The activation of an if-needed demon may result in reading off some of the gauges as soon as information concerning the status of the process is required.

To conclude this informal introduction to demons, we observe that frames are often used as a means of partitioning a given set of production rules: each frame then has command of a certain partition of the set of rules. In a frame taxonomy supporting this idea, a demon is used to initiate the consultation of such a partition. It is closely related to concept of *methods* in object-oriented programming. Besides the three general facets discussed in the foregoing it is of course possible to define several domain-dependent facets for a given application where appropriate.

With reference to the foregoing discussion, the following definition states a simple, implementation-oriented formalism for frames allowing for facets to be attached to attributes; once more we have refrained from type information.

Definition 4.3 *A* frame *is an expression of the following form:*

```
⟨frame⟩        ::= ⟨class⟩ | ⟨instance⟩

⟨class⟩        ::= class ⟨class-name⟩ is
                     superclass ⟨super-specification⟩;
                     ⟨attributes⟩
                   end

⟨instance⟩     ::= instance ⟨instance-name⟩ is
```

instance-of ⟨super-specification⟩;
⟨attributes⟩
end

⟨super-specification⟩	::=	⟨class-name⟩ | **nil**
⟨attributes⟩	::=	⟨attribute-facet-pair⟩ {; ⟨attribute-facet-pair⟩}* | ⟨empty⟩
⟨attribute-facet-pair⟩	::=	⟨attribute-name⟩ = (⟨facet⟩ {, ⟨facet⟩}*)
⟨facet⟩	::=	⟨facet-name⟩⟨value⟩ | **demon** ⟨demon-type⟩⟨demon-call⟩
⟨facet-name⟩	::=	**value** | **default**
⟨demon-type⟩	::=	**if-needed** | **if-added** | **if-removed**
⟨value⟩	::=	⟨elementary-constant⟩ | ⟨instance-name⟩
⟨empty⟩	::=	

Again, a class specification equal to the special symbol **nil** is used to indicate that the frame concerned is the root of the tree-shaped taxonomy.

EXAMPLE 4.25

The frame shown below describes the class of arteries which is a subclass of the class of blood-vessels:

```
class artery is
    superclass blood-vessel;
    wall = (value muscular);
    blood = (default oxygen-rich);
    blood-pressure = (default 20);
    blood-flow = (default 4);
    resistance = (demon if-needed R(blood-pressure, blood-flow))
end
```

The frame has five attributes. For the attribute with the name *wall* a value facet has been specified, since we are absolutely certain that all arteries have a muscular wall. For the three attributes with default facets things are different. We already know that not all arteries contain oxygen-rich blood, hence the default facet for the *blood* attribute. The attribute *blood-pressure* represents the difference in blood pressure between the pressure at beginning and at the end of an artery. The value specified for this attribute as well as the value for the *blood-flow* attribute are average values for middle-sized arteries. Evidently, such knowledge is best represented in default facets. Finally, we have one attribute, the *resistance* attribute, for which an if-needed demon has been specified for calculating a value upon request. The demon call R(*blood-pressure*, *blood-flow*) represents the call to the procedure R for computing the resistance to the blood flow in the given artery using the formula

$$resistance = \frac{blood\text{-}pressure}{blood\text{-}flow}$$

The values of the attributes *blood-pressure* and *blood-flow* are passed to the procedure.

Since we now allow for various facets to be attached to attributes, it has become necessary to incorporate information concerning the order in which such facets are considered in our algorithm for single inheritance. Basically, there are two types of inheritance of attributes with facets, only differing in the order in which the facets are dealt with:

- *N-inheritance*, and
- *Z-inheritance.*

These types of inheritance owe their respective names to the way the taxonomy is traversed.

The intuition underlying N-inheritance is that any value in a value facet appearing in a frame or in one of its generalizations is closer to the real value than any value obtained from a default facet or from invoking a demon. As usual, of all the values of an attribute stored in value facets in the various frames in the frame taxonomy the most specific one will be inherited by the frame of concern. The basic idea underlying the use of Z-inheritance is that any specific attribute value, whether obtained from a value facet, from invoking a demon or from a default facet, is more reliable than any more general attribute value no matter in which facet it has been specified; however, within a given frame, values specified in a value facet are preferred over those computed by a demon or provided by a default facet.

The procedures for N- and Z-inheritance can now be described informally as follows. Applying N-inheritance, an attribute value is determined by first examining the value facet of the frame concerned. If no value facet has been specified, then the value facet attached to the corresponding attribute in the superframe of the frame is examined. This process is repeated until in a frame higher in the taxonomy a value facet has been found or the root of the taxonomy has been reached. If the process has not yielded an attribute value as yet, then the control is returned to the frame of interest, and the process is repeated for the if-needed demons. Finally, if this process has still not yielded an attribute value, the default facets will be examined in a similar way. Figure 4.13 depicts the behaviour of N-inheritance graphically. Applying Z-inheritance, an attribute value is determined by successively examining the value facet, the if-needed demon and the default facet of the frame concerned before the frames higher in the taxonomy are considered. Figure 4.14 shows the behaviour of Z-inheritance.

4.2.4 Subtyping in tree-shaped taxonomies

In Section 4.2.1 we have introduced a syntax for a frame formalism which allowed for declarations of attribute types. Until now we have disregarded such type information. In the present section, however, we shall pay attention to type information and discuss the properties of the relation defined by the superclass links now viewed as a relation between attribute types. In discussing this relation in the context of type information, it is more common to speak of the *supertype relation*, or reversely of the *subtype relation*; exploiting the subtype relation is known as *subtyping.* In the present section, we study subtyping in tree-shaped

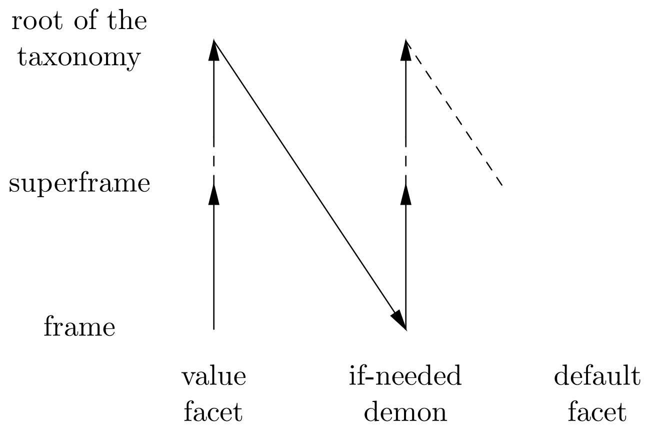


Figure 4.13: N-inheritance.

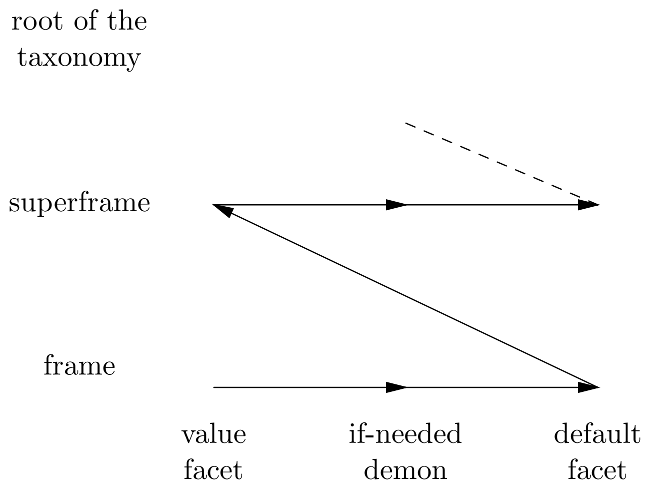


Figure 4.14: Z-inheritance.

frame taxonomies in which attribute-type pairs for a particular attribute may occur more than once.

It will be evident that it is desirable to have type information available in a fame taxonomy: attribute types may be exploited for checking entered values on being of the proper type. This way, an 'unexpected' attribute value can be detected as soon as it is entered into the attribute concerned; in that case, the attribute value is considered to be erroneous.

EXAMPLE 4.26

Consider a frame taxonomy representing information concerning the human cardiovascular system. We assume that the class frame with the name *artery* has an attribute *mean-blood-pressure* that describes the mean blood pressure in mmHg in the arteries in a normal, healthy human being. In the arteries the mean blood pressure ranges from 30 mmHg to 100 mmHg. This information may be stored as type information for the *mean-blood-pressure* attribute. When in a patient for a specific artery, say the ulnar artery, a mean blood pressure of 10 mmHg is found, then this is an unexpected value and probably some action has to be taken upon this event. The specification of an attribute type can be further specialized, that is, narrowed down, as the frame it is specified in is further specialized. Suppose for example that the *artery* class has three specializations: the *large-artery*, the *small-artery*, and the *arteriole* class. In the class representing the large arteries, the information for the *mean-blood-pressure* attribute is further specialized to the range 90 − 100 mmHg, in the *small-artery* class to the range 60–90 mmHg and in the *arteriole* class to 30–60 mmHg.

We give a more formal example in which we have specified some type information in the manner prescribed by the frame formalism.

EXAMPLE 4.27

Consider again the class of blood vessels. The following class representation shows some attribute types for this class of objects:

class *blood-vessel* **is**
    superclass nil;
    blood : {*oxygen-rich*, *oxygen-poor*};
    wall : {*muscular*, *fibrous*, *mixed*}
end

The class of arteries is a subclass of the class of blood-vessels. It is represented in the following class frame:

class *artery* **is**
    superclass *blood-vessel*;
    wall : {*muscular*, *mixed*};
    wall-thickness : **real**
end

Every attribute type specified in the class frame with the name *blood-vessel* now is taken to apply to the class *artery* as well, as long as it has not been further specialized in the *artery* class itself. Note that both classes contain a type declaration for the attribute with the name *wall*. The type declaration included in the *blood-vessel* class is more general than the one in the *artery* class.

From these two examples, we may conclude that subtyping involves the relationship between attributes occurring in the classes as well as the relationship between the types of those attributes. Before going into more detail, we first introduce some new notions that will be used in formalizing subtyping in a tree-shaped frame taxonomy. We shall see that it is convenient to have some representation of the set of attribute names that are of concern to a specific class. Since an attribute type may itself be a class, it does not suffice to simply enlist the attribute names actually occurring in a class. Therefore, we associated with a class a set of so-called attribute sequences.

Definition 4.4 *Let A be the set of all attribute names occurring in a frame taxonomy. An* attribute sequence *a is a string of the form $a_1 : \cdots : a_n$, where $a_i \in A$, $i = 1, \ldots, n$, $n \geq 0$, that is, an attribute sequence is composed of elements from A separated by semicolons. The attribute sequence comprising no elements at all, that is, for which $n = 0$, is called the* empty attribute sequence *and is denoted by ϵ. From now on, we shall use A^* to denote the (infinite) set of all attribute sequences constructed from A.*

Note that we have not imposed any restriction on for example the order in which attribute names are allowed to occur in an attribute sequence. With every class frame in a frame taxonomy we now associate a subset of the set of attribute sequences.

Definition 4.5 *Let A^* be a set of attribute sequences associated with a frame taxonomy as defined above. Let y be a class frame in the taxonomy. With y we associate the set $D(y) \subseteq A^*$, called the* domain *for y, defined by:*

(1) $\epsilon \in D(y)$;

(2) For every attribute name a specified in y, we have that $a \in D(y)$;

(3) For every attribute with the name a of type w specified in y, $D(y)$ contains the attribute sequences $a : b$ for all elements $b \in D(w)$;

(4) The set $D(z)$ of attribute sequences associated with the superframe z of y is a subset of $D(y)$.

We give an example.

EXAMPLE 4.28 ---

Consider the following class frame with the name *blood-vessel*:

class *blood-vessel* **is**
  superclass nil;
  volume : *cubic-measure*
end

This frame specifies an attribute *volume* providing information concerning the blood volume for the specializations of the class. The type *cubic-measure* of this attribute is a class frame itself. This class is defined as follows:

```
class cubic-measure is
    superclass nil;
    size : real;
    unit : {mm^3, cm^3, dm^3, m^3}
end
```

The set of attribute names in the taxonomy consisting of these two frames is equal to $A = \{volume, size, unit\}$. The set of attribute sequences associated with the class frame *blood-vessel* is the set $D(blood\text{-}vessel) = \{\epsilon, volume, volume : size, volume : unit\}$; the domain for the *cubic-measure* class is the set $D(cubic\text{-}measure) = \{\epsilon, size, unit\}$.

In the following definition we introduce the notion of a type function for computing the types of the attribute sequences associated with a given frame.

Definition 4.6 *Let A^* be the set of attribute sequences in a frame taxonomy. Let K be the set of class names in that frame taxonomy (including the standard classes and the most general class* **nil***). For each class $y_i \in K$, let $D(y_i) \subseteq A^*$ be the set of attribute sequences associated with y_i as in the preceding definition. Now, for each $y_i \in K$, we define a* type function *$\tau_i : A^* \to K$ as follows:*

(1) For the empty attribute sequence ϵ, we have that $\tau_i(\epsilon) = y_i$;

(2) For each attribute sequence $a = a_1 : \cdots : a_n \in D(y_i)$, $n \geq 1$, we have that $\tau_i(a) = t$ where t is the type of the attribute with the name a_n.

*(3) For each $a \in A^* \backslash D(y_i)$, we have $\tau_i(a) =$ **nil**.*

EXAMPLE 4.29 ---

Consider the frame taxonomy consisting of the two class frames from the previous example. The set K of classes in this taxonomy equals

$$K = \{\mathbf{nil}, blood\text{-}vessel, cubic\text{-}measure, \mathbf{real}, \{mm^3, cm^3, dm^3, m^3\}\}$$

Let $D(blood\text{-}vessel)$ be the set of attribute sequences associated with the *blood-vessel* class as in the previous example. For this class, the type function $\tau_1 : A^* \to K$ is defined by

$\tau_1(\epsilon) = blood\text{-}vessel$
$\tau_1(volume) = cubic\text{-}measure$
$\tau_1(volume : size) = \mathbf{real}$
$\tau_1(volume : unit) = \{mm^3, cm^3, dm^3, m^3\}$
$\tau_1(a) = \mathbf{nil}$ for all $a \in A^* \backslash D(blood\text{-}vessel)$

Let $D(cubic\text{-}measure)$ be the domain for the class frame *cubic-measure*. For this class, the type function $\tau_2 : A^* \to K$ is defined as follows:

$\tau_2(\epsilon) = \textit{cubic-measure}$
$\tau_2(\textit{size}) = \mathbf{real}$
$\tau_2(\textit{iunit}) = i\{mm^3, cm^3, dm^3, m^3\}$
$\tau_2(a) = \mathbf{nil}$ for all $a \in A^*\backslash D(\textit{cubic-measure})$

We now are ready to consider a type semantics for the superclass links in a taxonomy. We introduce the notion of a subtype.

Definition 4.7 *Let y_1 and y_2 be two class frames in a tree-shaped frame taxonomy. Let A^* be the set of attribute sequences in the taxonomy. Furthermore, let $D(y_i)$ be the set of attribute sequences associated with the class frame y_i, $i = 1, 2$. Now, let τ_i be the type function associated with y_i. We say that y_1 is a* subtype *of y_2, denoted by is $y_1 \leq y_2$, if the following two properties hold:*

(1) $D(y_2) \subseteq D(y_1)$;

(2) For each attribute sequence $a \in A^*$, we have that $\tau_1(a) \leq \tau_2(a)$.

We say that a taxonomy is *correctly typed* if the superclass links in the taxonomy satisfy the properties of the relation $\leq$ from the previous definition. Note that we now have that the meaning of the subtype relation can be described in terms of set inclusion $\subseteq$: we associate with each type t a subset $I(t)$ of elements from a domain of discourse U, such that if for two types t_1 and t_2 we have that $t_1 \leq t_2$, then we have that the property $I(t_1) \subseteq I(t_2)$ holds.

EXAMPLE 4.30

Consider the second example from this section once more. We have that the *artery* class is a superframe of the *blood-vessel* class. The set of attribute sequences associated with the class *blood-vessel* is equal to the set

$$D(\textit{blood-vessel}) = \{\epsilon, \textit{blood}, \textit{wall}\}$$

the domain for the *artery* class is equal to

$$D(\textit{artery}) = \{\epsilon, \textit{blood}, \textit{wall}, \textit{wall-thickness}\}$$

So, the first condition in the preceding definition is satisfied since we have that $D(\textit{blood-vessel}) \subseteq D(\textit{artery})$. The type function $\tau_1 : A^* \to K$ associated with the class *artery* is defined by:

$\tau_1(\epsilon) = \textit{artery}$
$\tau_1(\textit{blood}) = \{\textit{oxygen-rich}, \textit{oxygen-poor}\}$
$\tau_1(\textit{wall}) = \{\textit{muscular}, \textit{mixed}\}$
$\tau_1(a) = \mathbf{nil}$ for all $a \in A^*\backslash D(\textit{artery})$

The type function $\tau_2 : A^* \to K$ associated with the class *blood-vessel* is defined by:

$\tau_2(\epsilon) = \textit{blood-vessel}$
$\tau_2(\textit{blood}) = \{\textit{oxygen-rich}, \textit{oxygen-poor}\}$
$\tau_2(\textit{wall}) = \{\textit{muscular}, \textit{fibrous}, \textit{mixed}\}$
$\tau_2(a) = \mathbf{nil}$ for each $a \in A^*\backslash D(\textit{blood-vessel})$

The reader can easily verify that $\tau_1(a) \leq \tau_2(a)$ for each $a \in A^*$. We conclude that the frame taxonomy is correctly typed.

4.3 Frames and multiple inheritance

So far, we have only dealt with tree-shaped frame taxonomies and single inheritance. In this section, we introduce more general frame taxonomies. The frame formalism that has been defined in Section 4.2.1 is extended by admitting in class frames more than one class name in the superclass link field; this way, it is possible for a class to have more than one superclass. The graphical representation of such a taxonomy then takes the form of a general directed graph instead of a tree. In the following, we shall restrict the discussion to acyclic directed graphs, because trivial cycles obtained from the reflexivity property of the subclass relation, which are the only cycles that can be constructed when the superclass relation is viewed as a partial order, will not be explicitly indicated. The inheritance algorithm associated with graph-shaped frame taxonomies is known as *multiple* inheritance. In discussing multiple inheritance, we assume that more than one value for an attribute may have been specified in a frame taxonomy, that is, we allow for exceptions; however, we shall restrict the discussion to value facets of attributes only. Note that the theory developed in this section should hold for tree-shaped frame taxonomies and general graph-shaped taxonomies alike, since the former is just a special case of the latter.

4.3.1 Multiple inheritance of attribute values

Multiple inheritance differs from single inheritance mainly in the way it handles attributes occurring more than once in a taxonomy with different values. As in Section 4.2.2, we shall call such attribute values *exceptions*. Recall that when exceptions have been specified, conflicting information may be derived due to the inheritance of mutually exclusive values. In Section 4.2.2, however, we have seen that the problem of handling exceptions is easily solved in the case of single inheritance in a tree-shaped taxonomy: inheritance is taken as a process for finding a value for an attribute that starts with a given vertex in the tree, which moves along the branches of the tree towards the root and stops as soon as a value for the attribute of concern has been obtained. This algorithm always finds at most one attribute value. Unfortunately, the problem is much more complicated in the case of multiple inheritance in a general graph-shaped taxonomy. The algorithm for multiple inheritance in graph-shaped taxonomies in which exceptions occur has to incorporate a method for explicitly deciding which value of an attribute is to be preferred; we speak of *multiple inheritance with exceptions*. The main part of this section will be devoted to the development of such an algorithm for multiple inheritance with exceptions.

To start with, some new notions and notational conventions are introduced, including a more compact notation for the representation of frame information which has a stronger mathematical flavour than the implementation-oriented syntax introduced in the preceding section. From now on, $K = \{y_1, y_2, \ldots, y_n\}$, $n \geq 0$, will denote a fixed set of class frames, and $I = \{x_1, x_2, \ldots, x_m\}$, $m \geq 0$, will denote a fixed set of instance frames; the sets I and K are disjoint. The set of frames F is equal to $I \cup K$.

In a frame taxonomy, the superclass links are viewed as members of a relation between class frames. In the following definition we indicate that this relation may be viewed as a partial order.

Definition 4.8 *Let K denote the fixed set of class frames. The* subclass relation $\leq$ *is a binary relation on K, that is $\leq \subseteq K \times K$, that defines a partial order on the set K. For a pair $(x, y) \in \leq$, denoted by $x \leq y$, it is said that x* is a subclass of *y.*

Recall that, since the subclass relation $\leq$ defines a partial order on the set of class frames K, it satisfies the properties mentioned in Section 4.1.1.

The instance-of links are viewed as members of a relation between the set of instance frames I and the set of class frame K. We assume that an instance belongs to exactly one class. The instance-of links therefore are best formalized by means of a function.

Definition 4.9 *Let I denote the fixed set of instance frames and K the fixed set of classes. The* instance-of function $\ll: I \rightarrow K$ *is a mapping from I to K. In the sequel, we shall denote $\ll(x) = y$ by $x \ll y$; we say that x* is an instance of *y.*

The subclass relation and the instance-of function introduced in the two preceding definitions only describe reference information. The following definition introduces another relation meant to arrive at a full language for the specification of frame information.

Definition 4.10 *Let F be the set of frames such that $F = I \cup K$, where I is the set of instance frames in F, and K the set of class frames in F. Let A be a fixed set of attribute names and let C be a fixed set of constants. Then, a triple $(x, a, c) \in F \times A \times C$, denoted by $x[a = c]$, is called an* attribute-value specification. *An* attribute-value relation Θ *is a ternary relation on F, A and C, that is, $\Theta \subseteq F \times A \times C$.*

In the previous definition we have explicitly specified a set of constants C. Note, however, that this set of constants may be identified with the set of instances I, as we have done in the preceding sections. An attribute-value specification $x[a = c]$ expresses that in the frame x the attribute a has the constant value c. The notions introduced in the foregoing definitions are now used to formally define a frame taxonomy.

Definition 4.11 *Let I be the set of instances and K the set of classes. Furthermore, let A be the set of attribute names and C the set of constants. I, K, A and C are disjoint. Now, let N be the quadruple $N = (I, K, A, C)$. Furthermore, let the relations $\leq$ and Θ, and the function $\ll$ be defined as above. Then, a* taxonomy *T is a quadruple $T = (N, \Theta, \ll, \leq)$.*

We give an example of the frame formalism we have just defined and its relation with the frame formalism introduced in Section 4.2.1.

EXAMPLE 4.31

Consider the information specified in the following three classes represented in the frame formalism from Section 4.2.1:

```
class blood-vessel is
   superclass nil;
   contains = blood-fluid
end
```

```
class $artery$ is
    superclass $blood\text{-}vessel$;
    $blood = oxygen\text{-}rich$;
    $wall = muscular$
end

class $vein$ is
    superclass $blood\text{-}vessel$;
    $wall = fibrous$
end

instance $aorta$ is
    instance-of $artery$;
    $diameter = 2.5$
end
```

In the specified taxonomy, we have that $I = \{aorta\}$ is the set of instance frames and that $K = \{artery, vein, blood\text{-}vessel\}$ is the set of classes. Furthermore, we have that $A = \{contains, blood, wall, diameter\}$, and $C = \{blood\text{-}fluid, oxygen\text{-}rich, muscular, fibrous, 2.5\}$. We have the following set of attribute-value specifications:

$$\begin{aligned}\Theta = \{&blood\text{-}vessel[contains = blood\text{-}fluid],\\ &artery[blood = oxygen\text{-}rich],\\ &artery[wall = muscular],\\ &vein[wall = fibrous],\\ &aorta[diameter = 2.5]\}\end{aligned}$$

The function $\ll$ and the relation $\leq$ are defined by

$$\begin{aligned}&aorta \ll artery\\ &artery \leq blood\text{-}vessel\\ &vein \leq blood\text{-}vessel\end{aligned}$$

Now, $T = (N, \Theta, \ll, \leq)$ is the taxonomy shown above, this time represented using our new formalism.

Just as before, a taxonomy $T = (N, \Theta, \ll, \leq)$ can be represented graphically by means of an acyclic directed graph in which the vertices represent the frames in I and K, and the arcs represent the relation $\leq$ and the function $\ll$. A vertex is assumed to have an internal structure representing the collection of attribute-value specifications associated with the frame by the relation Θ. In the graphical representation, an attribute-value specification is depicted next to the vertex it belongs to; only the attribute and constant of an attribute-value specification are shown. We indicate the relation $\leq$ by means of a pulled arrow; and the function $\ll$ will be depicted by means of a dashed arrow. In the graphical representation of a taxonomy, arcs expressing the reflexivity and transitivity of the subclass relation will be left out in most cases. The omission of the arcs representing reflexivity, has no effect on the inheritance of attribute values, and is therefore permitted. However, leaving out arcs representing the transitivity property of the subclass relation, is one of the causes of problems concerning the inheritance of mutually exclusive attribute value, since this may alter the meaning of a taxonomy. Most of the remainder of this section is therefore devoted to an investigation of the consequences of this decision. Figure 4.15 shows the taxonomy from the previous example.

The relation $\leq$ defined above is now taken as the basis for reasoning with frames. We shall define so-called inheritance chains for the representation of the reasoning process that takes place in a frame taxonomy. These chains will constitute our principal device for dealing with exceptions in multiple inheritance. In the following two definitions the syntactic form of such inheritance chains and a procedure for their construction is presented. We shall first be concerned with inheritance of attribute values for classes only; later on we turn to inheritance of attribute values for instances.

Definition 4.12 *Let $T = (N, \Theta, \ll, \leq)$ be a taxonomy having the usual meaning, where $N = (I, K, A, C)$. An* inheritance chain *in T is an expression having one of the following forms:*

$$y_1 \leq \ldots \leq y_n$$
$$y_1 \leq \ldots \leq y_n[a = c]$$

where $y_i \in K$, $i = 1, \ldots, n$, $n \geq 1$, are class frames, and $y_n[a = c] \in \Theta$ is an attribute-value specification.

Note that attribute-value specifications are allowed only in isolation, or at the end of an inheritance chain. Furthermore, we observe that inheritance chains of the form $y_1 \leq \ldots \leq y_n$ are just another way of characterizing the subclass relation, obtained from its satisfying the properties of reflexivity and transitivity. Although we allow inheritance chains in which the reflexivity of the subclass relation $\leq$ is exploited, we shall not show such chains in our examples since they do not contribute to the notions we want to illustrate.

The set of all possible inheritance chains in a given frame taxonomy is constructed as described in the next definition.

Definition 4.13 *Let $T = (N, \Theta, \ll, \leq)$ be a taxonomy where $N = (I, K, A, C)$. The set Ω_T of inheritance chains in T is defined as follows:*

- *For each $y \in K$, we have $y \in \Omega_T$.*
- *For each $y[a = c] \in \Theta$ where $y \in K$, we have $y[a = c] \in \Omega_T$.*

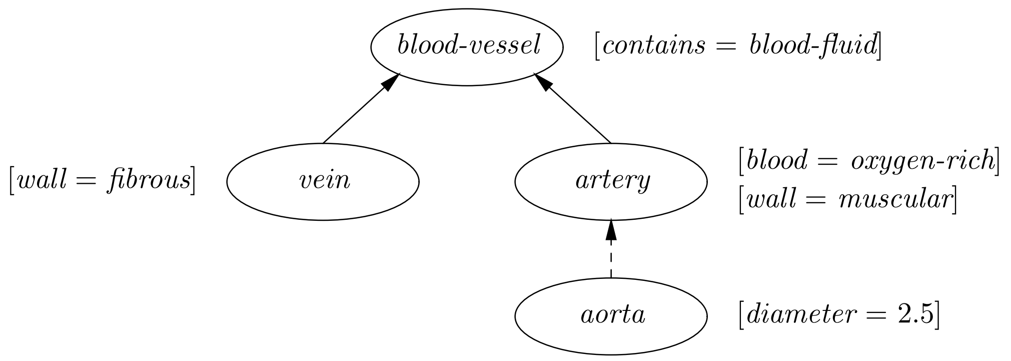


Figure 4.15: A taxonomy consisting of three classes and one instance.

- *For each pair* $(y_1, y_2) \in \leq$ *we have* $y_1 \leq y_2 \in \Omega_T$.
- *For each* $y_1 \leq \ldots \leq y_k \in \Omega_T$ *and* $y_k \leq \ldots \leq y_n \in \Omega_T$, $1 \leq k \leq n$, $n \geq 1$, *where* $y_i \in K$, $i = 1, \ldots, n$, *we have that* $y_1 \leq \ldots \leq y_n \in \Omega_T$.
- *For each* $y_1 \leq \ldots \leq y_n \in \Omega_T$ *and* $y_n[a = c] \in \Omega_T$, *where* $y_i \in K$, $i = 1, \ldots, n$, $n \geq 1$, *we have that* $y_1 \leq \ldots \leq y_n[a = c] \in \Omega_T$.

EXAMPLE 4.32

Consider the taxonomy $T = (N, \Theta, \ll, \leq)$ in which

$$\begin{aligned}
I &= \{aorta\}\\
K &= \{large\text{-}artery, artery, blood\text{-}vessel\}\\
\Theta &= \{aorta[diameter = 2.5], artery[wall = muscular], large\text{-}artery[mean\text{-}pressure = 100],\\
&\quad blood\text{-}vessel[contains = blood\text{-}fluid]\}
\end{aligned}$$

The function $\ll$ is defined by *aorta* $\ll$ *large-artery*, and the relation $\leq$ is defined by *large-artery* $\leq$ *artery* and *artery* $\leq$ *blood-vessel*. Recall that inheritance chains in which the reflexivity of the subclass relation $\leq$ is used, will not be shown; to give an impression of how such chains look like we show one of them:

artery $\leq$ *artery* $\leq$ *artery*

Now, the set of inheritance chains Ω_T consists of the following elements:

artery
large-artery
blood-vessel
artery[*wall* = *muscular*]
large-artery[*mean-pressure* = 100]
blood-vessel[*contains* = *blood-fluid*]
large-artery $\leq$ *artery*
large-artery $\leq$ *artery*[*wall* = *muscular*]
large-artery $\leq$ *artery* $\leq$ *blood-vessel*
large-artery $\leq$ *artery* $\leq$ *blood-vessel*[*contains* = *blood-fluid*]
artery $\leq$ *blood-vessel*
artery $\leq$ *blood-vessel*[*contains* = *blood-fluid*]

Inheritance chains are viewed as descriptions of which attribute-value specifications may *possibly* be inherited by the frames in the taxonomy. We shall see shortly that in multiple inheritance with exceptions certain combinations of attribute-value specifications when actually inherited represent contradictory information. Under suitable conditions, however, certain inheritance chains may be cancelled from the set of all inheritance chains in the taxonomy, thus preventing the occurrence of a contradiction. Before discussing this idea in further detail, we introduce the notion of the conclusion of an inheritance chain which is an explicit means for establishing which attribute-value specification may be inherited from the chain.

Definition 4.14 *Let $T = (N, \Theta, \ll<, \leq)$ be a taxonomy where $N = (I, K, A, C)$. Let Ω_T be the set of inheritance chains in T. The* conclusion $c(\omega)$ *of an inheritance chain $\omega \in \Omega_T$ is defined as follows:*

- *For each $\omega \equiv y_1 \leq \ldots \leq y_n[a = c]$, we have that $c(\omega) = y_1[a = c]$.*
- *For all other ω, we have that $c(\omega)$ is not defined.*

The conclusion set $C(\Omega_T)$ *of Ω_T is defined as the set of conclusions of all elements from Ω_T, that is, $C(\Omega_T) = \{c(\omega) \mid \omega \in \Omega_T\}$.*

When the attribute-value specification $z[a = c]$ is obtained as the conclusion of an inheritance chain, we say that the value c of the attribute a has been *inherited* by z.

EXAMPLE 4.33

Consider again the set Ω_T of inheritance chains from the preceding example. The conclusion set $C(\Omega_T)$ of Ω_T then consists of the following attribute-value specifications:

large-artery[*mean-pressure* = 100]
large-artery[*wall* = *muscular*]
large-artery[*contains* = *blood-fluid*]
artery[*wall* = *muscular*]
artery[*contains* = *blood-fluid*]
blood-vessel[*contains* = *blood-fluid*]

The conclusion set $C(\Omega_T)$ of a given set of inheritance chains Ω_T may contain attribute-value specifications which only differ in their specified constant. We have already encountered the notion of exception and its related problems in Section 4.2.2. In the following example, we restate the problem in terms of inheritance chains.

EXAMPLE 4.34

In the foregoing, it has frequently been pointed out that the left and right pulmonary arteries have much in common with arteries except that they contain oxygen-poor instead of oxygen-rich blood. Now, consider the set Ω of inheritance chains containing, among other ones, the following two chains:

pulmonary-artery[*blood* = *oxygen-poor*]
pulmonary-artery $\leq$ *artery*[*blood* = *oxygen-rich*]

The conclusion set constructed from Ω contains at least the following two attribute-value specifications:

pulmonary-artery[*blood* = *oxygen-poor*]
pulmonary-artery[*blood* = *oxygen-rich*]

Clearly, if a sensible meaning is to be associated with the frame formalism, only one of these conclusions should be satisfied.

We call a conclusion set $C(\Omega_T)$ inconsistent if it contains contradictory information such as in the previous example. In the following definition the notions of consistency and inconsistency of a conclusion set are defined more formally.

Definition 4.15 *Let $T = (N, \Theta, \ll, \leq)$ be a taxonomy where $N = (I, K, A, C)$. Let Ω_T be the set of inheritance chains in T. Furthermore, let $C(\Omega_T)$ be the conclusion set of Ω_T. The conclusion set $C(\Omega_T)$ is called* inconsistent *if it contains attribute-value specifications $y[a = c_1]$ and $y[a = c_2]$, $c_1, c_2 \in C$, such that $c_1 \neq c_2$. Otherwise, the conclusion set $C(\Omega_T)$ is said to be* consistent.

Inconsistency of the conclusion set of a taxonomy indicates that inheritance in the taxonomy is not defined uniquely: only if the conclusion set is consistent, the instances of the taxonomy inherit unambiguous information from the classes they belong to. We now consider inheritance of attribute values for instances in more detail. Informally speaking, the attribute-value specifications that hold for an instance of a specific class frame are the attribute-value specifications explicitly specified in the instance itself supplemented with the attribute-value specifications holding for the class it belongs to that do not contradict the attribute-value specifications from the instance. This is defined more formally below.

Definition 4.16 *Let $T = (N, \Theta, \ll, \leq)$ be a taxonomy where $N = (I, K, A, C)$. Let Ω_T be the set of inheritance chains in T and let $C(\Omega_T)$ be the conclusion set of Ω_T. For each instance frame $x \in I$, the set $e_C(x)$ is defined by $e_C(x) = \{x[a = c] \mid x[a = c] \in \Theta\} \cup \{x[a = c] \mid x \ll y, y \in K, y[a = c] \in C(\Omega_T)$ and for all $c \neq d$, $x[a = d] \notin \Theta\}$ if $C(\Omega_T)$ is consistent; $e_C(x)$ is undefined otherwise. The* extension of Ω_T, *denoted by $E_C(\Omega_T)$, is defined by*

$$E_C(\Omega_T) = \bigcup_{x \in I} e_C(x)$$

if $C(\Omega_T)$ is consistent; $E_C(\Omega_T)$ is undefined otherwise.

EXAMPLE 4.35

Consider the taxonomy T from the first example of this section once more. Let Ω_T be the set of inheritance chains in T and let $C(\Omega_T)$ be the conclusion set of Ω_T. The extension of Ω_T is equal to the following set of attribute-value specifications:

$$\begin{aligned} E_C(\Omega_T) = \{ & \mathit{aorta}[\mathit{contains} = \mathit{blood\text{-}fluid}], \\ & \mathit{aorta}[\mathit{mean\text{-}pressure} = 100], \\ & \mathit{aorta}[\mathit{wall} = \mathit{muscular}], \\ & \mathit{aorta}[\mathit{diameter} = 2.5]\} \end{aligned}$$

A taxonomy that is inconsistent in the sense of its having an inconsistent conclusion set can sometimes be ‘made’ consistent by cancelling some of the inheritance chains from the set of inheritance chains in the taxonomy by using knowledge concerning the hierarchical ordering of the frames. As a consequence, certain conclusions are cancelled from the conclusion set of the taxonomy as well, thereby preventing the occurrence of some contradictory attribute values. Note that this way non-monotonic reasoning is introduced within the frame formalism.

For cancelling inheritance chains, we shall exploit the notion of an intermediary, which is introduced in the following definition.

Definition 4.17 *Let $T = (N, \Theta, \ll, \leq)$ be a taxonomy where $N = (I, K, A, C)$. Let Ω_T be the set of inheritance chains in T. A class $y \in K$ is called an* intermediary *to an inheritance chain $y_1 \leq \ldots \leq y_n \in \Omega_T$, $y_i \in K$, $i = 1, \ldots, n$, $n \geq 1$, if one of the following conditions is satisfied:*

- *We have $y = y_i$ for some i, $1 \leq i \leq n$.*
- *There exists a chain $y_1 \leq \ldots \leq y_p \leq z_1 \leq \ldots \leq z_m \leq y_q \in \Omega_T$, for some p, q, $1 \leq p \leq q \leq n$, where $z_j \neq y_i$, $i = 1, \ldots, n$, $z_j \in K$, $j = 1, \ldots, m$, $m \geq 1$, such that $y = z_k$, for some k, $1 \leq k \leq m$.*

EXAMPLE 4.36

Consider the taxonomy $T = (N, \Theta, \ll, \leq)$, where $I = \varnothing$, $K = \{$*blood-vessel*, *artery*, *oxygen-poor-artery*, *pulmonary-artery*$\}$, Θ is empty, and the relation $\leq$ is defined by

pulmonary-artery $\leq$ *oxygen-poor-artery*
pulmonary-artery $\leq$ *artery*
artery $\leq$ *blood-vessel*
oxygen-poor-artery $\leq$ *artery*

The graphical representation of the taxonomy is shown in Figure 4.16. The set of inheritance chains in T contains, among other ones, the following two chains:

pulmonary-artery $\leq$ *artery* $\leq$ *blood-vessel*
pulmonary-artery $\leq$ *oxygen-poor-artery* $\leq$ *artery*

It will be evident that the class *oxygen-poor-artery* is an intermediary to both chains.

Figure 4.16 introduced in the foregoing example is useful for gaining some intuitive feeling concerning the notion of an intermediary.

We shall see that intermediaries may be applied for solving part of the problem of multiple inheritance with exceptions. We take a closer look at the figure. It seems as if the arc between the vertices *pulmonary-artery* and *artery*, an arc resulting from the transitivity property of the subclass relation, is redundant, since all attribute-value specification from the classes *artery* and *blood-vessel* can be inherited for *pulmonary-artery* via the vertex *oxygen-poor-artery*. Therefore, the removal of this arc from the taxonomy should not have any influence on the result of multiple inheritance. Whether or not this is true is, of course, dependent on our formalization of multiple inheritance. Therefore, let us investigate whether the notion of conclusion set defined in the foregoing renders a suitable means for dealing with exceptions. We do so by means of an example.

EXAMPLE 4.37

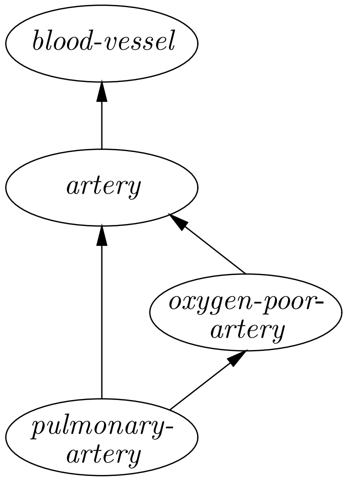


Figure 4.16: A taxonomy with an intermediary.

Consider Figure 4.16 once more. Figure 4.17 shows the taxonomy from Figure 4.16 after removal of the seemingly redundant arc. Now, suppose that the following attribute-value specifications are given:

oxygen-poor-artery[*blood* = *oxygen-poor*]
artery[*blood* = *oxygen-rich*]

Furthermore, suppose that no attribute-value specifications have been given for *pulmonary-artery*. In the taxonomy shown in Figure 4.17, the frame *pulmonary-artery* inherits only the value *oxygen-poor* for the attribute *blood*; note that this is a consequence of the way exceptions are handled in tree-shaped taxonomies. However, in Figure 4.16 the frame *pulmonary-artery* inherits both values *oxygen-poor* and *oxygen-rich* for the attribute *blood*, leading to an inconsistent conclusion set. The conclusion set of the taxonomy in Figure 4.16 therefore differs from the one obtained for the taxonomy shown in Figure 4.17, using the algorithm for single inheritance with exceptions discussed in Section 4.2.2s in the last case.

It turns out that a conclusion set only reveals the presence of exceptions in a taxonomy. We shall see that the notion of an intermediary is more useful in dealing with exceptions in multiple inheritance. In Figure 4.16 we have that the class *oxygen-poor-artery* lies in between the classes *pulmonary-artery* and *artery*, and is an intermediary to the inheritance chains in which the class *pulmonary-artery* and either or both the classes *artery* and *oxygen-poor-artery* occur. As we have suggested before, by means of intermediaries some of the inheritance chains may be cancelled rendering a different set of conclusions of the taxonomy. Such cancellation of inheritance chains is called *preclusion* and is defined more formally below.

Definition 4.18 *Let* $T = (N, \Theta, \ll, \leq)$ *be a taxonomy where* $N = (I, K, A, C)$. *Let* Ω_T *be the set of inheritance chains in* T. *A chain* $y_1 \leq \ldots \leq y_n[a = c_1] \in \Omega_T$, *where* $n \geq 1$, *is said to* preclude *a chain* $y_1 \leq \ldots \leq y_m[a = c_2] \in \Omega_T$, *where* $m \geq 1$, $m \neq n$, *and* $c_1, c_2 \in C$ *with* $c_1 \neq c_2$, *if* y_n *is an intermediary to* $y_1 \leq \ldots \leq y_m$.

EXAMPLE 4.38

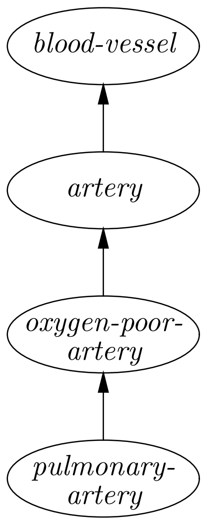


Figure 4.17: The taxonomy after removal of the redundant arc.

Consider the set Ω_T of inheritance chains consisting of the following elements:

ω_1: *pulmonary-artery* $\leq$ *oxygen-poor-artery*
ω_2: *pulmonary-artery* $\leq$ *artery*
ω_3: *pulmonary-artery* $\leq$ *oxygen-poor-artery* $\leq$ *artery*
ω_4: *pulmonary-artery* $\leq$ *oxygen-poor-artery*[*blood* = *oxygen-poor*]
ω_5: *pulmonary-artery* $\leq$ *artery*[*blood* = *oxygen-rich*]
ω_6: *pulmonary-artery* $\leq$ *oxygen-poor-artery* $\leq$ *artery*[*blood* = *oxygen-rich*]

The reader can easily verify that the inheritance chain ω_4 precludes both chains ω_5 and ω_6 since *oxygen-poor-artery* is an intermediary to the chains ω_2 and ω_3.

The notion of preclusion is used for introducing a new type of conclusion set of a set of inheritance chains.

Definition 4.19 *Let $T = (N, \Theta, \ll, \leq)$ be a taxonomy. Let Ω_T be the set of inheritance chains in T. An inheritance chain $\omega \in \Omega_T$ is said to be* inheritable *if there exists no other inheritance chain $\omega' \in \Omega_T$ which precludes ω. The set of conclusions of all inheritable chains $\omega \in \Omega_T$ is called the* inheritable conclusion set *of Ω_T and is denoted by $H(\Omega_T)$.*

From now on we take the notions of consistency and inconsistency defined for a conclusion set to apply to inheritable conclusion sets as well. We give some (more abstract) examples.

EXAMPLE 4.39

Consider the taxonomy $T = (N, \Theta, \ll, \leq)$ where $I = \{x\}$ is the set of instances and $K = \{y_1, y_2, y_3\}$ is the set of classes; furthermore, Θ is defined by

$$\Theta = \{x[a_1 = c_1],\ y_1[a_2 = c_2],\ y_2[a_3 = c_3],\ y_3[a_3 = c_4]\}$$

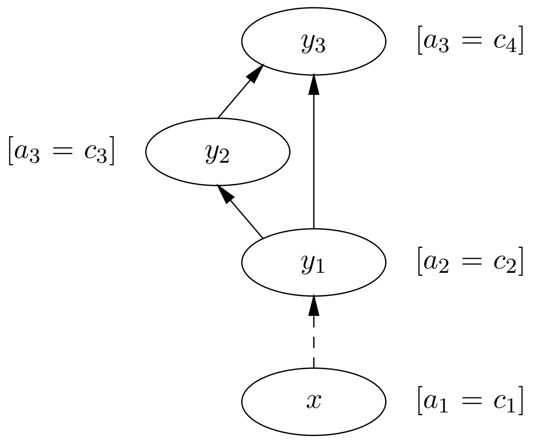


Figure 4.18: A taxonomy having a consistent inheritable conclusion set.

Herein a_1, a_2, and a_3 are distinct attribute names and c_1, c_2, c_3, and c_4 are different constants. In addition, the relation $\leq$ is defined by $y_1 \leq y_2$, $y_1 \leq y_3$ and $y_2 \leq y_3$; the function $\ll$ is defined by $x \ll y_1$. This taxonomy is depicted in Figure 4.18. The set of inheritance chains Ω_T consists of the following elements:

1. y_1
2. y_2
3. y_3
4. $y_1[a_2 = c_2]$
5. $y_2[a_3 = c_3]$
6. $y_3[a_3 = c_4]$
7. $y_1 \leq y_2$
8. $y_1 \leq y_3$
9. $y_2 \leq y_3$
10. $y_1 \leq y_2 \leq y_3$
11. $y_1 \leq y_2[a_3 = c_3]$
12. $y_1 \leq y_3[a_3 = c_4]$
13. $y_2 \leq y_3[a_3 = c_4]$
14. $y_1 \leq y_2 \leq y_3[a_3 = c_4]$

The conclusion set of Ω_T is equal to $C(\Omega_T) = \{y_1[a_2 = c_2], y_1[a_3 = c_3], y_1[a_3 = c_4], y_2[a_3 = c_3], y_2[a_3 = c_4], y_3[a_3 = c_4]\}$. Note that $C(\Omega_T)$ is inconsistent.

Consider the set Ω_T once more. We investigate which attribute-value specifications are in the inheritable conclusion set. As stated in the previous definition, an inheritance chain $\omega \in \Omega_T$ is inheritable if it is not precluded by any other chain from Ω_T. Since only a chain ending in an attribute-value specification can be precluded by another chain also ending in an attribute-value specification, examination of chains of such a form will suffice. So, we consider the following inheritance chains:

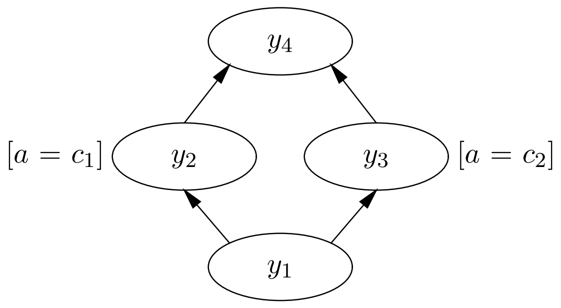


Figure 4.19: A taxonomy having an inconsistent inheritable conclusion set.

4. $y_1[a_2 = c_2]$
5. $y_2[a_3 = c_3]$
6. $y_3[a_3 = c_4]$
11. $y_1 \leq y_2[a_3 = c_3]$
12. $y_1 \leq y_3[a_3 = c_4]$
13. $y_2 \leq y_3[a_3 = c_4]$
14. $y_1 \leq y_2 \leq y_3[a_3 = c_4]$

Chain 12 is precluded by chain 11 because y_2 is an intermediary to $y_1 \leq y_3$. Furthermore, inheritance chain 13 is precluded by 5. The reader may verify that chain 14 is precluded by 11 as well. The inheritable conclusion set $H(\Omega_T)$ of Ω_T is therefore equal to $H(\Omega_T) = \{y_1[a_2 = c_2], y_1[a_3 = c_3], y_2[a_3 = c_3], y_3[a_3 = c_4]\}$. We conclude that the inheritable conclusion set is consistent.

In the next example, it will be shown that even if we apply preclusion it is still possible to obtain an inheritable conclusion set specifying contradictory information.

EXAMPLE 4.40

Consider the taxonomy $T = (N, \Theta, \ll, \leq)$ where the set I is empty, and $K = \{y_1, y_2, y_3, y_4\}$. The attribute-value relation Θ is defined by $\Theta = \{y_2[a = c_1], y_3[a = c_2]\}$ where a is an attribute, and c_1 and c_2 are distinct constants. Furthermore, the relation $\leq$ is defined by the following elements:

$y_1 \leq y_2$
$y_1 \leq y_3$
$y_2 \leq y_4$
$y_3 \leq y_4$

This taxonomy is depicted graphically in Figure 4.19. The inheritable conclusion set of the set of inheritance chains in this taxonomy is equal to $H(\Omega_T) = \{y_1[a = c_1], y_1[a = c_2], y_2[a = c_1], y_3[a = c_2]\}$. Note that $H(\Omega_T)$ is inconsistent.

From the foregoing example we have that the application of multiple inheritance using preclusion may still lead to the derivation of contradictory information. From now on, such a taxonomy will be called inconsistent.

Definition 4.20 *Let $T = (N, \Theta, \ll, \leq)$ be a taxonomy, and let Ω_T be the set of inheritance chains in T. Furthermore, let $H(\Omega_T)$ be the inheritable conclusion set obtained from Ω_T. The taxonomy T is said to be* consistent *if $H(\Omega_T)$ is consistent; otherwise T is said to be* inconsistent*.*

We conclude this section by introducing the refined notion of an inheritable extension of a set of inheritance chains.

Definition 4.21 *Let $T = (N, \Theta, \ll, \leq)$ be a taxonomy. Let Ω_T be the set of inheritance chains in T and let $H(\Omega_T)$ be the inheritable conclusion set of Ω_T. For each instance frame $x \in I$, the set $e_H(x)$ is defined by $e_H(x) = \{x[a = c] \mid x[a = c] \in \Theta\} \cup \{x[a = c] \mid x \ll y, y \in K, y[a = c] \in H(\Omega_T)$ and for all $c \neq d$, $x[a = d] \notin \Theta\}$ if $H(\Omega_T)$ is consistent; $e_H(x)$ is undefined otherwise. The* inheritable extension of Ω_T*, denoted by $E_H(\Omega_T)$, is defined by*

$$E_H(\Omega_T) = \bigcup_{x \in I} e_H(x)$$

if $H(\Omega_T)$ is consistent; $E_H(\Omega_T)$ is undefined otherwise.

EXAMPLE 4.41

Consider the taxonomy T from the second last example once more. The inheritable extension of Ω_T equals $E_H(\Omega_T) = \{x[a_1 = c_1], x[a_2 = c_2], x[a_3 = c_3]\}$.

4.3.2 Subtyping in graph-shaped taxonomies

In Section 4.2.4 we have discussed the subject of subtyping in tree-shaped frame taxonomies. In the present section, we extend the theory developed in that section to graph-shaped taxonomies. Recall that the subtype relation $\leq$ defines a partial order on a set of types. Before we treat subtyping in graph-shaped taxonomies in more detail, we review some properties of partially ordered sets.

Definition 4.22 *Let $S = \{t_1, t_2, \ldots, t_n\}$, $n \geq 1$, be a set on which we have a partial order $\leq$. An* upper bound *to a subset $X \subseteq S$ is an element $v \in S$ such that for each $x \in X$ we have that $x \leq v$; the* least upper bound *to X is an upper bound u for which we have that $u \leq v$ for each upper bound v to X. Similarly, a* lower bound *to a subset $X \subseteq S$ is an element $m \in S$ such that for each $x \in X$ we have that $m \leq x$; the* greatest lower bound *to X is a lower bound l for which we have that $m \leq l$ for each lower bound m to X.*

There may be more than one lower or upper bound to a subset of a partially ordered set, or even none at all. However, if lower and upper bounds do exist, then the least upper bound and the greatest lower bound are unique.

EXAMPLE 4.42

Consider the following set $S = \{\{1\}, \{1,2\}, \{1,2,3\}, \{1,2,3,4\}, \{2,3\}, \{2,3,4\}\}$ having elements which again are sets; on S we have the partial order induced by the set inclusion relation between its elements. We consider the subset $X = \{\{1,2,3\}, \{1,2,3,4\}\}$ of S. It will be evident that we have the following lower bounds to X: $\{1\}, \{1,2\}, \{1,2,3\}$ and $\{2,3\}$, since these sets are subsets of all elements of X. The greatest lower bound is equal to $\{1,2,3\}$: every other lower bound is a subset of this lower bound. In this example, we only have one upper bound, namely the set 1,2,3,4: each element of X is a subset of this upper bound. Note that this upper bound therefore at the same time is the least upper bound to X.

In Section 4.2.4 we have seen that type information in a correctly typed tree-shaped taxonomy may be considered to be a set of types which is ordered partially by the subtype relation $\leq$. For correctly typed graph-shaped taxonomies an even stronger property holds: in such taxonomies, the type information constitutes a so-called type lattice.

Definition 4.23 *A* type lattice *S is a set of types with a partial order $\leq$ such that for every two types $t_1, t_2 \in S$ there exist in S a least upper bound and a greatest lower bound. The greatest lower bound of t_1 and t_2 is denoted by $t_1 \wedge t_2$ and is usually called the* meet *of t_1 and t_2. Furthermore, the least upper bound of t_1 and t_2 is denoted by $t_1 \vee t_2$ and is usually called the* join *of t_1 and t_2. The* universal lower bound *of S, denoted by $\top$, is an element of S such that for each type $t \in S$ we have that $t \leq \top$. The* universal lower bound *of S, denoted by $\perp$, is an element of S such that for each type $t \in S$ we have that $\perp \leq t$.*

We shall see that for assigning a suitable meaning to type information, the types in a graph-shaped taxonomy should constitute a type lattice. First, we give an example of a type lattice.

EXAMPLE 4.43

Consider the graphical representation of a type lattice shown in Figure 4.20. This type lattice contains the type *vessel* having the type *blood-vessel* as a specialization. The *blood-vessel* type itself is considered to be subdivided into the types *vein* and *artery*. The type lattice furthermore specifies a kind of blood vessel, called *AV-anastomosis*, being a subtype of both the *vein* and *artery* type. Furthermore, the type lattice contains the type *blood* which describes the characteristics of blood. Two kinds of blood are distinguished: oxygen-rich blood and oxygen-poor blood having different characteristics (think for example of the range of the typical oxygen pressure in oxygen-rich and oxygen-poor blood, and of the colour of the blood which is typically darker in oxygen-poor than in oxygen-rich blood). The type *mixed-blood* describes blood having characteristics lying in between those of oxygen-poor and oxygen-rich blood. Now, note that from the type lattice we have *vein* $\vee$ *artery* = *blood-vessel*, in other words, the type *blood-vessel* is the meet of the types *vein* and *artery*. Note that the type *vessel* is an upper bound for the *vein* and *artery* types, but not the least one. Furthermore, the join of the types *vein* and *artery* is equal to the type *AV-anastomosis*, that is, we have *vein* $\wedge$ *artery* = *AV-anastomosis*.

Now recall from Section 4.2.4 that in a correctly typed tree-shaped taxonomy, the relation defined by the superclass links in the taxonomy coincides with the partial order $\leq$ on the set of

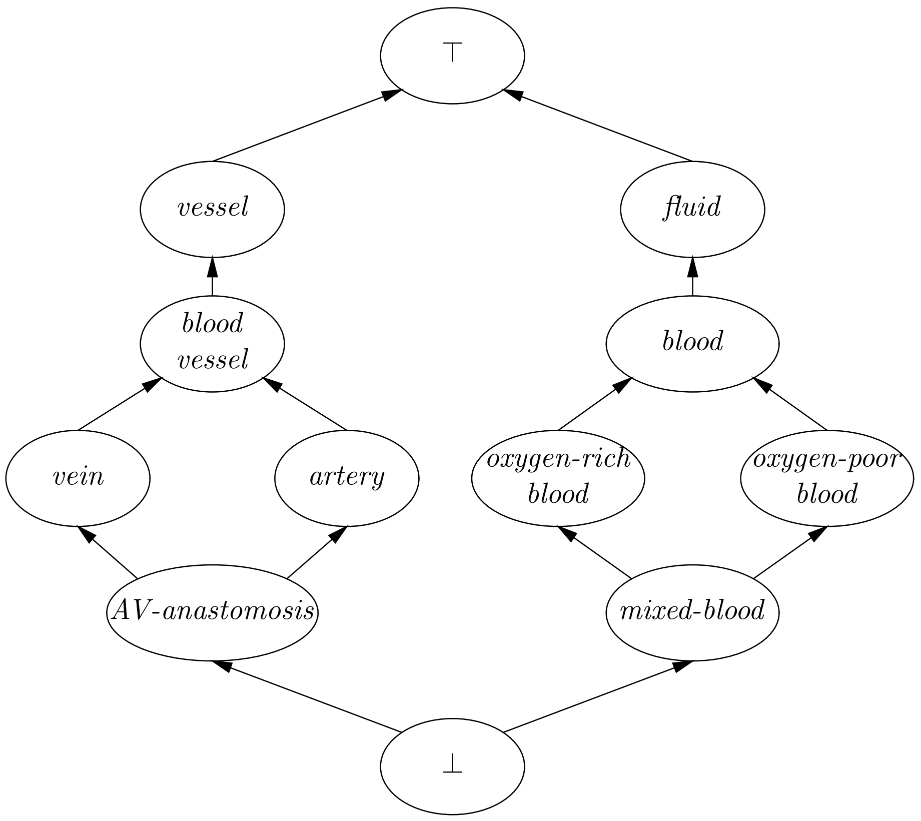


Figure 4.20: A type lattice.

types specified by it. Similarly, we have that we may view a graph-shaped taxonomy as a type lattice (including the predefined classes), and vice versa. Furthermore, recall the definitions of the notions of attribute sequence, domain, and type function, presented in Section 4.2.4. Informally speaking, we now take these notions to apply to graph-shaped taxonomies.

In Section 4.2.4 we interpreted types as sets. Here, we do so likewise: consider two types t_1 and t_2. We associated with these types the set $I(t_1)$ and $I(t_2)$, respectively, where $I(t_i) \subseteq U$, $i = 1, 2$, for some domain of discourse U. As we have mentioned before, in a graph-shaped frame taxonomy having the form of a lattice there exists a meet and a join for every pair of types. The set associated with the meet $t_1 \wedge t_2$ now has to satisfy the property $I(t_1 \wedge t_2) = I(t_1) \cap I(t_2)$; the set associated with the join $t_1 \vee t_2$ has to satisfy $I(t_1 \vee t_2) = I(t_1) \cup I(t_2)$. We conclude this section with an example giving a sketch of subtyping in a graph-shaped taxonomy.

EXAMPLE 4.44

Consider the type lattice from Figure 4.20 once more; we look upon it as a taxonomy. Suppose that the class frames *blood-vessel*, *vein*, *artery* and *AV-anastomosis* all contain an attribute-type specification concerning an attribute named *contains*, in which the specified type is one of the classes *blood*, *oxygen-rich-blood*, *oxygen-poor-blood*, and *mixed-blood*:

claiss *blood-vessel* **is**

```
        superclass vessel;
        contains : blood
    end

    class vein is
        superclass blood-vessel;
        contains : oxygen-poor-blood
    end

    class artery is
        superclass blood-vessel;
        contains : oxygen-rich-blood
    end

    class AV-anastomosis is
        superclass {artery,vein};
        contains : mixed-blood
    end
```

Furthermore, the classes *blood*, *oxygen-rich-blood*, *oxygen-poor-blood* and *mixed-blood* specify a single attribute named *colour*:

```
    class blood is
        superclass nil;
        colour : {blue, dark-red, red, bright-red}
    end

    class oxygen-rich-blood is
        superclass blood;
        colour : {dark-red, red, bright-red}
    end

    class oxygen-poor-blood is
        superclass blood;
        colour : {blue, dark-red, red}
    end

    class mixed-blood is
        superclass {oxygen-rich-blood,oxygen-poor-blood};
        colour : {dark-red, red}
    end
```

In the present example, the set of attributes A in the frame taxonomy is equal to $A = \{contains, colour\}$. Let A^* be the set of attribute sequences. We now have eight domains to consider; they are denoted by D_1 up to D_8 for the classes in the order shown above. The domains for the first four classes are all equal to $\{\epsilon, contains, contains : colour\}$. The domains D_5 to D_8 for the classes starting with the class *blood* are equal to $\{\epsilon, colour\}$. The reader can easily verify that the properties required for these domains for subtyping hold. For simplicity's sake we now only consider the type functions associated with the first four classes. The type functions associated with the classes *blood-vessel*, *vein*, *artery*, *AV-anastomosis* are denoted by τ_1 up to τ_4, respectively. Recall from Section 4.2.4 that we have to verify that for all $a \in A^*$ we have:

$$\begin{array}{l}\tau_2(a) \leq \tau_1(a)\\ \tau_3(a) \leq \tau_1(a)\\ \tau_4(a) \leq \tau_2(a)\\ \tau_4(a) \leq \tau_3(a)\end{array}$$

We only discuss some of these properties in detail. We start by noting that the type for the attribute *contains* in the class *blood-vessel* equals the join of the types *oxygen-rich-blood* and *oxygen-rich-blood* given for the classes *artery* and *vein*, respectively. Furthermore, the meet of the types *oxygen-poor-blood* and *oxygen-rich-blood* for the attribute *contains* in the classes *vein* and *artery* respectively is equal to *mixed-blood*. It will be evident that we have the following properties:

$$\begin{array}{l}\tau_3(\mathit{contains}) \leq \tau_1(\mathit{contains})\\ \tau_2(\mathit{contains}) \leq \tau_1(\mathit{contains})\\ \tau_4(\mathit{contains}) \leq \tau_3(\mathit{contains})\\ \tau_4(\mathit{contains}) \leq \tau_2(\mathit{contains})\end{array}$$

Furthermore, note that we not only have that *blood-vessel* = *vein* $\vee$ *artery*, but in addition that $\tau_1(\mathit{contains}) = \tau_2(\mathit{contains}) \vee \tau_3(\mathit{contains})$, since *blood* = *oxygen-poor-blood* $\vee$ *oxygen-rich-blood* that is: the subtyping of classes is extended from the classes to the attributes of the classes. Similarly, we have that $\tau_4(\mathit{contains}) = \tau_2(\mathit{contains}) \wedge \tau_3(\mathit{contains})$. Furthermore, we have that

$$\begin{array}{l}\tau_4(\mathit{contains} : \mathit{colour}) \leq \tau_3(\mathit{contains} : \mathit{colour}\\ \tau_4(\mathit{contains} : \mathit{colour}) \leq \tau_2(\mathit{contains} : \mathit{colour})\\ \tau_4(\mathit{contains} : \mathit{colour}) = \tau_2(\mathit{contains} : \mathit{colour}) \wedge \tau_3(\mathit{contains} : \mathit{colour})\end{array}$$

Checking the remaining properties is rather straightforward and is left to the reader. We conclude that the shown taxonomy is correctly typed.

4.4 Frames as a representation formalism

Frames (and semantic nets) provide a knowledge-representation formalism in which hierarchically structured knowledge can be specified in a natural way. Especially for the representation of knowledge of a descriptive nature, such as the knowledge concerning the cardiovascular system used in the examples in this chapter, the frame formalism appears to be highly suitable. The advantage of frames when compared to for example Horn clauses or production rules lies in the ease with which distinct types of knowledge can be distinguished and handled as such, and in the fact that an explicit hierarchical organization of knowledge is obtained.

In this chapter the only means of knowledge manipulation discussed was the method of inheritance. We stress once more that this knowledge-manipulation scheme in itself is not sufficient as an inference engine for all applications: is often turns out to be necessary to develop a more elaborate inference engine for the manipulation of frames in which inheritance only is part of a set of knowledge-manipulation methods. In chapter 7 we shall discuss some examples of such inference engines. For many non-trivial applications it will be necessary to use a enriched frame formalism, for example by procedural components. We have suggested before that an often employed hybrid knowledge-representation scheme is that of frames containing demons for invoking small sets of production rules. Instead of integrating frames with production rules, one could also think of a more declarative extension to the frame formalism for example by Horn clauses in combination with SLD resolution. This way a hybrid system is obtained that still has a clear declarative semantics. Most present-day frame-based systems are more alike programming languages than alike languages for knowledge-representation. A major disadvantage of many of these systems is the loss of a neat declarative semantics. Furthermore, working with these systems often requires a lot from the knowledge engineer such as a store of programming tricks.

Exercises

(4.1) Use the semantic net formalism to represent information concerning a problem domain you are familiar with. Try to define a neat semantics for the types of links you have used. What information can you derive from the net by property inheritance?

(4.2) Write a program that implements property inheritance in semantic nets. The program should be able to find for a specific object all properties that can be derived for it.

(4.3) Consider the following three frames:

```
class computer-program is
   superclass nil
end

class expert-system is
   superclass computer-program;
   synonym = knowledge-system;
contains = expert-knowledge
end

instance MYCIN is
   instance-of expert-system;
   implementer = Shortliffe
end
```

(a) Translate the knowledge specified in the three frames shown above into standard first-order logic with equality.

(b) Suppose that the following frame

```
instance Internist-I is
   instance-of expert-system;
   contains = medical-knowledge;
implementer = Pople
end
```

is added to the three frames given above. Discuss the problem that arises if we translate this frame together with the three frames shown above into standard first-order predicate logic with equality. Give a possible solution to the problem.

(4.4) Consider the following two frames:

```
class automobile is
   superclass nil;
   wheels = 4;
   seats = 4
end

instance Rolls-Royce is
   instance-of automobile;
max-velocity = enough
end
```

Translate the knowledge specified in these two frames into a semantic net representation.

(4.5) Consider the algorithm for single inheritance with exceptions in a tree-shaped taxonomy discussed in Section 4.2.2 once more. Extend this algorithm in a straightforward manner to render it applicable to graph-shaped taxonomies. Show by means of an example that your algorithm may produce results that are incorrect from a semantic point of view.

(4.6) Develop an algorithm for N-inheritance and implement your algorithm in a programming language (for example Prolog, Lisp or Haskell).

(4.7) Consider the following frame taxonomy $T = (N, \Theta, \ll, \leq)$ where $N = (I, K, A, C)$ We have that $K = \{x, y, z\}$ is the set of classes, $I = \{w\}$ is the set of instances and $\Theta = \{x[a = 1], y[b = 2], z[b = 4]\}$ is the set of attribute-value specifications. The relation $\leq$ and the function $\ll$ are defined by:

$x \leq y$
$z \leq x$
$z \leq y$
$w \ll z$

First, determine the set Ω_T of inheritance chains in T. Subsequently, compute the conclusion set and inheritable conclusion set of Ω_T. Is the taxonomy T consistent? If so, what is the inheritable extension of Ω_T?

(4.8) Consider the following taxonomy $T = (N, \Theta, \ll, \leq)$ where $N = (I, K, A, C)$. We have that $K = \{u, x, y, z\}$ is the set of classes, $I = \{w\}$ is the set of instances and $\Theta = \{u[a = 1], x[b = 2], y[c = 10], z[c = 20]\}$. The relation $\leq$ and the function $\ll$ are defined as follows:

$x \leq y$
$y \leq z$
$u \leq x$
$u \leq y$
$w \ll u$

Answer the same questions as in Exercise 4.7.

Chapter 5

Reasoning with Uncertainty

In the early 1960s, researchers in applied logic assumed that theorem provers were powerful and general enough to solve practical, real-life problems. In particular, the introduction of the resolution principle by Allan Robinson led to this conviction. By and by however it became apparent that the appropriateness of mathematical logic for solving practical problems was highly overrated. One of the complications with real-life situations is that the facts and experience necessary for solving the problems often are typified by a degree of uncertainty; moreover, often the available information is imprecise and insufficient for solving the problems. Yet human experts are able to form judgements and take decisions from uncertain, incomplete and contradictory information. To be useful in an environment in which only such imprecise knowledge is available, a knowledge-based system has to capture and exploit not only the highly specialized expert knowledge, but the uncertainties that go with the represented pieces of information as well. This observation has led to the introduction of models for handling uncertain information in knowledge-based systems. Research into the representation and manipulation of uncertainty has grown into a major research area called *inexact reasoning* or *plausible reasoning.*

Probability theory is one of the oldest mathematical theories concerning uncertainty, so it is no wonder that in the early 1970s this formal theory was chosen as the first point of departure for the development of models for handling uncertain information in rule-based systems. It was soon discovered that this theory could not be applied in such a context in a straightforward manner; in Section 5.2 we shall discuss some of the problems encountered in a straightforward application of probability theory. Research then centred for a short period of time around the development of modifications of probability theory that should overcome the problems encountered and that could be applied efficiently in a rule-based environment. Several models were proposed, but neither of these presented a mathematically well-founded solution to these problems. This observation explains why we use the phrase *quasi-probabilistic models* to denote all models developed in the 1970s for rule-based systems. In this chapter, two quasi-probabilistic models will be discussed in some detail:

- the *subjective Bayesian method*, which was developed for application in the knowledge-based system PROSPECTOR;
- the *certainty factor model* which was designed by Edward Shortliffe and Bruce Buchanan for the purpose of dealing with uncertain information in MYCIN.

The treatment of these models will not only comprise a discussion of their basic notions but will also include an outline of their application in a rule-based system. In preparation for this, Section 5.1 shows which components should be present in a model for handling uncertainty in such a knowledge-based system.

The incorrectness of the quasi-probabilistic models from a mathematical point of view and an analysis of the problems the researchers were confronted with, led to a world-wide discussion concerning the appropriateness of probability theory for handling uncertain information in a knowledge-based context. This discussion has on the one hand yielded other points of departure, that is, other (more or less) mathematical foundations for models for handling uncertainty, and on the other hand new, less naive applications of probability theory. In Section 5.5 we shall present an introduction to the *Dempster-Shafer theory*, a theory which has largely been inspired by probability theory and may be considered to be an extension of it. We conclude this chapter with a discussion of two so-called *network models* which have resulted from a more recent probabilistic trend in plausible reasoning in which graphical representations of problem domains are employed.

5.1 Production rules, inference and uncertainty

In Chapter 3 we have seen that in a rule-based system the specialized domain knowledge an expert has, is modelled in production rules having the following form:

if e **then** h **fi**

The left-hand side e of such a rule is a combination of atomic conditions which are interrelated by means of the operators **and** and **or**. In the sequel such a combination of conditions will be called a (*piece of*) *evidence*. The right-hand side h of a production rule in general is a conjunction of conclusions. In this chapter we assume production rules to have just one conclusion. Notice that this restriction is not an essential one from a logical point of view. Henceforth, an atomic conclusion will be called a *hypothesis*. Furthermore, we will abstract from actions and predicates, and from variables and values, or objects, attributes, and values: conditions and conclusions will be taken to be indivisible primitives. A production rule now has the following meaning: if evidence e has been observed, then the hypothesis h is confirmed as being true.

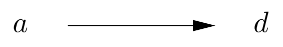
In this Section we depart from top-down inference as the method for applying production rules, and from backward chaining as described in chapter 3, more in specific. The application of production rules as it takes place in top-down inference, may be represented graphically in a so-called *inference network*. We introduce the notion of an inference network by means of an example.

EXAMPLE 5.1 ____

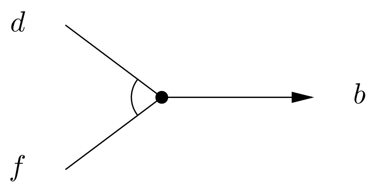
Consider the following production rules:

R_1: **if** a **and** $(b$ **or** $c)$ **then** h **fi**
R_2: **if** d **and** f **then** b **fi**
R_3: **if** f **or** g **then** h **fi**
R_4: **if** a **then** d **fi**

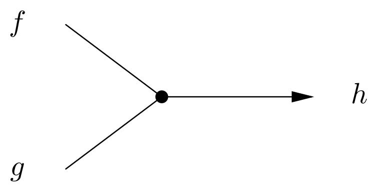
In the following, the goal for consulting a specific rule base will be called the *goal hypothesis*. We suppose that h is the goal hypothesis for consulting the set of production rules shown above. The first production rules that are selected for evaluation, are the rules R_1 and R_3. Of these, rule R_1 is evaluated first. The piece of evidence a mentioned in the left-hand side of the rule now becomes the current goal hypothesis. Since none of the production rules concludes on a, the user is requested to supply further information on a. We assume that the user confirms a being true. Subsequently, b becomes the new goal hypothesis. Since rule R_2 concludes on the hypothesis b, this rule is now selected for evaluation. The first piece of evidence mentioned in rule R_2 is d; the truth of d will be derived from rule R_4. The success of rule R_4 is depicted as follows:



In the evaluation of rule R_2 it remains to be examined whether or not the piece of evidence f has been observed. We assume that upon a request for further information, the user confirms the truth of f. So, rule R_2 succeeds; the success of rule R_2 is shown in the following figure:



Success of rule R_3 is depicted as follows:



The three figures shown above are the basic building blocks for constructing an inference network from a given set of production rules and a given goal hypothesis. The inference network resulting from a consultation of the four production rules of this example with h as the goal hypothesis is shown in Figure 5.1.

Up to now a production rule **if** e **then** h **fi** has been interpreted as stating: if evidence e has been observed, then the hypothesis h is confirmed as being true. In practice, however, a hypothesis seldom is confirmed to absolute certainty by the observation of a certain piece of evidence. Therefore, the notion of a production rule is extended by allowing for a *measure of uncertainty*: with the hypothesis h of the production rule **if** e **then** h **fi** a measure of uncertainty is associated indicating the degree to which h is confirmed by the observation of e.

EXAMPLE 5.2

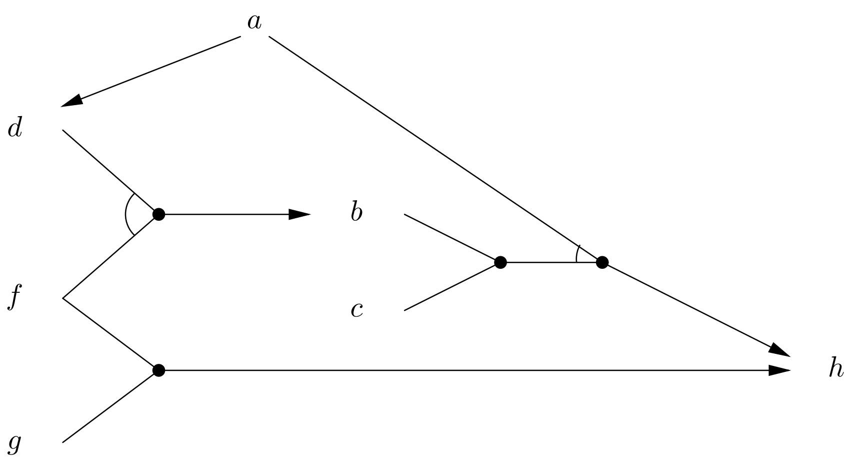
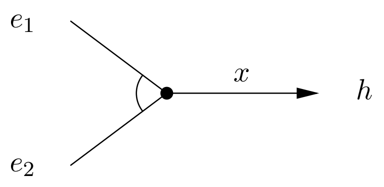


Figure 5.1: An inference network.

The measure of uncertainty x being associated with the hypothesis h in the rule **if** e_1 **and** e_2 **then** h **fi** is denoted as follows:

if e_1 **and** e_2 **then** h_x **fi**

In an inference network an associated measure of uncertainty is shown next to the arrow in the graphical representation of the rule. So, success of the production rule shown above is represented in an inference network as follows:



A model for handling uncertain information therefore provides an expert with a means for representing the uncertainties that go with the pieces of information he has specified; so, the model provides a means for *knowledge representation*.

The purpose of employing a model for dealing with uncertain information is to associate a measure of uncertainty with each conclusion the system arrives at. Such a measure of uncertainty is dependent upon the measures of uncertainty associated with the conclusions of the production rules used in deriving the final conclusion, and the measures of uncertainty the user has specified with the information he has supplied to the system. For this purpose, a model for handling uncertainty provides a means for reasoning with uncertainty, that is, it provides an *inference method*. Such an inference method consists of several components:

- Because of the way production rules of the form **if** e **then** h_y **fi** are applied during a top-down inference process, the truth of the evidence e (that is, whether or not e has actually been observed) can not always be established with absolute certainty: e may itself have been confirmed to some degree by the application of other production rules. In this case, e acts as an intermediate hypothesis that in turn is used as evidence for the confirmation of another hypothesis. The inference network shown below depicts the situation where the hypothesis e has been confirmed to the degree x on account of some prior evidence e':

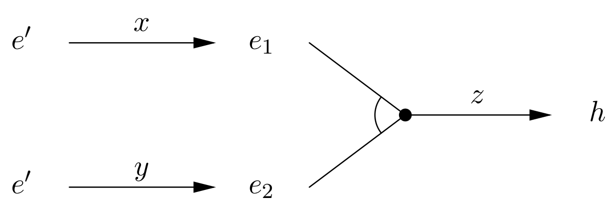
$$\begin{tikzcd} e' \arrow[r, "x"] & e \arrow[r, "y"] & h \end{tikzcd}$$

Note that the left half of this figure shows a *compressed* inference network whereas the right half represents a single production rule. We recall that the measure of uncertainty y associated with the hypothesis h in the rule **if** e **then** h_y **fi** indicates the degree to which h is confirmed by the *actual observation*, that is, the absolute truth of e. It will be evident that in the situation shown above, we cannot simply associate the measure of uncertainty y with the hypothesis h. The actual measure of uncertainty to be associated with h depends upon y as well as on x, the measure of uncertainty associated with the evidence e used in confirming h: the uncertainty of e has to be *propagated* to h. A model for handling uncertainty provides a function for computing the actual measure of uncertainty to be associated with h on account of all prior evidence. In the sequel, such a function will be called the *combination function for (propagating) uncertain evidence*; the function will be denoted by f_{prop}. The inference network shown above can now be compressed to:

$$\begin{tikzcd} e' \arrow[r, "f_{prop}(x,y)"] & h \end{tikzcd}$$

where e' denotes *all* prior evidence (now including e).

- The evidence e in a production rule **if** e **then** h_z **fi** in general is a combination of atomic conditions which are interrelated by means of the operators **and** and **or**. For instance, the production rule may have the form **if** e_1 **and** e_2 **then** h_z **fi** as depicted in the inference network below. Each of the constituent pieces of evidence of e may have been derived with an associated measure of uncertainty. The inference network, for example, shows that e_1 and e_2 are confirmed to the degrees x and y, respectively, on account of the prior evidence e':



To be able to apply the combination function for propagating uncertain evidence, a measure of uncertainty for e has to be computed from the measures of uncertainty that have been associated separately with the constituent pieces of evidence of e. For this purpose, a model for handling uncertainty provides two functions which will be called the *combination functions for composite hypotheses*; they will be denoted by f_{and} and f_{or}. The inference network shown above is now compressed to:

$$\begin{tikzcd} e' \arrow[r, "f_{and}(x,y)"] & e_1 \textbf{ and } e_2 \end{tikzcd}$$

- The occurrence of different production rules **if** e_i **then** h **fi** (that is, rules with different left-hand sides e_i) concluding on the same hypothesis h in the rule base, indicates that the hypothesis h may be confirmed and/or disconfirmed along different lines of reasoning. The following inference network, for example, shows the two production rules **if** e_1 **then** h_{x_2} **fi** and **if** e_2 **then** h_{y_2} **fi** concluding on the hypothesis h, the first of which uses the prior evidence e_1' in (dis)confirming h and the second of which uses the prior evidence e_2':

$$\begin{tikzcd} e_1' \arrow[r, "x_1"] & e_1 \arrow[dr, "x_2"] & \\ & & h \\ e_2' \arrow[r, "y_1"] & e_2 \arrow[ur, "y_2"'] & \end{tikzcd}$$

The combination function for propagating uncertain evidence is applied to compute two *partial* measures of uncertainty x and y for h such that:

$$\begin{tikzcd} e_1' \arrow[dr, "x"] & \\ & h \\ e_2' \arrow[ur, "y"'] & \end{tikzcd}$$

The total or *net* measure of uncertainty to be associated with h depends upon the partial measures of uncertainty that have been computed for h from the two different lines of reasoning. A model for handling uncertain information therefore provides a function for computing the net measure of uncertainty for h in the inference network shown above. Such a function will be called the *combination function for co-concluding production rules*; it will be denoted by f_{co}:

$$\begin{tikzcd} e' = e_1' \textbf{ co } e_2' \arrow[r, "f_{co}(x,y)"] & h \end{tikzcd}$$

To summarize, we have introduced four combination functions:

- the function for propagating uncertain evidence: f_{prop};
- the functions for composite hypotheses: f_{and} and f_{or};
- the function for co-concluding production rules: f_{co}.

It will be evident that a model for handling uncertainty in a rule-based system has to provide fill-ins for these combination functions.

5.2 Probability theory

Probability theory is one of the earliest methods for associating with a statement a measure of uncertainty concerning its truth. In this section several notions from probability theory are introduced briefly, before we discuss the problems one encounters in applying this theory in a rule-based system in a straightforward manner.

5.2.1 The probability function

The notions that play a central role in probability theory have been developed for the description of experiments. In empirical research a more or less standard procedure is to repeatedly perform a certain experiment under essentially the same conditions. Each performance yields an *outcome* which cannot be predicted with certainty in advance. For many types of experiments, however, one is able to describe the set of all *possible* outcomes. The nonempty set of all possible outcomes of such an experiment is called its *sample space*; it is generally denoted by Ω. In the sequel, we shall only be concerned with experiments having a countable sample space.

EXAMPLE 5.3

> Consider the experiment of throwing a die. The outcome of the experiment is the number of spots up the die. The sample space of this experiment therefore consists of six elements: $\Omega = \{1, 2, 3, 4, 5, 6\}$

A subset e of the sample space Ω of a certain experiment is called an *event*. If upon performance of the experiment the outcome is in e, then it is said that the event e has occurred. In case the event e has not occurred, we use the notation $\bar{e}$, called the *complement* of e. Note that we have $\bar{e} = \Omega \setminus e$. The event that occurs if and only if both events e_1 and e_2 occur, is called the *intersection* of e_1 and e_2, and will be denoted by $e_1 \cap e_2$. The intersection of n events e_i will be denoted by

$$\bigcap_{i=1}^{n} e_i$$

The event occurring if at least one of e_1 and e_2 occurs is called the *union* of e_1 and e_2, and will be denoted by $e_1 \cup e_2$. The union of n events e_i will be denoted by

$$\bigcup_{i=1}^{n} e_i$$

EXAMPLE 5.4

> Consider the experiment of throwing a die and its associated sample space Ω once more. The subset $e_1 = \{2, 4, 6\}$ of Ω represents the event that an even number of spots has come up the die. The subset $e_2 = \bar{e}_1 = \Omega \setminus e_1 = \{1, 3, 5\}$ represents the event that an

odd number of spots has come up. The events e_1 and e_2 cannot occur simultaneously: if event e_1 occurs, that is, if an even number of spots has come up, then it is not possible that in the same throw an odd number of spots has come up. So, the event $e_1 \cap e_2$ cannot occur. Note that the event $e_1 \cup e_2$ occurs in every performance of the experiment. The subset $e_3 = \{3, 6\}$ represents the event that the number of spots that has come up is a multiple of three. Note that the events e_1 and e_3 have occurred simultaneously in case six spots are shown up the die: in that case the event $e_1 \cap e_3$ has occurred.

Definition 5.1 *The events* $e_1, \ldots, e_n \subseteq \Omega$, $n \geq 1$, *are called* mutually exclusive *or* disjoint events *if* $e_i \cap e_j = \varnothing$, $i \neq j$, $1 \leq i, j \leq n$.

We assume that an experiment yields an outcome independent of the outcomes of prior performances of the experiment. Now suppose that a particular experiment has been performed N times. If throughout these N performances an event e has occurred n times, the ratio $\frac{n}{N}$ is called the *relative frequency* of the occurrence of event e in N performances of the experiment. As N increases, the relative frequency of the occurrence of the event e tends to stabilize about a certain value; this value is called the *probability* that the outcome of the experiment is in e, or the probability of event e, for short.

In general, the notions of a probability and a probability function are defined axiomatically.

Definition 5.2 *Let Ω be the sample space of an experiment. If a number $P(e)$ is associated with each subset $e \subseteq \Omega$, such that*

(1) $P(e) \geq 0$,

(2) $P(\Omega) = 1$, *and*

(3) $P(\bigcup_{i=1}^{n} e_i) = \sum_{i=1}^{n} P(e_i)$, *if e_i, $i = 1, \ldots, n$, $n \geq 1$, are mutually exclusive events,*

then P is called a probability function *on the sample space Ω. For each subset $e \subseteq \Omega$, the number $P(e)$ is called the probability that event e will occur.*

Note that a probability function P on a sample space Ω is a function $P : 2^{\Omega} \rightarrow [0, 1]$.

EXAMPLE 5.5

Consider the experiment of throwing a die once more, and its associated sample space $\Omega = \{1, 2, 3, 4, 5, 6\}$. The function P such that $P(\{1\}) = P(\{2\}) = \cdots = P(\{6\}) = \frac{1}{6}$ is a probability function on Ω. Since the sets $\{2\}$, $\{4\}$, and $\{6\}$ are disjoint, we have according to the third axiom of the preceding definition that $P(\{2, 4, 6\}) = \frac{1}{2}$: the probability of an even number of spots coming up the die, equals $\frac{1}{2}$.

THEOREM 2 *Let Ω be the sample space of an experiment and P a probability function on Ω. Then, for each event $e \subseteq \Omega$, we have*

$$P(\bar{e}) = 1 - P(e)$$

Proof: We have $\Omega = e \cup \bar{e}$. Furthermore, $e \cap \bar{e} = \varnothing$ holds since e and $\bar{e}$ are mutually exclusive events. From the axioms 2 and 3 of the preceding definition we have that $P(\Omega) = P(e \cup \bar{e}) = P(e) + P(\bar{e}) = 1$. $\Diamond$

5.2.2 Conditional probabilities and Bayes' Theorem

We consider the case in which probability theory is applied in a medical diagnostic system. One would like to know for example the probability of the event that a specific patient has a certain disease. For many diseases, the prior probability of the disease occurring in a certain population is known. In the case of a specific patient, however, information concerning the patient's symptoms, medical history, etc. is available that might be useful in determining the probability of the presence of the disease in this specific patient.

So, in some cases we are interested only in those outcomes which are in a given nonempty subset e of the entire sample space which represents the pieces of evidence concerning the final outcome that are known in advance. Let h be the event we are interested in, that is, the hypothesis. Given that the evidence e has been observed, we now are interested in the degree to which this information influences $P(h)$, the prior probability of the hypothesis h. The probability of h given e is defined in the following definition.

Definition 5.3 *Let Ω be the sample space of a certain experiment and let P be a probability function on Ω. For each $h, e \subseteq \Omega$ with $P(e) > 0$, the* conditional probability *of h given e, denoted by $P(h \mid e)$, is defined as*

$$P(h \mid e) = \frac{P(h \cap e)}{P(e)}$$

A conditional probability $P(h \mid e)$ often is called a *posterior* probability.

The conditional probabilities given a fixed event $e \subseteq \Omega$ with $P(e) > 0$, again define a probability function on Ω since the three axioms of a probability function are satisfied:

- $P(h \mid e) = \dfrac{P(h \cap e)}{P(e)} \geq 0$, since $P(h \cap e) \geq 0$ and $P(e) > 0$;
- $P(\Omega \mid e) = \dfrac{P(\Omega \cap e)}{P(e)} = \dfrac{P(e)}{P(e)} = 1$;
- $P(\bigcup\limits_{i=1}^{n} h_i \mid e) = \dfrac{P((\bigcup_{i=1}^{n} h_i) \cap e)}{P(e)} = \dfrac{P(\bigcup_{i=1}^{n}(h_i \cap e))}{P(e)} = \dfrac{\sum_{i=1}^{n} P(h_i \cap e)}{P(e)} =$
  $\sum\limits_{i=1}^{n} \dfrac{P(h_i \cap e)}{P(e)} = \sum\limits_{i=1}^{n} P(h_i \mid e)$, for mutually exclusive events h_i, $i = 1, \ldots, n$, $n \geq 1$.

This probability function is called the *conditional probability function given e*.

In real-life practice, the probabilities $P(h \mid e)$ cannot always be found in the literature or obtained from statistical analysis. The conditional probabilities $P(e \mid h)$, however, often are easier to come by: in medical textbooks for example, a disease is described in terms of the signs likely to be found in a typical patient suffering from the disease. The following theorem now provides us with a method for computing the conditional probability $P(h \mid e)$ from the probabilities $P(e)$, $P(h)$, and $P(e \mid h)$; the theorem may therefore be used to reverse the 'direction' of probabilities.

THEOREM 3 (Bayes' theorem) *Let P be a probability function on a sample space Ω. For each $h, e \subseteq \Omega$ such that $P(e) > 0$ and $P(h) > 0$, we have:*

$$P(h \mid e) = \frac{P(e \mid h) \cdot P(h)}{P(e)}$$

Proof: The conditional probability of h given e is defined as

$$P(h \mid e) = \frac{P(h \cap e)}{P(e)}$$

Furthermore, we have

$$P(e \mid h) = \frac{P(e \cap h)}{P(h)}$$

So,

$$P(e \mid h) \cdot P(h) = P(h \mid e) \cdot P(e) = P(h \cap e)$$

The property stated in the theorem now follows from these observations. $\Diamond$

EXAMPLE 5.6

Consider the problem domain of medical diagnosis. Let h denote the hypothesis that a patient is suffering from liver cirrhosis; furthermore, let e denote the evidence that the patient has jaundice. In this case, the prior probability of liver cirrhosis, that is, $P($*liver-cirrhosis*$)$, is known: it is the relative frequency of the disease in a particular population. If the prior probability of the occurrence of jaundice in the same population, that is, $P($*jaundice*$)$, is likewise available and if the probability that a patient suffering from liver cirrhosis has jaundice, that is, the conditional probability $P($*jaundice* $\mid$ *liver-cirrhosis*$)$, is known, then we can compute the probability that a patient showing signs of jaundice suffers from liver cirrhosis, that is, using Bayes' theorem we can compute the conditional probability $P($*liver-cirrhosis* $\mid$ *jaundice*$)$. It will be evident that the last-mentioned probability is of importance in medical diagnosis.

To conclude, we define the notions of independence and conditional independence. Intuitively speaking, it seems natural to call an event h independent of an event e if $P(h \mid e) = P(h)$: the prior probability of event h is not influenced by the knowledge that event e has occurred. However, this intuitive definition of the notion of independency is not symmetrical in h and e; furthermore, the notion is defined this way only in case $P(e) > 0$. By using the definition of conditional probability and by considering the case for n events, we come to the following definition.

Definition 5.4 *The events* $e_1, \ldots, e_n \subseteq \Omega$ *are (*mutually*)* independent *if*

$$P(e_{i_1} \cap \cdots \cap e_{i_k}) = P(e_{i_1}) \cdots P(e_{i_k})$$

for each subset $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$, $1 \leq k \leq n$, $n \geq 1$. *The events* $e_1, \ldots, e_n$ *are* conditionally independent *given an event* $h \subseteq \Omega$ *if*

$$P(e_{i_1} \cap \cdots \cap e_{i_k} \mid h) = P(e_{i_1} \mid h) \cdots P(e_{i_k} \mid h)$$

for each subset $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$.

Note that if the events h and e are independent and if $P(e) > 0$, we have that the earlier mentioned, intuitively more appealing notion of independency

$$P(h \mid e) = \frac{P(h \cap e)}{P(e)} = \frac{P(h) \cdot P(e)}{P(e)} = P(h)$$

is satisfied.

5.2.3 Application in rule-based systems

We have mentioned before in our introduction that probability theory was chosen as the first point of departure in the pioneering work on automated reasoning under uncertainty. During the 1960s several research efforts on probabilistic reasoning were undertaken. The systems constructed in this period of time were primarily for (medical) diagnosis. Although these systems did not exhibit any intelligent reasoning behaviour, they may now be viewed as the precursors of the diagnostic systems developed in the 1970s.

Let us take a closer look at the task of diagnosis. Let $H = \{h_1, \ldots, h_n\}$ be a set of n possible hypotheses, and let $E = \{e_1, \ldots, e_m\}$ be a set of pieces of evidence which may be observed. For ease of exposition, we assume that each of the hypotheses is either true or false for a given case; equally, we assume that each of the pieces of evidence is either true (that is, it is actually observed in the given case) or false. The diagnostic task now is to find a set of hypotheses $h \subseteq H$, called the (differential) *diagnosis*, which most likely accounts for the set of observed evidence $e \subseteq E$. If we have observed a set of pieces of evidence $e \subseteq E$, then we can simply compute the conditional probabilities $P(h \mid e)$ for each subset $h \subseteq H$ and select the set $h' \subseteq H$ with the highest probability. We have mentioned before that since for real-life applications, the conditional probabilities $P(e \mid h)$ often are easier to come by than the conditional probabilities $P(h \mid e)$, generally Bayes' theorem is used for computing $P(h \mid e)$. It will be evident that the task of diagnosis in this form is computationally complex: since a diagnosis may comprise more than one hypothesis out of n possible ones, the number of diagnoses to be investigated, that is, the number of probabilities to be computed, equals 2^n. A simplifying assumption generally made in the systems for probabilistic reasoning developed in the 1960s, is that the hypotheses in H are mutually exclusive and collectively exhaustive. With this assumption, we only have to consider the n singleton hypotheses $h_i \in H$ as separate possible diagnoses. Bayes' theorem can easily be reformulated to deal with this case.

THEOREM 4 (Bayes' theorem) *Let P be a probability function on a sample space Ω. Let $h_i \subseteq \Omega$, $i = 1, \ldots, n$, $n \geq 1$, be mutually exclusive hypotheses with $P(h_i) > 0$, such that $\bigcup_{i=1}^{n} h_i = \Omega$ (that is, they are collectively exhaustive). Furthermore, let $e \subseteq \Omega$ such that $P(e) > 0$. Then, the following property holds:*

$$P(h_i \mid e) = \frac{P(e \mid h_i) \cdot P(h_i)}{\sum_{j=1}^{n} P(e \mid h_j) \cdot P(h_j)}$$

Proof: Since $h_1, \ldots, h_n$ are mutually exclusive and collectively exhaustive, we have that $P(e)$ can be written as

$$P(e) = P((\bigcup_{i=1}^{n} h_i) \cap e) = P(\bigcup_{i=1}^{n} (h_i \cap e)) = \sum_{i=1}^{n} P(h_i \cap e) = \sum_{i=1}^{n} P(e \mid h_i) \cdot P(h_i)$$

Substitution of this result in the before-mentioned form of Bayes' theorem yields the property stated in the theorem. $\Diamond$

For a successful application of Bayes' theorem in the form mentioned in the previous theorem, several conditional and prior probabilities are required. For example, conditional probabilities $P(e \mid h_i)$ for every combination of pieces of evidence $e \subseteq E$, have to be available; note that in general, these conditional probabilities $P(e \mid h_i)$ cannot be computed from their 'component' conditional probabilities $P(e_j \mid h_i)$, $e_j \in e$. It will be evident that exponentially many probabilities have to be known beforehand. Since it is hardly likely that for practical applications all these probabilities can be obtained from for example statistical analysis, a second simplifying assumption was generally made in the systems developed in the 1960s: it was assumed that the pieces of evidence $e_j \in E$ are conditionally independent given any hypothesis $h_i \in H$. Under this assumption Bayes' theorem reduces to the following form.

THEOREM 5 (Bayes' theorem) *Let P be a probability function on a sample space Ω. Let $h_i \subseteq \Omega$, $i = 1, \ldots, n$, $n \geq 1$, be mutually exclusive and collectively exhaustive hypotheses as in the previous theorem. Furthermore, let $e_{j_1}, \ldots, e_{j_k} \subseteq \Omega$, $1 \leq k \leq m$, $m \geq 1$, be pieces of evidence such that they are conditionally independent given any hypothesis h_i. Then, the following property holds:*

$$P(h_i \mid e_{j_1} \cap \cdots \cap e_{j_k}) = \frac{P(e_{j_1} \mid h_i) \cdots P(e_{j_k} \mid h_i) \cdot P(h_i)}{\sum_{i=1}^{n} P(e_{j_1} \mid h_i) \cdots P(e_{j_k} \mid h_i) \cdot P(h_i)}$$

Proof: The theorem follows immediately from the preceding theorem and the definition of conditional independence. $\Diamond$

It will be evident that with the two assumptions mentioned above only $m \cdot n$ conditional probabilities and $n - 1$ prior probabilities suffice for a successful use of Bayes' theorem.

The pioneering systems for probabilistic reasoning constructed in the 1960s which basically employed the last-mentioned form of Bayes' theorem, were rather small-scaled: they were devised for clear-cut problem domains with only a small number of hypotheses and restricted evidence. For these small systems, all probabilities necessary for applying Bayes' theorem were acquired from a statistical analysis of the data of several hundred sample cases. Now recall that in deriving the last-mentioned form of Bayes' theorem several assumptions were made:

- the hypotheses $h_1, \ldots, h_n$, $n \geq 1$, are mutually exclusive;
- the hypotheses $h_1, \ldots, h_n$ furthermore are collectively exhaustive, that is, $\bigcup_{i=1}^{n} h_i = \Omega$;
- the pieces of evidence $e_1, \ldots, e_m$, $m \geq 1$, are conditionally independent given any hypothesis h_i, $1 \leq i \leq n$.

These conditions, which have to be satisfied for a correct use of Bayes' theorem, generally are not met in practice. But, in spite of these (over-)simplifying assumptions underlying the systems from the 1960s, they performed considerably well. Nevertheless, interest in this approach to reasoning with uncertainty faded in the early 1970s. One of the reasons for this decline in interest is that the method informally sketched in the foregoing is feasible only for highly restricted problem domains: for larger domains or domains in which the above-mentioned simplifying assumptions are seriously violated, the method inevitably will become demanding, either computationally or from the point of view of obtaining the necessary probabilities: often a large number of conditional and prior probabilities is needed, thus requiring enormous amounts of experimental data.

At this stage, the first diagnostic rule-based systems began to emerge from the early artificial intelligence research efforts. As a consequence of their ability to concentrate only on those hypotheses which are suggested by the evidence, these systems in principle were capable of dealing with larger and complexer problem domains than the early probabilistic systems were. At least, they were so from a computational point of view: the problem that a large number of probabilities was required still remained. In many practical applications, the experimental data necessary for computing all probabilities required simply were not available. In devising a probabilistic reasoning component to be incorporated in a rule-based system, the artificial intelligence researchers therefore had to depart from *subjective probabilities* which had been assessed by human experts in the field. Human experts, however, often are uncertain and uncomfortable about the probabilities they are providing. The difficulty of assessing probabilities is well-known as a result of research on human decision making and judgement under uncertainty. We do not discuss this issue any further; we merely depart from the observation that domain experts generally are unable to fully and correctly specify a probability function on the problem domain. In a rule-based context, an expert now typically is asked to associate probabilities only with the production rules he has provided.

Recall that the production rule formalism is defined in terms of expressions more or less resembling logical formulas, whereas the notion of a probability function has been related to sets. Therefore, we have to have a mapping that transforms logical propositions into sets and that preserves probability, for then we have that the probability of an event is equivalent to the probability of the truth of the proposition asserting the occurrence of the event. A more or less standard translation of sets into logical formulas is the following: if Ω is a sample space, then we define for each event $e \subseteq \Omega$ a predicate e' such that $e'(x) = \mathit{true}$ if and only if $x \in e$. The intersection of two events then corresponds with the conjunction of two corresponding propositions; the union of two events translates into the disjunction of the corresponding propositions.

With each production rule **if** e **then** h **fi** an expert now associates a conditional probability $P(h \mid e)$ indicating the influence of the observation of evidence e on the prior probability $P(h)$ of the hypothesis h:

$$\begin{tikzcd} e \arrow[r, "P(h \mid e)"] & h \end{tikzcd}$$

The last-mentioned form of Bayes' theorem now provides us with a method for computing the probability of a certain hypothesis when several pieces of evidence have been observed. Bayes' theorem therefore can be taken as the combination function for co-concluding production rules when probability theory is viewed as a method for handling uncertainty as discussed in Section 5.1. Consider the following inference network:

$$\begin{tikzcd} e_1 \arrow[dr, "P(h_i \mid e_1)"] & \\ \vdots & h_i \\ e_m \arrow[ur, "P(h_i \mid e_m)"'] & \end{tikzcd}$$

Using Bayes' theorem we can compute the combined influence of the pieces of evidence $e_1, \ldots, e_m$ on the prior probability of the hypothesis h_i such that:

$$\begin{tikzcd} \bigcap_{j=1}^{m} e_j \arrow[r, "P(h_i \mid \bigcap_{j=1}^{m} e_j)"] & h_i \end{tikzcd}$$

(Note that some prior probabilities have to be known to the system as well).

In a rule-based system, the production rules are used for pruning the search space of possible diagnoses; in this pruning process, heuristic as well as probabilistic criteria are employed. It is, therefore, necessary to compute the probabilities of all intermediate results derived using the production rules. However, these probabilities generally cannot be computed from the probabilities associated with the rules only: probability theory does not provide an explicit combination function for propagating uncertain evidence nor does it provide combination functions for composite hypotheses in terms of the available probabilities. We have suggested before that the quasi-probabilistic models do offer explicit combination functions. From the previous observation it will be evident that these functions cannot accord with the axioms of probability theory. Therefore, they can only be viewed as approximation functions rendering the models to some extent insensitive to the lack of a fully specified probability function and erroneous probability assessments.

5.3 The subjective Bayesian method

In the preceding section we have highlighted some of the problems one encounters when applying probability theory in a rule-based system. R.O. Duda, P.E. Hart, and N.J. Nilsson have recognized these problems and have developed a new method for handling uncertainty in PROSPECTOR, a knowledge-based system for assisting non-expert field geologists in exploring sites. Part of the knowledge incorporated in PROSPECTOR is represented in production rules. The model of Duda, Hart, and Nilsson is based on probability theory but provides solutions to the problems mentioned in the previous section.

5.3.1 The likelihood ratios

As has been mentioned before, the subjective Bayesian method is a modification of probability theory. However, the model uses the notion of 'odds' instead of the equivalent notion of probability.

Definition 5.5 *Let P be a probability function on a sample space Ω. Furthermore, let $h \subseteq \Omega$ such that $P(h) < 1$. The* prior odds *of the event h, denoted by $O(h)$, is defined as follows:*

$$O(h) = \frac{P(h)}{1 - P(h)}$$

Note that conversely

$$P(h) = \frac{O(h)}{1 + O(h)}$$

In probability theory the notion of conditional or posterior probability is used. The subjective Bayesian method uses the equivalent notion of posterior odds.

Definition 5.6 *Let P be a probability function on a sample space Ω. Let $h, e \subseteq \Omega$ such that $P(e) > 0$ and $P(h \mid e) < 1$. The* posterior odds *of a hypothesis h, given evidence e, denoted by $O(h \mid e)$, is defined as follows:*

$$O(h \mid e) = \frac{P(h \mid e)}{1 - P(h \mid e)}$$

We introduce another two notions: the positive and the negative likelihood ratios.

Definition 5.7 *Let P be a probability function on a sample space Ω. Furthermore, let $h, e \subseteq \Omega$ such that $0 < P(h) < 1$ and $P(e \mid \bar{h}) > 0$. The (*positive*)* likelihood ratio *λ, given h and e, is defined by*

$$\lambda = \frac{P(e \mid h)}{P(e \mid \bar{h})}$$

The likelihood ratio λ often is called the *level of sufficiency*; it represents the degree to which the observation of evidence e influences the prior probability of hypothesis h. A likelihood ratio $\lambda > 1$ indicates that the observation of e tends to confirm the hypothesis h; a likelihood ratio $\lambda < 1$ indicates that the hypothesis $\bar{h}$ is confirmed to some degree by the observation of e, or in other words that the observation of e tends to disconfirm h. If $\lambda = 1$, then the observation of e does not influence the prior confidence in h.

Definition 5.8 *Let P be a probability function on a sample space Ω. Let $h, e \subseteq \Omega$ be such that $0 < P(h) < 1$ and $P(e \mid \bar{h}) < 1$. The (*negative*)* likelihood ratio *$\bar{\lambda}$, given h and e, is defined by*

$$\bar{\lambda} = \frac{1 - P(e \mid h)}{1 - P(e \mid \bar{h})}$$

The negative likelihood ratio $\bar{\lambda}$ often is called the *level of necessity.* A comparison of the likelihood ratios λ and $\bar{\lambda}$ shows that from $\lambda > 1$ it follows that $\bar{\lambda} < 1$, and vice versa; furthermore we have $\lambda = 1$ if and only if $\bar{\lambda} = 1$.

When applying the subjective Bayesian method in a production system, a positive likelihood ratio λ and a negative likelihood ratio $\bar{\lambda}$ have to be associated with each production rule **if** e **then** h **fi**:

$$e \xrightarrow{\lambda, \bar{\lambda}} h$$

Furthermore, the prior probabilities $P(h)$ as well as $P(e)$ have to be known to the system. Note that this information is not sufficient for uniquely defining a probability function on the sample space: the expert has provided probabilities for only a few events occurring in the specified production rules.

In the following section, in some cases the conditional probabilities $P(h \mid e)$ and $P(h \mid \bar{e})$ will be preferred to λ and $\bar{\lambda}$: we then assume that with each production rule these conditional probabilities are associated. We note that the probabilities $P(h \mid e)$ and $P(h \mid \bar{e})$ can be computed uniquely from λ, $\bar{\lambda}$, $P(h)$ and $P(e)$. The reader may for example verify that the following property holds:

$$P(e \mid h) = \lambda \cdot \frac{1 - \bar{\lambda}}{\lambda - \bar{\lambda}}$$

Bayes' theorem can subsequently be applied to compute the probability $P(h \mid e)$.

5.3.2 The combination functions

Recall that a model for dealing with uncertainty provides means for representing and reasoning with uncertainty. The purpose of applying such a model is to compute a measure of uncertainty for each goal hypothesis. If a probability function on the domain were known, then the probabilities of these goal hypotheses could simply be calculated from the probability function. However, as we have argued before, such a probability function is virtually never available in practical applications. The required probabilities therefore are approximated from the ones that actually are known to the system.

In a rule-based system using top-down inference, several intermediate hypotheses are confirmed or disconfirmed to some degree. We have seen before that these uncertain hypotheses may in turn be used as pieces of evidence in other production rules. In Section 5.1 a combination function for propagating such uncertain evidence has been introduced: the function f_{prop}. Recall that probability theory does not provide an explicit filling-in for this function f_{prop} in terms of the probabilities that are known to the system. The subjective Bayesian method, however, does provide such a combination function.

Suppose that the intermediate hypothesis e is used as evidence in confirming hypothesis h by applying the production rule **if** e **then** h **fi**. We suppose that the intermediate hypothesis e has been confirmed by the observation of some prior evidence e', and that for e the posterior probability $P(e \mid e')$ has been computed.

$$\begin{tikzcd} e' \arrow[r, "P(e \mid e')"] & e \arrow[r, "{P(h \mid e), P(h \mid \bar{e})}"] & h \end{tikzcd}$$

After application of the rule, we are interested in the probability $P(h \mid e')$ such that

$$\begin{tikzcd} e' \arrow[r, "P(h \mid e')"] & h \end{tikzcd}$$

Note that in general the probability $P(h \mid e')$ will not have been assessed by the expert and cannot be computed from the probability function P since P has not been fully specified. Therefore, it has to be approximated. In general, we have

$$\begin{aligned}
P(h \mid e') &= P(h \cap e \mid e') + P(h \cap \bar{e} \mid e') \\
&= \frac{P(h \cap e \cap e')}{P(e')} \cdot \frac{P(e \cap e')}{P(e \cap e')} + \frac{P(h \cap \bar{e} \cap e')}{P(e')} \cdot \frac{P(\bar{e} \cap e')}{P(\bar{e} \cap e')} \\
&= \frac{P(h \cap e \cap e')}{P(e \cap e')} \cdot \frac{P(e \cap e')}{P(e')} + \frac{P(h \cap \bar{e} \cap e')}{P(\bar{e} \cap e')} \cdot \frac{P(\bar{e} \cap e')}{P(e')} \\
&= P(h \mid e \cap e')P(e \mid e') + P(h \mid \bar{e} \cap e')P(\bar{e} \mid e')
\end{aligned}$$

We assume that if we know e to be absolutely true (or false), then the observations e' relevant to e do not provide any *further* information on the hypothesis h. This assumption can be taken into account into the formula given above as follows:

$$\begin{aligned}
P(h \mid e') &= P(h \mid e)P(e \mid e') + P(h \mid \bar{e})P(\bar{e} \mid e') \\
&= (P(h \mid e) - P(h \mid \bar{e})) \cdot P(e \mid e') + P(h \mid \bar{e})
\end{aligned}$$

We have that $P(h \mid e')$ is a linear interpolation function in $P(e \mid e')$ (since the function has the form $f(x) = ax + b$). In Figure 5.2 such an interpolation function for the situation of the

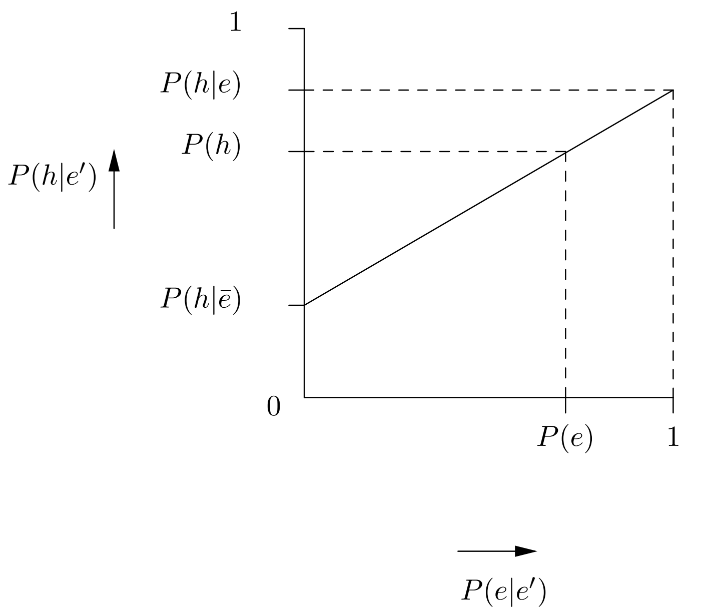


Figure 5.2: $P(h \mid e')$ as a linear interpolation function in $P(e \mid e')$.

production rule **if** e **then** h **fi** shown above, is depicted. This interpolation function has two extreme values: for $P(e \mid e') = 0$ we have the extreme value $P(h \mid e') = P(h \mid \bar{e})$, and for $P(e \mid e') = 1$ we have the extreme value $P(h \mid e') = P(h \mid e)$. For any $P(e \mid e')$ between 0 and 1 the corresponding value for $P(h \mid e')$ can be read from the figure. For instance, if evidence e' has been observed confirming e, that is, if $P(e \mid e') > P(e)$, we find that the probability of h increases from applying the production rule **if** e **then** h **fi**: $P(h \mid e') > P(h)$. Notice that this effect is exactly what is meant by the rule. In the special case where $P(e \mid e') = P(e)$, we have

$$P(h \mid e') = P(h \mid e)P(e) + P(h \mid \bar{e})P(\bar{e}) = P(h)$$

In principle, this interpolation function offers an explicit computation rule for propagating uncertain evidence. Duda, Hart, and Nilsson however have observed that when an expert is asked to assess for each rule **if** e **then** h **fi** the four probabilities $P(h), P(e), P(h \mid e)$, and $P(h \mid \bar{e})$, the specified values are likely to be *inconsistent*, in the sense that there is not an underlying actual probability function. More in specific, the relation between $P(h)$ and $P(e)$ as shown in Figure 5.2 will be violated. We show to which problems such an inconsistency may lead. Consider Figure 5.3. The assessed probabilities $P(h), P(e), P(h \mid e)$ and $P(h \mid \bar{e})$ shown in the figure are inconsistent; the consistent value for $P(e \mid e')$ corresponding with $P(h)$ is indicated as $P_c(e)$. Now suppose that evidence e' has been observed confirming e to a degree $P(e \mid e')$ such that $P(e) < P(e \mid e') < P_c(e)$. From Figure 5.3 we have that $P(h \mid e') < P(h)$. The production rule **if** e **then** h **fi** however was meant to express that confirmation of e leads to confirmation of h: due to the inconsistency the reverse has been achieved! A natural solution to this problem would be to reassess $P(e)$ by choosing $P(e) = P_c(e)$ (or, in case the assessment of $P(h)$ is less certain than the assessment of $P(e)$, to reassess $P(h)$ by choosing a consistent value for $P(h)$). The hypotheses h and e however may occur in several places in a given set of production rules and each reassessment affects all these occurrences. Reassessing prior probabilities therefore is not a feasible solution to the problem we have discussed.

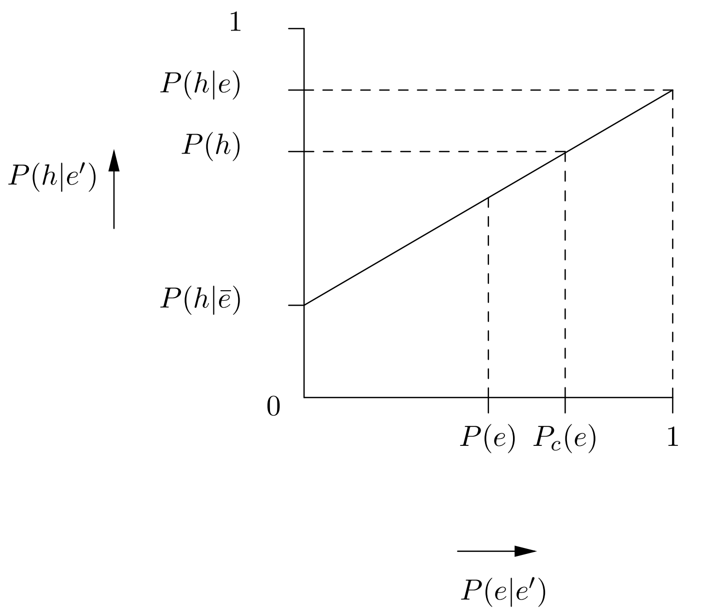


Figure 5.3: Inconsistent prior probabilities $P(h)$ and $P(e)$.

Duda, Hart, and Nilsson have developed several methods for employing inconsistently specified probabilities, one of which has been implemented as the function for propagating uncertain evidence in PROSPECTOR. The basic idea of the method that has been chosen for implementation is shown in Figure 5.4. The original interpolation function is splitted in two separate interpolation functions on the intervals $[0, P(e)]$ and $(P(e), 1]$, respectively, so as to enforce the property $P(h \mid e') = P(h)$ if $P(e \mid e') = P(e)$. Note that the closer the function value for $P(e)$ is to the value for $P(e)$ from the original interpolation function, the better the initial assessments of $P(e)$ and $P(h)$ are. The resulting interpolation function is defined as follows:

$$P(h \mid e') = \begin{cases} P(h \mid \bar{e}) + \frac{P(h)-P(h|\bar{e})}{P(e)} \cdot P(e \mid e') & \text{if } 0 \leq P(e \mid e') \leq P(e) \\ P(h) + \frac{P(h|e)-P(h)}{1-P(e)} \cdot (P(e \mid e') - P(e)) & \text{if } P(e) < P(e \mid e') \leq 1 \end{cases}$$

Recall that the conditional probabilities $P(h \mid e)$ and $P(h \mid \bar{e})$ used in this function are obtained from the likelihood ratios λ en $\bar{\lambda}$ provided by the expert.

We have mentioned before that with each production rule **if** e **then** h **fi** the two likelihood ratios λ and $\bar{\lambda}$ have been associated: λ stands for the influence of the observation of evidence e on the prior probability of the hypothesis h, and $\bar{\lambda}$ indicates the degree to which observation of $\bar{e}$ changes the probability of h. The ratios λ and $\bar{\lambda}$ can be viewed as the bounds of an interval in which lies a value indicating the degree to which evidence e, which has been (dis)confirmed to some degree by some prior evidence e', really influences the prior probability of h. This value is called the effective likelihood ratio, and will be denoted by λ'. The ratio λ' is computed from the value $P(h \mid e')$ according to the following definition.

Definition 5.9 *Let P be a probability function on a sample space Ω, and let O be the corresponding odds as defined in the foregoing. Furthermore, let $h, e' \subseteq \Omega$. The* effective likelihood ratio *λ', given h and e', is defined as follows:*

$$\lambda' = \frac{O(h \mid e')}{O(h)}$$

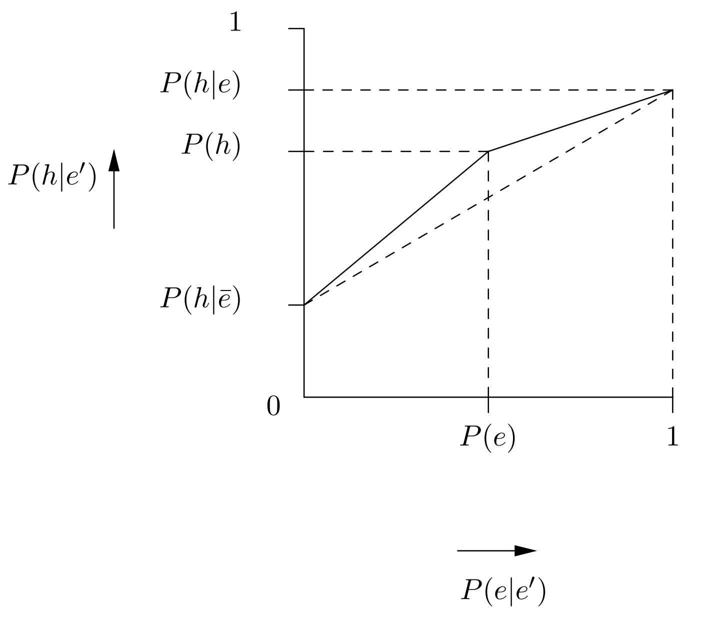
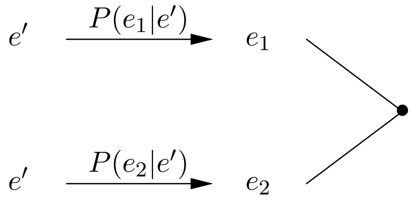


Figure 5.4: A consistent interpolation function.

The effective likelihood ratio λ' lies between λ and $\bar{\lambda}$. λ' will be closer to λ if e has been confirmed to some degree by the observation of the evidence e'; conversely, λ' will be closer to $\bar{\lambda}$ if e has been disconfirmed to some degree by the prior evidence e'.

Until now we have only considered production rules **if** e **then** h **fi** in which e is an atomic piece of evidence. In the foregoing we have seen that the condition part of a production rule may be a combination of atomic pieces of evidence which are interrelated by means of the logical operators **and** and **or**. In the inference network shown below, for example, the evidence e_1 **or** e_2 is depicted; the constituting pieces of evidence have been obtained from prior observations e':



To be able to propagate the uncertainty of the composite evidence e_1 **or** e_2, we have to know the probability $P(e_1$ **or** $e_2 \mid e')$ such that:

$$e' \xrightarrow{P(e_1 \textbf{ or } e_2|e')} e_1 \textbf{ or } e_2$$

Note that the exact probability cannot be computed from the probabilities $P(e_1 \mid e')$ and $P(e_2 \mid e')$ of the separate components. Again, we have to approximate the required probability using a combination function.

Let evidence e be composed of a number of atomic pieces of evidence e_i, $i = 1, \ldots, n, n \geq 2$, which are interrelated by means of **and** and **or**. In PROSPECTOR, the probability $P(e \mid e')$

of e given the prior observations e' is approximated from the separate probabilities $P(e_i \mid e')$ of the constituting pieces of evidence e_i in e by recursively applying the following two functions:

$$\begin{aligned} P(e_1 \textbf{ and } e_2 \mid e') &= \min\{P(e_1 \mid e'), P(e_2 \mid e')\} \\ P(e_1 \textbf{ or } e_2 \mid e') &= \max\{P(e_1 \mid e'), P(e_2 \mid e')\} \end{aligned}$$

These functions therefore fulfill the role of the combination functions for composite hypotheses, that is, of f_{and} and f_{or}, respectively. Note that the order in which the constituting pieces of evidence have been specified does not influence the resulting probability of a composite hypothesis.

The combination function which still remains to be discussed is the function for co-concluding production rules **if** e_i **then** h **fi**, that is, we still have to discuss the function f_{co}. If the pieces of evidence e_i specified in a number of co-concluding production rules have been obtained from prior observations ${e'}_i$, respectively, then the uncertainty of these pieces of evidence e_i given ${e'}_i$ can be propagated to h in the manner described above. For two co-concluding production rules, the resulting inference network is the following:

$$\begin{tikzcd} {e'}_1 \arrow[dr, "\lambda'_1"] & \\ & h \\ {e'}_2 \arrow[ur, "\lambda'_2"'] & \end{tikzcd}$$

Recall that in probability theory Bayes' theorem may be used as the combination function f_{co}. In the subjective Bayesian method, Bayes' theorem is used as well, however, in a somewhat different form in terms of the odds.

THEOREM 6 *Let P be a probability function on a sample space Ω, and let O be the corresponding odds as defined above. Let $h, e \subseteq \Omega$. Furthermore, let the likelihood ratio λ be defined as above. Then, the following property holds:*

$$O(h \mid e) = \lambda \cdot O(h)$$

Proof: From Bayes' theorem we have

$$P(h \mid e) = \frac{P(e \mid h)P(h)}{P(e)}$$

For the complement of h we have, again from Bayes' theorem,

$$P(\bar{h} \mid e) = \frac{P(e \mid \bar{h})P(\bar{h})}{P(e)}$$

Dividing the first equation by the second one results in the following equation:

$$\frac{P(h \mid e)}{P(\bar{h} \mid e)} = \frac{P(e \mid h)P(h)}{P(e \mid \bar{h})P(\bar{h})}$$

from which we have

$$\frac{P(h \mid e)}{1 - P(h \mid e)} = \frac{P(e \mid h)}{P(e \mid \bar{h})} \cdot \frac{P(h)}{1 - P(h)}$$

From this observation it follows that $O(h \mid e) = \lambda \cdot O(h)$. $\Diamond$

This alternative form of Bayes' theorem is called *odds-likelihood form* of the theorem.

The theorem stated above concerns the situation where evidence e has been obtained with absolute certainty. In case we have that e has definitely not occurred, that is, in case $\bar{e}$ has been observed with absolute certainty, we obtain a similar formula.

THEOREM 7 *Let P be a probability function on a sample space Ω, and let O be the corresponding odds as defined in the foregoing. Let $h, e \subseteq \Omega$. Furthermore, let the negative likelihood ratio $\bar{\lambda}$ be defined as above. Then, the following property holds:*

$$O(h \mid \bar{e}) = \bar{\lambda} \cdot O(h)$$

The above theorems apply to the case of a single production rule. In the situation where several production rules **if** e_i **then** h **fi** conclude on the same hypothesis h, the results from these production rules have to be combined into a single measure of uncertainty for h. Again, we first consider the case where all e_i's have been obtained with absolute certainty. It should be evident that by assuming that the e_i's are conditionally independent given h we have that the following property holds:

$$O\left(h \mid \bigcap_{i=1}^{n} e_i\right) = \prod_{i=1}^{n} \lambda_i O(h)$$

where $\lambda_i = \frac{P(e_i|h)}{P(e_i|\bar{h})}$. Similarly, for the case where all $\bar{e}_i$'s have been obtained with absolute certainty, we have:

$$O\left(h \mid \bigcap_{i=1}^{n} \bar{e}_i\right) = \prod_{i=1}^{n} \bar{\lambda}_i O(h)$$

We have argued before that in general the e_i's (or $\bar{e}_i$'s respectively) will not have been obtained with absolute certainty, but with a probability $P(e_i \mid e_i')$ given some prior observations e_i'. From the probabilities $P(e_i \mid e_i')$ the posterior odds $O(h \mid e_i')$ are obtained from applying the combination function for propagating uncertain evidence. From these posterior odds we then compute the effective likelihood ratios λ_i'. Again under the assumption that the e_i''s are conditionally independent given h we obtain:

$$O\left(h \mid \bigcap_{i=1}^{n} e_i'\right) = \prod_{i=1}^{n} \lambda'_i O(h)$$

Since multiplication is commutative and associative, we have that the order in which the co-concluding production rules are applied, will be irrelevant for the resulting uncertainty for h. This finishes our discussion of the subjective Bayesian method.

5.4 The certainty factor model

The certainty factor model has been developed by E.H. Shortliffe and B.G. Buchanan for the purpose of introducing the notion of uncertainty in the MYCIN system. The development of the model was motivated, just as the subjective Bayesian method was, by the problems encountered in applying probability theory in production systems in a straightforward manner. We have suggested before that the model is unfounded from a theoretical point of view.

Nevertheless, the model has since its introduction enjoyed widespread use in rule-based systems built after MYCIN: the model has been used, and is still being used, in a large number of rule-based systems. Even though it is not well-founded, in practice it seems to behave 'satisfactorily'. The relative success of the model can be accounted for by its computational simplicity.

5.4.1 The measures of belief and disbelief

In Section 5.1 it has been argued that when modeling knowledge in production rules of the form **if** e **then** h_x **fi**, a measure of uncertainty x is associated with the hypothesis h expressing the degree to which the observation of evidence e influences the confidence in h. In developing the certainty factor model Shortliffe and Buchanan have chosen two basic measures of uncertainty: the *measure of belief* expressing the degree to which an observed piece of evidence increases the belief in a certain hypothesis, and the *measure of disbelief* expressing the degree to which an observed piece of evidence decreases the belief in a hypothesis. Although both measures are probability based, they model a notion of uncertainty conceptually different from probabilities. According to Shortliffe and Buchanan the need for new notions of uncertainty arose from their observation that an expert often was unwilling to accept the logical implications of his probabilistic statements, such as: if $P(h \mid e) = x$, then $P(\bar{h} \mid e) = 1 - x$. They state that in the mentioned case an expert would claim that 'evidence e in favour of hypothesis h should not be construed as evidence against the hypothesis as well'. The reason that the logical implication concerning $P(\bar{h} \mid e)$ may seem counterintuitive is explained by J. Pearl as follows. The phrase 'evidence e in favour of hypothesis h' is interpreted as stating an *increase* in the probability of the hypothesis from $P(h)$ to $P(h \mid e)$, with $P(h \mid e) > P(h)$: $P(h \mid e)$ is viewed relative to $P(h)$. On the other hand, in the argument of Shortliffe and Buchanan $P(\bar{h} \mid e)$ seems to be taken as an absolute probability irrespective of the prior $P(\bar{h})$. This somehow conveys the false idea that $P(\bar{h})$ increases by some positive factor. However if for example $P(\bar{h}) = 0.9$ and $P(\bar{h} \mid e) = 0.5$, then no expert will construe this considerable decrease in the probability of $\bar{h}$ as supporting the negation of h!

Anyhow, Shortliffe and Buchanan concluded from their observation that the number attached by an expert to a production rule is not a probability, but a measure of belief or disbelief in the hypothesis concerned.

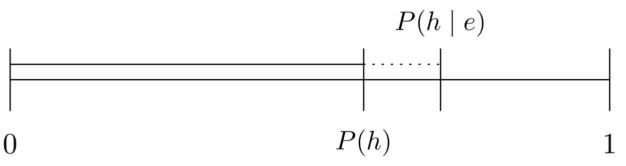
Definition 5.10 *Let P be a probability function defined on a sample space Ω, and let $h, e \subseteq \Omega$ such that $P(e) > 0$. The* measure of (increased) belief MB *is a function* $\mathrm{MB} : 2^{\Omega} \times 2^{\Omega} \to [0,1]$, *such that*

$$\mathrm{MB}(h,e) = \begin{cases} 1 & \text{if } P(h) = 1 \\ \max\left\{0, \dfrac{P(h \mid e) - P(h)}{1 - P(h)}\right\} & \text{otherwise} \end{cases}$$

The measure of (increased) disbelief MD *is a function* $\mathrm{MD} : 2^{\Omega} \times 2^{\Omega} \to [0,1]$, *such that*

$$\mathrm{MD}(h,e) = \begin{cases} 1 & \text{if } P(h) = 0 \\ \max\left\{0, \dfrac{P(h) - P(h \mid e)}{P(h)}\right\} & \text{otherwise} \end{cases}$$

The measure of belief can be accounted for intuitively as follows. Let us depict the prior probability of the hypothesis h, that is, $P(h)$, on a scale from 0 to 1:



The maximum amount of belief that can still be added to the prior belief in h, equals $1-P(h)$. If a piece of evidence e is observed confirming h, that is, such that $P(h \mid e) > P(h)$, then this observation results in adding the amount of belief $P(h \mid e) - P(h)$ to the prior belief in h. The belief in h therefore has been increased to the degree

$$\frac{P(h \mid e) - P(h)}{1 - P(h)}$$

The measure of disbelief can be accounted for similarly.

From the previous definition, it can readily be seen that for a given hypothesis h and a given piece of evidence e only one of the functions MB and MD attains a function value greater than zero. If $\mathrm{MB}(h,e) > 0$, we have either $P(h \mid e) - P(h) > 0$ or $P(h) = 1$. If $P(h \mid e) - P(h) > 0$ then we have $P(h) - P(h \mid e) < 0$ and consequently $\mathrm{MD}(h,e) = 0$. In case $P(h) = 1$, we have that $P(h \mid e) = 1$, hence $P(h) - P(h \mid e) = 0$ and $\mathrm{MD}(h,e) = 0$. Similarly, it can be shown that $\mathrm{MB}(h,e) = 0$ if $\mathrm{MD}(h,e) > 0$. This corresponds explicitly with the idea that a particular piece of evidence may not be used both for as well as against a hypothesis. For evidence e neither confirming nor disconfirming the hypothesis h, that is, evidence e for which $P(h \mid e) = P(h)$ holds, we have $\mathrm{MB}(h,e) = \mathrm{MD}(h,e) = 0$.

We now associate a measure of belief $\mathrm{MB}(h,e)$ and a measure of disbelief $\mathrm{MD}(h,e)$ with a hypothesis h in a production rule **if** e **then** h **fi**, as follows:

$$e \xrightarrow{\mathrm{MB}(h,e),\ \mathrm{MD}(h,e)} h$$

In this rule, the numbers $\mathrm{MB}(h,e)$ and $\mathrm{MD}(h,e)$ have the following meaning: an $\mathrm{MB}(h,e) > 0$ (and hence $\mathrm{MD}(h,e) = 0$) means that the observation of evidence e increases the confidence in h. $\mathrm{MB}(h,e) = 1$ means that the hypothesis h has been fully confirmed by e. An $\mathrm{MD}(h,e) > 0$ (and hence $\mathrm{MB}(h,e) = 0$) indicates that the observation of e tends to disconfirm the hypothesis h. Note that the measures of belief and disbelief MB and MD generally are specified by the domain expert only for a selection of the arguments in their domain. If a probability function on the domain were known, then the other function values of MB and MD could be computed using the respective definitions of these functions. However, we have argued before that such a probability function is virtually never known in practical applications. Similar to the subjective Bayesian method, the certainty factor model therefore offers a number of combination functions for approximating the function values of MB and MD that were not specified beforehand by the expert.

5.4.2 The combination functions

As we have seen before, when applying production rules various intermediate results are derived with a certain measure of uncertainty, which in turn are used as evidence in other production rules. The combination function which will be considered first, is the one for propagating such uncertainty in evidence. Suppose that an intermediate result e has been obtained from earlier evidence e' with a measure of belief $\mathrm{MB}(e, e')$ and a measure of disbelief $\mathrm{MD}(e, e')$. This e is subsequently used as evidence in the production rule **if** e **then** h **fi**:

$$\begin{tikzcd} e' \arrow[r, "{\mathrm{MB}(e,e'),\ \mathrm{MD}(e,e')}"] & e \arrow[r, "{\mathrm{MB}(h,e), \mathrm{MD}(h,e)}"] & h \end{tikzcd}$$

Note once more that the left half of the figure shows a compressed network whereas the right half represents a single production rules. After applying the rule, we are interested in the measure of belief $\mathrm{MB}(h, e')$ and the measure of disbelief $\mathrm{MD}(h, e')$ such that:

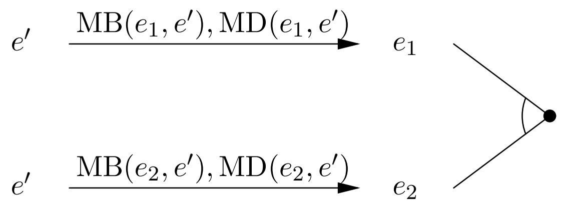
$$\begin{tikzcd} e' \arrow[r, "{\mathrm{MB}(h,e'), \mathrm{MD}(h,e')}"] & h \end{tikzcd}$$

The following combination functions prescribe that the measure of belief of e given e' will be used as a scaling factor for the measures of belief and disbelief associated with the production rule:

$$\begin{aligned} \mathrm{MB}(h, e') &= \mathrm{MB}(h, e) \cdot \mathrm{MB}(e, e') \\ \mathrm{MD}(h, e') &= \mathrm{MD}(h, e) \cdot \mathrm{MB}(e, e') \end{aligned}$$

Herein, $\mathrm{MB}(h, e)$ is the measure of belief to be assigned to the hypothesis h if the piece of evidence e has been fully confirmed; it is the measure of belief associated with h in the production rule **if** e **then** h **fi**. The meaning of $\mathrm{MD}(h, e)$ is analogous. Note that the production rule does not contribute to the belief nor to the disbelief in h if e has been disconfirmed to some extent by evidence e', in other words if the condition e has failed. The certainty factor model in this respect differs conceptually from the subjective Bayesian method.

The condition part of a production rule generally consists of a number of constituent pieces of evidence which are interrelated by means of the operators **and** and **or**. For example, the following inference network represents the composite evidence e_1 **and** e_2 where the constituent pieces of evidence e_1 and e_2 have been derived from some prior evidence e':



e'
$\mathrm{MB}(e_1, e'), \mathrm{MD}(e_1, e')$
e_1
e'
$\mathrm{MB}(e_2, e'), \mathrm{MD}(e_2, e')$
e_2

The certainty factor model comprises a number of combination functions for computing the measure of belief and the measure of disbelief for certain combinations of pieces of evidence. These combination functions are equivalent to the corresponding functions in the subjective Bayesian method:

$$\begin{aligned} \mathrm{MB}(e_1 \textbf{ and } e_2, e') &= \min\{\mathrm{MB}(e_1, e'), \mathrm{MB}(e_2, e')\} \\ \mathrm{MB}(e_1 \textbf{ or } e_2, e') &= \max\{\mathrm{MB}(e_1, e'), \mathrm{MB}(e_2, e')\} \end{aligned}$$

$$\begin{aligned} \mathrm{MD}(e_1 \textbf{ and } e_2, e') &= \max\{\mathrm{MD}(e_1, e'), \mathrm{MD}(e_2, e')\} \\ \mathrm{MD}(e_1 \textbf{ or } e_2, e') &= \min\{\mathrm{MD}(e_1, e'), \mathrm{MD}(e_2, e')\} \end{aligned}$$

The combination functions given above are commutative and associative in the first argument; so, the order in which two constituent pieces of evidence in the condition part of a production rule have been specified, has no influence on the resulting measures of belief and disbelief.

Until now, a production rule has been considered in isolation from the other production rules in a rule base. It is however possible that more than one production rule **if** e_i **then** h **fi** concludes on the same hypothesis h. Each of these different rules results in a separate measure of belief and disbelief for the same hypothesis h. We suppose that the pieces of evidence e_i specified in the co-concluding production rules have been derived from prior evidence e'_i. The uncertainty of the pieces of evidence e_i may be propagated to h in the manner described earlier in this section. For two co-concluding production rules the inference network looks as follows:

$$\begin{tikzcd}
e'_1 \arrow[dr, "{\mathrm{MB}(h,e'_1),\mathrm{MD}(h,e'_1)}"] & \\
& h \\
e'_2 \arrow[ur, "{\mathrm{MB}(h,e'_2),\mathrm{MD}(h,e'_2)}"'] &
\end{tikzcd}$$

These partial measures of belief and disbelief each contribute to the total belief and disbelief in h. The combination functions for co-concluding production rules combine these partial measures of belief and disbelief in order to obtain the total belief and disbelief in h:

$$\mathrm{MB}(h, e'_1 \textbf{ co } e'_2) = \begin{cases} 0 & \text{if } \mathrm{MD}(h, e'_1 \textbf{ co } e'_2) = 1 \\ \mathrm{MB}(h, e'_1) + \mathrm{MB}(h, e'_2)(1 - \mathrm{MB}(h, e'_1)) & \text{otherwise} \end{cases}$$

$$\mathrm{MD}(h, e'_1 \textbf{ co } e'_2) = \begin{cases} 0 & \text{if } \mathrm{MB}(h, e'_1 \textbf{ co } e'_2) = 1 \\ \mathrm{MD}(h, e'_1) + \mathrm{MD}(h, e'_2)(1 - \mathrm{MD}(h, e'_1)) & \text{otherwise} \end{cases}$$

These combination functions are commutative and associative in the second argument, so the order in which the production rules are applied has no effect on the final result.

It should be remarked that the formulas given by Shortliffe and Buchanan as shown above suggest a number of properties of the measures of belief and disbelief which do not hold in general. For instance, it is possible that the measure of belief in h as well as the measure of disbelief in h given prior evidence e' are greater than zero after applying the combination functions for co-concluding production rules, which is contradictory to the original definitions of the functions MB and MD. Only in a small number of special cases under rigorous conditions concerning the interrelationships between the pieces of evidence and the hypotheses, do the properties suggested in the formulas hold. In general, however, the combination functions are not correct with respect to the probabilistic foundation of the model.

5.4.3 The certainty factor function

In the original formulation of the certainty factor model, computation took place in terms of the measures of belief and disbelief; the uncertainties were propagated through the inference network obtained from top-down inference on a set of production rules by using the combination functions discussed above. Soon, however, the need arose to express the finally derived measures of belief and disbelief for a certain hypothesis in a single number. For this purpose, Shortliffe and Buchanan have introduced a new measure derived from the two basic ones mentioned: the certainty factor function.

Definition 5.11 *Let Ω be a sample space, and let $h, e \subseteq \Omega$. Let* MB *and* MD *be defined as in Section 5.4.1. The* certainty factor function CF *is a function* $\text{CF} : 2^\Omega \times 2^\Omega \rightarrow [-1, 1]$*, such that:*

$$\text{CF}(h, e) = \frac{\text{MB}(h, e) - \text{MD}(h, e)}{1 - \min\{\text{MB}(h, e), \text{MD}(h, e)\}}$$

The 'scaling factor' $1 - \min\{\text{MB}(h, e), \text{MD}(h, e)\}$ has been incorporated into the model for pragmatic reasons. This scaling factor has no influence on the certainty factor when considering only one piece of evidence, since then we have $1 - \min\{\text{MB}(h, e), \text{MD}(h, e)\} = 1$. However, when we consider more than one piece of evidence or more than one hypothesis, this is not always the case as has been mentioned before.

Note that for given h and e, a certainty factor is a number between -1 and $+1$; this is contrary to the measures of belief and disbelief, each lying in the closed interval $[0, 1]$. It can easily be seen from the definition given above that a negative certainty factor indicates that the hypothesis is disconfirmed by the evidence and that a positive certainty factor indicates that the hypothesis is confirmed by the evidence. A certainty factor equal to zero indicates that the evidence does not influence the belief in the hypothesis.

In present implementations of the certainty factor model, the measures of belief and disbelief are no longer used in the computation: only the certainty factor is applied instead of the two measures of belief and disbelief $\text{MB}(h, e)$ and $\text{MD}(h, e)$. With each production rule **if** e **then** h **fi** now is associated a certainty factor $\text{CF}(h, e)$:

$$e \xrightarrow{\text{CF}(h, e)} h$$

For manipulating these certainty factors, Shortliffe and Buchanan have defined new combination functions expressed in terms of certainty factors only. A small calculation effort suffices to prove that these combination functions can be derived from the corresponding ones for the measures of belief and disbelief.

The combination function for propagating uncertain evidence is the following:

$$\text{CF}(h, e') = \text{CF}(h, e) \cdot \max\{0, \text{CF}(e, e')\}$$

Here, $\text{CF}(h, e)$ is the certainty factor associated with the hypothesis h by the production rule **if** e **then** h **fi** if the evidence e has been observed with absolute certainty; $\text{CF}(e, e')$ indicates the actual confidence in e based on some prior evidence e'.

The function for combining two certainty factors $\text{CF}(e_1, e')$ and $\text{CF}(e_2, e')$ of two constituting pieces of evidence e_1 and e_2 to obtain a certainty factor for the conjunction e_1 **and** e_2 of these pieces of evidence is the following:

$$\text{CF}(e_1 \textbf{ and } e_2, e') = \min\{\text{CF}(e_1, e'), \text{CF}(e_2, e')\}$$

For the disjunction of these pieces of evidence, we have the following formula:

$$\text{CF}(e_1 \textbf{ or } e_2, e') = \max\{\text{CF}(e_1, e'), \text{CF}(e_2, e')\}$$

Finally, the combination function for combining two certainty factors $\text{CF}(h, e'_1)$ and $\text{CF}(h, e'_2)$ which have been derived from two co-concluding production rules **if** e_i **then** h **fi**, $i = 1, 2$, is as follows:

$$\mathrm{CF}(h, e_1' \textbf{ co } e_2') = \begin{cases} \mathrm{CF}(h, e_1') + \mathrm{CF}(h, e_2')(1 - \mathrm{CF}(h, e_1')) & \text{if } \mathrm{CF}(h, e_i' > 0, i = 1, 2 \\[1em] \dfrac{CF(h, e_1') + \mathrm{CF}(h, e_2')}{1 - \min\{|\mathrm{CF}(h, e_1')|, |\mathrm{CF}(h, e_2')|\}} & \text{if } -1 \leq \mathrm{CF}(h, e_1') \cdot \mathrm{CF}(h, e_2') \leq 0 \\[1em] \mathrm{CF}(h, e_1') + \mathrm{CF}(h, e_2')(1 + \mathrm{CF}(h, e_1')) & \text{if } \mathrm{CF}(h, e_i') < 0, i = 1, 2 \end{cases}$$

The following example demonstrates how these combination functions for certainty factors can be applied.

EXAMPLE 5.7

Consider the following five production rules:

R_1 : **if** a **and** (b **or** c) **then** $h_{0.80}$ **fi**
R_2 : **if** d **and** f **then** $b_{0.60}$ **fi**
R_3 : **if** f **or** g **then** $h_{0.40}$ **fi**
R_4 : **if** a **then** $d_{0.75}$ **fi**
R_5 : **if** i **then** $g_{0.30}$ **fi**

The expert has associated with the conclusion h of rule R_1 the certainty factor CF(h, a **and** (b **or** c)) = 0.80, with the conclusion b of rule R_2 the certainty factor CF(b, d **and** f) = 0.60, and so on. We suppose that h is the goal hypothesis. When applying backward chaining, the user will be asked to provide further information on a, c, f and i. We assume that using his prior knowledge e', the user associates the following certainty factors with his answers:

$$\begin{aligned} CF(a, e') &= 1.00 \\ CF(c, e') &= 0.50 \\ CF(f, e') &= 0.70 \\ CF(i, e') &= -0.40 \end{aligned}$$

Using backward chaining, R_1 will be the first rule selected for application. Note that this rule will eventually yield a partial certainty factor for h. It will be evident that we cannot simply associate the certainty factor 0.80 with h after application of R_1: this number only indicates the certainty of h in case of absolute certainty of a **and** (b **or** c). Recall that for computing the actual certainty of h from this rule, we first have to compute the actual certainty of a **and** (b **or** c) and then propagate it to h using the combination function for uncertain evidence. However, the actual certainty of a **and** (b **or** c) is not known: we have to compute it from the separate certainty factors for a, b, and c using the combination functions for composite hypotheses. The actual certainty factors of a and c are known: the user has specified the certainty factors 1.00 and 0.50 for these pieces of evidence. For b, however, we still have to compute a certainty factor. We select the production rule R_2 for doing so. The combination function for uncertain evidence now prescribes that we have to multiply the certainty factor 0.60 for b mentioned in the rule by the actual certainty factor of the evidence d **and** f. Again, we have to obtain separate certainty factors for d and f. The user has associated the certainty factor 0.70 with f; by applying rule R_4 we find for d the certainty factor $1.00 \cdot 0.75 = 0.75$. Using the combination function for composite hypotheses we arrive at the following certainty factor for d **and** f (we use e_1' to denote all evidence used in this particular reasoning chain):

$$\mathrm{CF}(d \textbf{ and } f, e_1') \quad = \min\{\mathrm{CF}(d, e_1'), \mathrm{CF}(f, e_1')\} = 0.70$$

Subsequently, the combination function for uncertain evidence is applied to compute the actual certainty factor for b:

$$\begin{aligned}\mathrm{CF}(b, e_1') &= \mathrm{CF}(b, d \textbf{ and } f) \cdot \max\{0, \mathrm{CF}(d \textbf{ and } f, e_1')\} = \\ &= 0.60 \cdot 0.70 = 0.42\end{aligned}$$

Recall that we had to compute certainty factors for a, b, and c separately in order to be able to compute a certainty factor for the composite evidence a **and** (b **or** c). All the required certainty factors are now available. We apply the combination function for a disjunction of hypotheses to compute:

$$\mathrm{CF}(b \textbf{ or } c, e_1') \quad = \max\{\mathrm{CF}(b, e_1'), \mathrm{CF}(c, e_1')\} = 0.50$$

And, subsequently, the combination function for a conjunction of hypotheses to compute:

$$\mathrm{CF}(a \textbf{ and } (b \textbf{ or } c), e_1') \quad = \min\{\mathrm{CF}(a, e_1'), \mathrm{CF}(b \textbf{ or } c, e_1')\} = 0.50$$

From the production rule R_1 we therefore obtain the following (partial) certainty factor for h:

$$\begin{aligned}\mathrm{CF}(h, e_1') &= \mathrm{CF}(h, a \textbf{ and } (b \textbf{ or } c)) \cdot \max\{0, \mathrm{CF}(a \textbf{ and } (b \textbf{ or } c), e_1')\} = \\ &= 0.80 \cdot 0.50 = 0.40\end{aligned}$$

Similarly, from the other production rule concluding on h, that is, rule R_3, the following certainty factor is obtained:

$$\begin{aligned}\mathrm{CF}(h, e_2') &= \mathrm{CF}(h, f \textbf{ or } g) \cdot \max\{0, \mathrm{CF}(f \textbf{ or } g, e_2')\} = \\ &= 0.40 \cdot 0.70 = 0.28\end{aligned}$$

In the course of this computation a certainty factor equal to zero is associated with g due to $\mathrm{CF}(i, e') = -0.40$. The net certainty factor for h is computed from the two partial ones by applying the combination function for co-concluding production rules:

$$\begin{aligned}\mathrm{CF}(h, e_1' \textbf{ co } e_2') &= \mathrm{CF}(h, e_1') + \mathrm{CF}(h, e_2') \cdot (1 - \mathrm{CF}(h, e_1')) = \\ &= 0.40 + 0.28 \cdot 0.60 = 0.568\end{aligned}$$

Note that this net certainty factor is greater than each of the certainty factors for h separately.

5.5 The Dempster-Shafer theory

In the 1960s, A. Dempster laid the foundations for a new mathematical theory of uncertainty; in the seventies, this theory was extended by G. Shafer to what at present is known as the *Dempster-Shafer theory.* This theory may be viewed as a generalization of probability theory. Contrary to the subjective Bayesian method and the certainty factor model, Dempster-Shafer theory has not especially been developed for reasoning with uncertainty in knowledge-based systems. Only at the beginning of the eighties, it became apparent that the theory might be suitable for such a purpose. However, the theory cannot be applied in a knowledge-based system without modification. For application in a rule-based system, for example, several combination functions are lacking. Moreover, the theory in its original form has an exponential computational complexity. For rendering it useful in the context of knowledge-based systems, therefore, several modifications of the theory have been proposed. In Sections 5.5.1 and 5.5.2 the main principles of the theory are discussed. Section 5.5.3 briefly touches upon a possible adaptation of the theory for application in a production system.

5.5.1 The probability assignment

We have mentioned above that the Dempster-Shafer theory may be viewed as a generalization of probability theory. The development of the theory has been motivated by the observation that probability theory is not able to distinguish between *uncertainty* and *ignorance* due to incompleteness of information. Recall that in probability theory, probabilities have to be associated with individual atomic hypotheses. Only if these probabilities are known, are we able to compute other probabilities of interest. In the Dempster-Shafer theory, however, it is possible to associate measures of uncertainty with sets of hypotheses, then interpreted as disjunctions, instead of with the individual hypotheses only, and nevertheless be able to make statements concerning the uncertainty of other sets of hypotheses. Note that this way, the theory is able to distinguish between uncertainty and ignorance.

EXAMPLE 5.8

Consider a house officers' practice where a patient consults his physician for chest pain, radiating to the arms and neck; the pain does not disappear in rest. In this simplified example we assume that there are only four possible disorders to be considered as a diagnosis: the patient is either suffering from a heart attack, a pericarditis, pulmonary embolism, or an aortic dissection. Heart attack and pericarditis are disorders of the heart; pulmonary embolism and aortic dissection are disorders of the blood vessels. Now suppose that we have certain clues indicating that the patient has a disorder of the heart; the strength of our belief is expressed in the number 0.4. In the Dempster-Shafer theory this number is assigned to the set *heart-attack*, *pericarditis*, viewed as the composite hypothesis *heart-attack* **or** *pericarditis*; there is no number associated with the individual hypotheses, since more specific information indicating that one of these two hypotheses is the cause of the complaints, is not available. Note that in probability theory the number 0.4 would have to be distributed over the individual hypotheses (without more information, each of the two hypotheses would be assigned the number 0.2). In that case, the false impression of more information than actually present would be given.

The strategy followed in the Dempster-Shafer theory for dealing with uncertainty roughly amounts to the following: starting with an initial set of hypotheses, due to pieces of evidence each time a measure of uncertainty is associated with certain subsets of the original set of hypotheses, until measures of uncertainty may be associated with all possible subsets on account of the combined evidence. The initial set of all hypotheses in the problem domain is called the *frame of discernment*. In such a frame of discernment the individual hypotheses are assumed to be disjoint. The impact of a piece of evidence on the confidence or belief in certain subsets of a given frame of discernment is described by means of a function which is defined below.

Definition 5.12 *Let* Θ *be a frame of discernment. If with each subset* $x \subseteq \Theta$ *a number* $m(x)$ *is associated such that:*

(1) $m(x) \geq 0$,

(2) $m(\varnothing) = 0$, *and*

(3) $\sum_{x \subseteq \Theta} m(x) = 1$

then m *is called a* basic probability assignment *on* Θ*. For each subset* $x \subseteq \Theta$*, the number* $m(x)$ *is called the* basic probability number *of* x*.*

We define another two notions.

Definition 5.13 *Let* Θ *be a frame of discernment and let* m *be a basic probability assignment on* Θ*. A set* $x \subseteq \Theta$ *is called a* focal element *in* m *if* $m(x) > 0$*. The* core *of* m*, denoted by* $\kappa(m)$*, is the set of all focal elements in* m*.*

Note the similarity between a basic probability assignment and a probability function. A probability function associates with each element in Θ a number from the interval [0,1] such that the sum of these numbers equals 1; a basic probability assignment associates with each element in 2^{Θ} a number in the interval [0,1] such that once more the sum of the numbers equals 1.

EXAMPLE 5.9

Consider the preceding medical example once more. In this example, the frame of discernment is the set $\Theta = \{$*heart-attack*, *pericarditis*, *pulmonary-embolism*, *aortic-dissection*$\}$. Note that each basic probability assignment on Θ assigns basic probability numbers to $2^4 = 16$ sets (including the empty set). If for a specific patient there is no evidence pointing at a certain diagnosis in particular, then the basic probability number 1 is assigned to the entire frame of discernment:

$$m_0(x) = \begin{cases} 1 & \text{if } x = \Theta \\ 0 & \text{otherwise} \end{cases}$$

Note that each proper subset of the frame of discernment gets assigned the number 0. The core of m_0 is equal to Θ. Now suppose that some evidence has become available that points to the composite hypothesis *heart-attack* **or** *pericarditis* with some certainty. Then, the subset $\{$*heart-attack*, *pericarditis*$\}$ will be assigned a basic probability number, for example 0.4. Due to lack of further information, the remaining certainty 0.6 is assigned to the entire frame of discernment:

$$m_1(x) = \begin{cases} 0.6 & \text{if } x = \Theta \\ 0.4 & \text{if } x = \{\textit{heart-attack}, \textit{pericarditis}\} \\ 0 & \text{otherwise} \end{cases}$$

The set $\{\textit{heart-attack}, \textit{pericarditis}\}$ is an element of the core of m_1. Now suppose that we furthermore have obtained some evidence against the hypothesis that our patient is suffering from pericarditis. This information can be considered as support for the hypothesis that the patient is *not* suffering from pericarditis. This latter hypothesis is equivalent to the composite hypothesis *heart-attack* **or** *pulmonary-embolism* **or** *aortic-dissection*. In consequence of this evidence, we therefore assign a basic probability number, for example 0.7, to the set *heart-attack*, *pulmonary-embolism*, *aortic-dissection*:

$$m_2(x) = \begin{cases} 0.3 & \text{if } = \Theta \\ 0.7 & \text{if } x = \{\textit{heart-attack}, \textit{pulmonary-embolism}, \textit{aortic-dissection}\} \\ 0 & \text{otherwise} \end{cases}$$

A probability number $m(x)$ expresses the confidence or belief assigned to precisely the set x: it does not express any belief in subsets of x. It will be evident, however, that the total confidence in x is not only dependent upon the confidence in x itself, but also upon the confidence assigned to subsets of x. For a given basic probability assignment, we define a new function describing the cumulative belief in a set of hypotheses.

Definition 5.14 *Let Θ be a frame of discernment, and let m be a basic probability assignment on Θ. Then, the* belief function *(or* credibility function*) corresponding to m is the function* $\text{Bel} : 2^\Theta \to [0,1]$ *defined by*

$$\text{Bel}(x) = \sum_{y \subseteq x} m(y)$$

for each $x \subseteq \Theta$.

Several properties of this belief function can easily be proven:

- $\text{Bel}(\Theta) = 1$ since $\sum_{y \subseteq \Theta} m(y) = 1$.
- For each $x \subseteq \Theta$ containing exactly one element, we have that $\text{Bel}(x) = m(x)$.
- For each $x \subseteq \Theta$, we have $\text{Bel}(x) + \text{Bel}(\bar{x}) \leq 1$, since

$$\text{Bel}(\Theta) = \text{Bel}(x \cup \bar{x}) = \text{Bel}(x) + \text{Bel}(\bar{x}) + \sum_{\substack{x \cap y \neq \varnothing \\ \bar{x} \cap y \neq \varnothing}} m(y) = 1$$

We furthermore have the inequality $\text{Bel}(x) + \text{Bel}(y) \neq \text{Bel}(x \cup y)$ for each $x, y \in \Theta$.

We define some special belief functions. In the preceding example, we have demonstrated how complete ignorance may be expressed. Recall that a basic probability assignment describing lack of evidence had the following form:

$$m(x) = \begin{cases} 1 & \text{if } x = \Theta \\ 0 & \text{otherwise} \end{cases}$$

The belief function corresponding to such an assignment has been given a special name.

Definition 5.15 *Let Θ be a frame of discernment and let m be a basic probability assignment such that $\kappa(m) = \{\Theta\}$. The belief function corresponding to m is called a* vacuous belief function*.*

The following definition concerns belief functions corresponding to basic probability assignments of the form

$$m(x) = \begin{cases} 1 - c_1 & \text{if } x = \Theta \\ c_1 & \text{if } x = A \\ 0 & \text{otherwise} \end{cases}$$

where $A \subseteq \Theta$, and $0 \leq c_1 \leq 1$ is a constant.

Definition 5.16 *Let Θ be a frame of discernment and let m be a basic probability assignment such that $\kappa(m) = \{A, \Theta\}$ for a certain $A \subset \Theta$. The belief function corresponding to m is called a* simple support function*.*

A belief function provides for each set x only a lower bound to the 'actual' belief in x: it is also possible that belief has been assigned to a set y such that $x \subseteq y$. Therefore, in addition to the belief function the Dempster-Shafer theory defines another function corresponding to a basic probability assignment.

Definition 5.17 *Let Θ be a frame of discernment and let m be a basic probability assignment on Θ. Then, the* plausibility function *corresponding to m is the function* $\mathrm{Pl} : 2\Theta \rightarrow [0, 1]$ *defined by*

$$\mathrm{Pl}(x) = \sum_{x \cap y \neq \varnothing} m(y)$$

for each $x \subseteq \Theta$.

A function value $\mathrm{Pl}(x)$ indicates the total confidence *not* assigned to $\bar{x}$. So, $\mathrm{Pl}(x)$ provides an upper bound to the 'real' confidence in x. It can easily be shown that for a given basic probability assignment m, the property

$$\mathrm{Pl}(x) = 1 - \mathrm{Bel}(\bar{x})$$

for each $x \subseteq \Theta$, holds for the the belief function Bel and the plausibility function Pl corresponding to m. The difference $\mathrm{Pl}(x) - \mathrm{Bel}(x)$ indicates the confidence in the sets y for which $x \subseteq y$ and therefore expresses the uncertainty with respect to x.

Definition 5.18 *Let Θ be a frame of discernment and let m be a basic probability assignment on Θ. Let* Bel *be the belief function corresponding to m, and let* Pl *be the plausibility function corresponding to m. For each $x \subseteq \Theta$, the closed interval* $[\mathrm{Bel}(x), \mathrm{Pl}(x)]$ *is called the* belief interval *of x.*

EXAMPLE 5.10

Let Θ be a frame of discernment, and let $x \subseteq \Theta$. Now, consider a basic probability assignment m on Θ and its corresponding functions Bel and Pl.

- If $[\mathrm{Bel}(x), \mathrm{Pl}(x)] = [0, 1]$, then no information concerning x is available.
- If $[\mathrm{Bel}(x), \mathrm{Pl}(x)] = [1, 1]$, then x has been completely confirmed by m.
- If $[\mathrm{Bel}(x), \mathrm{Pl}(x)] = [0.3, 1]$, then there is some evidence in favour of the hypothesis x.
- If $[\mathrm{Bel}(x), \mathrm{Pl}(x)] = [0.15, 0.75]$, then we have evidence in favour as well as against x.

If we have $\mathrm{Pl}(x) - \mathrm{Bel}(x) = 0$ for each $x \subseteq \Theta$, then we are back at conventional probability theory. In such a case, the belief function is called a Bayesian belief function. This notion is defined more formally in the following definition.

Definition 5.19 *Let Θ be a frame of discernment and let m be a basic probability assignment such that the core of m only consists of singleton sets. The belief function corresponding to m is called a* Bayesian belief function.

5.5.2 Dempster's rule of combination

The Dempster-Shafer theory provides a function for computing from two pieces of evidence and their associated basic probability assignment a new basic probability assignment describing the combined influence of these pieces of evidence. This function is known as *Dempster's rule of combination*. The remainder of this section is devoted to an example of the use of this function. First, however, it is defined formally in the following definition.

Definition 5.20 (Dempster's rule of combination) *Let Θ be a frame of discernment, and let m_1 and m_2 be basic probability assignments on Θ. Then, $m_1 \oplus m_2$ is a function $m_1 \oplus m_2 : 2^\Theta \to [0, 1]$ such that*

(1) $m_1 \oplus m_2(\varnothing) = 0$, *and*

(2) for all $x \neq \varnothing$:

$$m_1 \oplus m_2(x) = \frac{\sum_{y \cap z = x} m_1(y) \cdot m_2(z)}{\sum_{y \cap z \neq \varnothing} m_1(y) \cdot m_2(z)}$$

$\mathrm{Bel}_1 \oplus \mathrm{Bel}_2$ *is the function* $\mathrm{Bel}_1 \oplus \mathrm{Bel}_2 : 2^\Theta \to [0, 1]$ *defined by*

$$\mathrm{Bel}_1 \oplus \mathrm{Bel}_2(x) = \sum_{y \subseteq x} m_1 \oplus m_2(y)$$

The usage of Dempster's rule of combination will now be illustrated by means of an example.

EXAMPLE 5.11

Consider once more the frame of discernment $\Theta = \{\textit{heart-attack}, \textit{pericarditis}, \textit{pulmonary-embolism}, \textit{aortic-dissection}\}$. Furthermore, consider the basic probability assignment m_1 obtained from the evidence that a given patient suffers from a heart attack or a pericarditis, and the basic probability assignment m_2 obtained from the evidence that the patient does not suffer from pericarditis. These functions are shown below:

$$m_1(x) = \begin{cases} 0.6 & \text{if } x = \Theta \\ 0.4 & \text{if } x = \{\textit{heart-attack}, \textit{pericarditis}\} \\ 0 & \text{otherwise} \end{cases}$$

$$m_2(x) = \begin{cases} 0.3 & \text{if } x = \Theta \\ 0.7 & \text{if } x = \{\textit{heart-attack}, \textit{pulmonary-embolism}, \textit{aortic-dissection}\} \\ 0 & \text{otherwise} \end{cases}$$

From applying Dempster's rule of combination, we obtain a new basic probability assignment $m_1 \oplus m_2$, describing the combined effect of m_1 and m_2. The basic principle of this rule is demonstrated in Figure 5.5; such a figure is called an *intersection tableau.* In front of each row of the intersection tableau is specified a subset of the frame of discernment and the basic probability number assigned to it by the basic probability assignment m_1; the figure shows only those subsets having a basic probability number not equal to zero. Above the columns of the intersection tableau again all subsets of Θ are specified, but this time with their basic probability numbers according to m_2. The crossing of a row and a column now contains the intersection of the sets associated with the row and column concerned, and specifies the product of the two basic probability numbers associated with these sets. So, at the crossing of the row corresponding to the set $\{\textit{heart-attack}, \textit{pericarditis}\}$ having the basic probability number 0.4, and the column corresponding to the set $\{\textit{heart-attack}, \textit{pulmonary-embolism}, \textit{aortic-dissection}\}$ with the basic probability number 0.7, we find the set $\{\textit{heart-attack}\}$ with the number 0.28.

Now observe that the set $\{\textit{heart-attack}\}$ is also present at other places in the tableau since there are various possibilities for choosing two sets $x, y \subseteq \Theta$ such that $x \cap y = \{\textit{heart-attack}\}$. Dempster's rule of combination now sums all basic probability numbers assigned to the set $\{\textit{heart-attack}\}$. The result of this computation (possibly after normalization to 1; we shall return to this shortly) is the basic probability number assigned by $m_1 \oplus m_2$ to that specific set. The intersection tableau in Figure 5.5 shows all sets having a probability number not equal to zero. So, we have obtained the following probability assignment:

$$m_1 \oplus m_2(x) = \begin{cases} 0.18 & \text{if } x = \Theta \\ 0.28 & \text{if } x = \{\textit{heart-attack}\} \\ 0.12 & \text{if } x = \{\textit{heart-attack}, \textit{pericarditis}\} \\ 0.42 & \text{if } x = \{\textit{heart-attack}, \textit{pulmonary-embolism}, \textit{aortic-dissection}\} \\ 0 & \text{otherwise} \end{cases}$$

m_2 / m_1	$\cdots$	{*heart-attack*, *pulmonary-embolism*, *aortic-dissection*} (0.7)	$\cdots$	Θ (0.3)
$\cdots$				
{*heart-attack*, *pericarditis*} (0.4)		{*heart-attack*} (0.28)		{*heart-attack*, *pericarditis*} (0.12)
$\cdots$				
Θ (0.6)		{*heart-attack*, *pulmonary-embolism*, *aortic-dissection*} (0.42)		Θ (0.18)

Figure 5.5: Intersection tableau for m_1 and m_2.

m_3 / m_1	$\cdots$	{*pulmonary-embolism*} (0.5)	$\cdots$	Θ (0.5)
$\cdots$				
{*heart-attack*, *pericarditis*} (0.4)		$\varnothing$ (0.2)		{*heart-attack*, *pericarditis*} (0.2)
$\cdots$				
Θ (0.6)		{*pulmonary-embolism*} (0.3)		Θ (0.3)

Figure 5.6: An *erroneous* intersection tableau for m_1 en m_3.

However, in computing the combination of the two basic probability assignments, as demonstrated above, we may encounter a problem.

Consider m_1 once more and the basic probability assignment m_3 defined by

$$m_3(x) = \begin{cases} 0.5 & \text{if } x = \Theta \\ 0.5 & \text{if } x = \{\mathit{pulmonary\text{-}embolism}\} \\ 0 & \text{otherwise} \end{cases}$$

Figure 5.6 now shows an intersection tableau which has been constructed using the same procedure as before. However, in this *erroneous* intersection tableau a basic probability assignment greater than zero has been assigned to the empty set: we have that $m_1 \oplus m_3(\varnothing) = 0.2$. So, the function $m_1 \oplus m_3$ is not a basic probability assignment, since it does not satisfy the axiom $m_1 \oplus m_3(\varnothing) = 0$. Dempster's rule of combination now simply sets $m_1 \oplus m_3(\varnothing) = 0$. As a consequence, the second axiom is violated: we now have that

$$\sum_{x \subseteq \Theta} m_1 \oplus m_3(x)$$

is less than instead of equal to 1. To remedy this problem, Dempster's rule of combination divides the remaining numbers by the scaling factor

$$\sum_{x \cap y \neq \varnothing} m_1(x) \cdot m_3(y)$$

in this example the factor 0.8. The correct intersection tableau for m_1 and m_3 is depicted in Figure 5.7.

5.5.3 Application in rule-based systems

In the preceding subsections, we have paid some attention to the principle notions of the Dempster-Shafer theory. These principles have been dealt with separate from an application in a knowledge-based system since the theory in its original form is not directly applicable as a model for plausible reasoning in this context. However, in the early eighties, research was initiated to further elaborate the model to render it suitable for application in a knowledge-based system. We have mentioned before that the basic problems preventing the use of the model in rule-based systems are its computational complexity and the lack of several combination functions. In this book, we shall not discuss the complexity problem. With respect to the second problem, various ad-hoc solutions have been proposed none of which is really satisfactory. One of these ad-hoc solutions will be briefly discussed just to illustrate the problems one encounters in providing for the missing combination functions. The simple approach sketched here has been developed by M. Ishizuka for the knowledge-based system SPERIL.

We consider a production rule **if** e_1 **then** h **fi**. The Dempster-Shafer theory does not prescribe explicitly which information should be associated with the hypothesis h of this production rule. It is rather straightforward, however, to associate a basic probability assignment with the rule. If the rule **if** e_1 **then** h **fi** is meant to express that the hypothesis h is confirmed

m_3 / m_1	$\cdots$	$\{$*pulmonary-embolism*$\}$ (0.5)	$\cdots$	Θ (0.5)
$\cdots$				
$\{$*heart-attack*, *pericarditis*$\}$ (0.4)		$\varnothing$ (0)		$\{$*heart-attack*, *pericarditis*$\}$ (0.25)
$\cdots$				
Θ (0.6)		$\{$*pulmonary-embolism*$\}$ (0.375)		Θ (0.375)

Figure 5.7: The *correct* intersection tableau for m_1 and m_3.

with certainty c_1 if the evidence e_1 has been observed with absolute certainty, then a basic probability assignment m_{e_1} such that

$$m_{e_1}(x) = \begin{cases} 1 - c_1 & \text{if } x = \Theta \\ c_1 & \text{if } x = h \\ 0 & \text{otherwise} \end{cases}$$

is associated with the hypothesis of the rule. Note that the corresponding belief function Bel_{e_1} is a simple support function. So, we have

$$\begin{tikzcd} e_1 \arrow[r, "m_{e_1}"] & h \end{tikzcd}$$

Recall from Section 5.1 that plausible reasoning in a rule-based system requires the presence of a number of combination functions: a combination function for propagating uncertain evidence, a combination function for co-concluding production rules, and two combination functions for composite hypotheses. In the Dempster-Shafer theory in its original form, only the combination function for co-concluding production rules is available; we shall see that Dempster's rule of combination may be viewed as such. Consider again the production rule **if** e_1 **then** h **fi** given above and its associated functions m_{e_1} en Bel_{e_1}. Furthermore, suppose that we have a second rule **if** e_2 **then** h **fi** also concerning the hypothesis h, with the following associated basic probability assignment:

$$m_{e_2}(x) = \begin{cases} 1 - c_2 & \text{if } x = \Theta \\ c_2 & \text{if } x = h \\ 0 & \text{otherwise} \end{cases}$$

This situation is shown in the following inference network:

$$\begin{tikzcd}
e_1 \arrow[dr, "m_{e_1}"] & \\
& h \\
e_2 \arrow[ur, "m_{e_2}"'] &
\end{tikzcd}$$

If we assume that e_1 and e_2 have been observed with complete certainty, then the basic probability assignment that will be associated with h based on e_1 and e_2 is equal to $m_{e_1} \oplus m_{e_2}$. The other three combination functions unfortunately are lacking in the Dempster-Shafer theory.

M. Ishizuka has augmented the Dempster-Shafer theory by providing combination functions for use in his system SPERIL. We first consider the combination function for propagating uncertain evidence. Suppose that we are given a production rule **if** e **then** h **fi** with which a basic probability assignment m_e has been associated. We have seen in Section 5.1, that the evidence e is not always established with complete certainty since e itself may have been derived from applying other production rules. For example, e may have been confirmed with a measure of uncertainty $\text{Bel}_{e'}(e)$ on account of some prior evidence e':

$$\begin{tikzcd}
e' \arrow[r, "\text{Bel}_{e'}(e)"] & e \arrow[r, "m_e"] & h
\end{tikzcd}$$

In this situation we are interested in $\text{Bel}_{e'}(h)$, the actual measure of uncertainty of h after application of the production rule shown above. This $\text{Bel}_{e'}(h)$ may be obtained from $m_{e'}(h)$ which is computed as follows:

$$m_{e'}(h) = m_e(h) \cdot \text{Bel}_{e'}(e)$$

Note that this provides us with a combination function for uncertain evidence. The following functions are employed in SPERIL as combination functions for composite hypotheses:

$$\begin{aligned}
\text{Bel}_{e'}(e_1 \textbf{ and } e_2) &= \min\{\text{Bel}_{e'}(e_1), \text{Bel}_{e'}(e_2)\} \\
\text{Bel}_{e'}(e_1 \textbf{ or } e_2) &= \max\{\text{Bel}_{e'}(e_1), \text{Bel}_{e'}(e_2)\}
\end{aligned}$$

The approach to applying Dempster-Shafer theory in a rule-based setting as sketched in this section is simple, but hardly satisfying. We have mentioned before that in the recent literature, several other approaches have been proposed, none of which is really satisfactory. We chose to discuss Ishizuka's method merely because of its simplicity and its obvious similarity to the quasi-probabilistic models treated earlier in this chapter.

5.6 Bayesian networks

In the mid-1980s a new trend in probabilistic reasoning with uncertainty in knowledge-based systems became discernable taking a graphical representation of knowledge as a point of departure. We use the phrase *network models* to denote this type of model In the preceding sections, we have concentrated primarily on models for plausible reasoning that were developed especially for knowledge-based systems using production rules for knowledge representation.

In contrast, the network models depart from another knowledge-representation formalism: the so-called *Bayesian network*. Common synonyms for the formalism are: belief network, probabilistic network, Bayesian belief network, and causal probabilistic network. Informally speaking, a Bayesian network is a graphical representation of a problem domain consisting of the statistical variables discerned in the domain and their probabilistic interrelationships. The relationships between the statistical variables are quantified by means of 'local' probabilities together defining a total probability function on the variables. This section presents a brief introduction to network models. In Section 5.6.1 we shall discuss the way knowledge is represented in a Bayesian network. The Sections 5.6.3 and 5.6.4 discuss two approaches to reasoning with such a network.

5.6.1 Knowledge representation in a Bayesian network

We have mentioned before that Bayesian networks provide a formalism for representing a problem domain. A Bayesian network comprises two parts: a *qualitative representation* of the problem domain and an associated *quantitative representation*. The qualitative part takes the form of an acyclic directed graph $G = (V(G), A(G))$ where $V(G) = \{V_1, \ldots, V_n\}$, $n \geq 1$, is a finite set of *vertices* and $A(G)$ is a finite set of arcs (V_i, V_j), $V_i, V_j \in V(G)$. Each vertex V_i in $V(G)$ represents a statistical variable which in general can take one of a set of values. In the sequel, however, we shall assume for simplicity's sake that the statistical variables can take only one of the truth values *true* and *false*. We take an arc $(V_i, V_j) \in A(G)$ to represent a direct 'influential' or 'causal' relationship between the variables V_i and V_j: the arc (V_i, V_j) is interpreted as stating that 'V_i directly influences V_j'. Absence of an arc between two vertices means that the corresponding variables do not influence each other directly. In general, such a directed graph has to be configured by a domain expert from human judgment; hence the phrase *belief* network. We give an example of such a qualitative representation of a problem domain.

EXAMPLE 5.12

Consider the following qualitative medical information:

> Shortness-of-breath (V_7) may be due to tuberculosis (V_2), lung cancer (V_4) or bronchitis (V_5), or more than one of them. A recent visit to Asia (V_1) increases the chances of tuberculosis, while smoking (V_3) is known to be a risk factor for both lung cancer and bronchitis. The results of a single chest X-ray (V_8) do not discriminate between lung cancer and tuberculosis (V_6), as neither does the presence or absence of shortness-of-breath.

In this information, we may discern several statistical variables; with each variable we have associated a name V_i. The information has been represented in the acyclic directed graph G shown in Figure 5.8. Each vertex in G represents one of the statistical variables, and the arcs in G represent the causal relationships between the variables. The arc between the vertices V_3 and V_4 for example represents the information that smoking may cause lung cancer. Note that although the graph only depicts direct causal relationships, we can read indirect influences from it. For example, the graph shows that V_3 influences V_7 indirectly through V_4, V_5 and V_6: smoking may cause lung cancer and bronchitis, and these may in turn cause shortness-of-breath. However, as

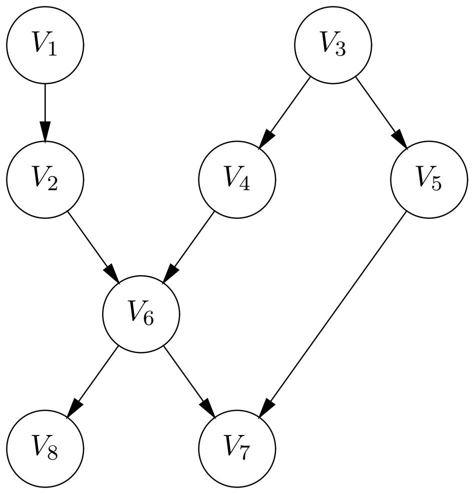


Figure 5.8: The acyclic directed graph of a Bayesian network.

soon as V_4, V_5 and V_6 are known, V_3 itself does not provide any further information concerning V_7.

The qualitative representation of the problem domain now is interpreted as the representation of all probabilistic dependency and independency relationships between the statistical variables discerned. With the graph, a domain expert associates a numerical assessment of the 'strengths' of the represented relationships in terms of a probability function P on the sample space defined by the statistical variables. Before discussing this in further detail, we introduce the notions of predecessor and successor.

Definition 5.21 *Let $G = (V(G), A(G))$ be a directed graph. Vertex $V_j \in V(G)$ is called a* successor *of vertex $V_i \in V(G)$ if there is an arc $(V_i, V_j) \in A(G)$; alternatively, vertex V_i is called a* predecessor *of vertex V_j. A vertex V_k is a neighbour of V_i if V_k is either a successor or a predecessor of V_i.*

Now, for each vertex in the graphical part of a Bayesian network, a set of (conditional) probabilities describing the influence of the values of the predecessors of the vertex on the values of the vertex itself, is specified. We shall illustrate the idea with the help of our example shortly.

We introduce some new notions and notational conventions. From now on, the variable V_i taking the truth value *true* will be denoted by v_i; the probability that the variable V_i has the value *true* will then be denoted by $P(v_i)$. We use $\neg v_i$ to denote that $V_i =$ *false*; the probability that $V_i =$ *false* then is denoted by $P(\neg v_i)$. Now, let $V(G) = \{V_1, \ldots, V_n\}$, $n \geq 1$, again be the set of all statistical variables discerned in the problem domain. We consider a subset $V \subseteq V(G)$ with $m \geq 1$ elements. A conjunction of length m in which for each $V_i \in V$ either v_i or $\neg v_i$ occurs, is called a *configuration* of V. The conjunction $v_1 \wedge \neg v_2 \wedge v_3$ is an example of a configuration of the set $V = \{V_1, V_2, V_3\}$. The conjunction of length m in which each $V_i \in V$ is named only, that is, specified without its value, is called the *configuration template* of V. For example, the configuration template of $V = \{V_1, V_2, V_3\}$ is $V_1 \wedge V_2 \wedge V_3$. Note that we can obtain the configuration $v_1 \wedge \neg v_2 \wedge v_3$ from the template $V_1 \wedge V_2 \wedge V_3$ by filling in v_1, $\neg v_2$, and v_3 for the variables V_1, V_2, and V_3, respectively. In fact, every possible configuration of a set V can be obtained from its template by filling in proper values for the variables occurring in the template.

We return to the quantitative part of a Bayesian network. With each variable, that is, with each vertex $V_i \in V(G)$ in the qualitative part of the belief network, a domain expert associates conditional probabilities $P(v_i \mid c)$ for all configurations c of the set of predecessors of V_i in the graph. Note that for a vertex with m incoming arcs, 2^m probabilities have to be assessed; for a vertex V_i with zero predecessors, only one probability has to be specified, namely the prior probability $P(v_i)$.

EXAMPLE 5.13

Consider the medical information from the previous example and its graphical representation in Figure 5.8 once more. For example, with the vertex V_3 the domain expert associates the prior probability that a patient smokes. For the vertex V_4 two conditional probabilities have to be specified: the probability that a patient has lung cancer given the information that he smokes, that is, the probability $P(v_4 \mid v_3)$, and the probability that a non-smoker gets lung cancer, that is, the probability $P(v_4 \mid \neg v_3)$. Corresponding with the graph, a domain expert therefore has to assess the following eighteen probabilities:

$P(v_1)$
$P(v_2 \mid v_1)$ and $P(v_2 \mid \neg v_1)$
$P(v_3)$
$P(v_4 \mid v_3)$ and $P(v_4 \mid \neg v_3)$
$P(v_5 \mid v_3)$ and $P(v_5 \mid \neg v_3)$
$P(v_6 \mid v_2 \wedge v_4), P(v_6 \mid v_2 \wedge \neg v_4), P(v_6 \mid \neg v_2 \wedge v_4)$, and $P(v_6 \mid \neg v_2 \wedge \neg v_4)$
$P(v_7 \mid v_5 \wedge v_6), P(v_7 \mid v_5 \wedge \neg v_6), P(v_7 \mid \neg v_5 \wedge v_6)$, and $P(v_7 \mid \neg v_5 \wedge \neg v_6)$
$P(v_8 \mid v_6)$ and $P(v_8 \mid \neg v_6)$

Note that from these probabilities we can uniquely compute the 'complementary' probabilities; for example, we have that $P(\neg v_7 \mid v_5 \wedge v_6) = 1 - P(v_7 \mid v_5 \wedge v_6)$.

We observe that a probability function P on a sample space defined by n statistical variables $V_1, \dots, V_n$, $n \geq 1$, is completely described by the probabilities $P(c)$ for all configurations c of $V(G) = \{V_1, \dots, V_n\}$. The reader can easily verify that from these probabilities any other probability may be computed using the axioms mentioned in Section 5.2.1. In the sequel, therefore, we will frequently use the template $P(V_1 \wedge \cdots \wedge V_n)$ to denote a probability function: note that from this template we can obtain the probabilities $P(c)$ for all configurations c of $V(G)$, from which we can compute any probability of interest. Since there are 2^n different configurations c of $V(G)$, in theory 2^n probabilities $P(c)$ are necessary for defining a probability function. In a belief network, however, often far less probabilities suffice for doing so: an important property is that under the assumption that the graphical part of a Bayesian network represents *all* independency relationships between the statistical variables discerned, the probabilities associated with the graph provide enough information to define a unique probability function on the domain of concern. To be more precise, we have

$$P(V_1 \wedge \cdots \wedge V_n) = \prod_{i=1}^{n} P(V_i \mid C_{\rho(V_i)})$$

where $C_{\rho(V_i)}$ is the configuration template of the set $\rho(V_i)$ of predecessors of V_i. Note that the probability of any configuration of $V(G)$ can be obtained by filling in proper values for the statistical variables V_1 up to V_n inclusive and then computing the resulting product on the right-hand side from the initially assessed probabilities. We look again at our example.

EXAMPLE 5.14

Consider the previous examples once more. We have that

$$\begin{aligned} P(V_1 \wedge \cdots \wedge V_8) &= P(V_8 \mid V_6) \cdot P(V_7 \mid V_5 \wedge V_6) \cdot P(V_6 \mid V_2 \wedge V_4) \cdot \cdot P(V_5 \mid V_3) \cdots \\ & \quad P(V_4 \mid V_3) \cdot P(V_3) \cdot P(V_2 \mid V_1) \cdot P(V_1) \end{aligned}$$

Note that in this example only eighteen probabilities suffice for specifying a probability function on our problem domain.

In a Bayesian network, the quantitative representation of the problem domain only comprises probabilities that involve a vertex and its predecessors in the qualitative part of the network. Note that the representation of uncertainty in such local factors closely resembles the approach followed in the quasi-probabilistic models in which uncertainty is represented in factors that are local to the production rules constituting the qualitative representation of the domain.

5.6.2 Evidence propagation in a Bayesian network

In the preceding section we have introduced the notion of a Bayesian network as a means for representing a problem domain. Such a Bayesian network may be used for reasoning with uncertainty, for example for interpreting pieces of evidence that become available during a consultation. For making probabilistic statements concerning the statistical variables discerned in the problem domain, we have to associate with a Bayesian network two methods:

- A method for computing probabilities of interest from the Bayesian network.
- A method for processing evidence, that is, a method for entering evidence into the network and subsequently computing the conditional probability function given this evidence. This process is generally called *evidence propagation*.

In the relevant literature, the emphasis lies on methods for evidence propagation; in this chapter we do so likewise.

Now recall that the probabilities associated with the graphical part of a Bayesian network uniquely define a probability function on the sample space defined by the statistical variables discerned in the problem domain. The impact of a value of a specific variable becoming known on each of the other variables, that is, the conditional probability function given the evidence, can therefore be computed from these initially assessed local probabilities. The resulting conditional probability function is often called the *updated probability function*. Calculation of a conditional probability from the initially given probabilities in a straightforward manner will generally not be restricted to computations which are local in terms of the graphical part of the Bayesian network. Furthermore, the computational complexity of such an approach is exponential in the number of variables: the method will become prohibitive for larger networks. In the literature, therefore, several less naive schemes for updating a probability function as evidence becomes available have been proposed. Although all methods build on the same notion of a Bayesian network, they differ considerably in concept and in computational complexity. All schemes proposed for evidence propagation however have two important characteristics in common:

- For propagating evidence, the graphical part of a Bayesian network is exploited more or less directly as a computational architecture.
- After a piece of evidence has been processed, again a Bayesian network results. Note that this property renders the notion of a Bayesian network invariant under evidence propagation and therefore allows for recursive application of the method for processing evidence.

In the following two sections, we shall discuss different methods for evidence propagation. In Section 5.6.3, we shall discuss the method presented by J.H. Kim and J. Pearl. In this method, computing the updated probability function after a piece of evidence has become available essentially entails each statistical variable (that is, each vertex in the graphical part of the Bayesian network) updating the probability function locally from messages it receives from its neighbours in the graph, that is, from its predecessors as well as its successors, and then in turn sending new, updated messages to them. S.L. Lauritzen and D.J. Spiegelhalter have presented another, elegant method for evidence propagation. They have observed that calculating the updated probability function after a piece of evidence has become available will generally entail going against the initially assessed 'directed' conditional probabilities. They concluded that the directed graphical representation of a Bayesian network is not suitable as an architecture for propagating evidence directly. This observation, amongst other ones, motivated an initial transformation of the Bayesian network into an undirected graphical and probabilistic representation of the problem domain. We shall see in Section 5.6.4 where this method will be discussed in some detail, that this new representation allows for an efficient method for evidence propagation in which the computations to be performed are local to small sets of variables.

5.6.3 The reasoning method of Kim and Pearl

One of the earliest methods for reasoning with a Bayesian network was proposed by J.H. Kim and J. Pearl. Their method is defined for a restricted type of Bayesian network only. It therefore is not as general as the method of Lauritzen and Spiegelhalter which will be discussed in the following section.

The method of Kim and Pearl is applicable to Bayesian networks in which the graphical part is a so-called causal polytree. A *causal polytree* is an acyclic directed graph in which between any two vertices at most one path exists. Figure 5.9 shows such a causal polytree; note that the graph shown in figure 5.8 is not a causal polytree since there exist two different paths from the vertex V_3 to the vertex V_7. For evidence propagation in their restricted type of Bayesian network, Kim and Pearl exploit the mentioned topological property of a causal polytree. Observe that from this property we have that by deleting an arbitrary arc from a causal polytree, it falls apart into two separate components. In a causal polytree G, therefore, we can identify for a vertex V_i with m neighbours, m subgraphs of G each containing a neighbour of V_i such that after removal of V_i from G there does not exist a path from one such subgraph to another one. The subgraphs corresponding with the predecessors of the

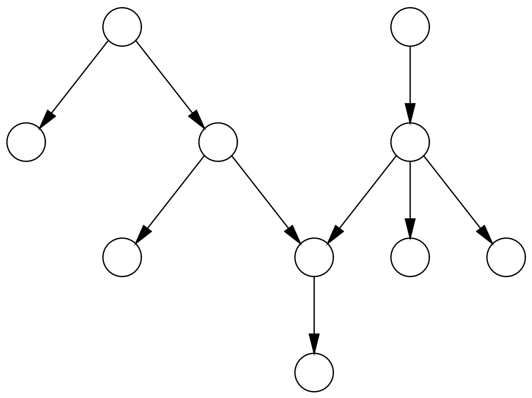


Figure 5.9: A causal polytree.

vertex will be called the *upper graphs* of V_i; the subgraphs corresponding with the successors of V_i will be called the *lower graphs* of V_i. The following example illustrates the idea. From now on, we shall restrict the discussion to this example; the reader may verify, however, that it can easily be extended to apply to more general causal polytrees.

EXAMPLE 5.15

Figure 5.10 shows a part of a causal polytree G. The vertex V_0 has the four neighbours V_1, V_2, V_3, and V_4. V_0 has two predecessors and therefore two upper graphs, which are denoted by G_1 and G_2, respectively; V_0 has also two lower graphs, denoted by G_3 and G_4. Note that there do not exist any paths between these subgraphs G_1, G_2, G_3, and G_4 other than through V_0.

So far, we have only looked at the graphical part of a Bayesian network. Recall that associated with the causal polytree we have a quantitative representation of the problem domain concerned: for each vertex, a set of local probabilities has been specified.

Let us suppose that evidence has become available that one of the statistical variables in the problem domain has adopted a specific value. This piece of evidence has to be entered into the Bayesian network in some way, and subsequently its effect on all other variables has to be computed to arrive at the updated probability function. The method for propagating evidence associated with this type of Bayesian network will be discussed shortly. First, however, we consider how probabilities of interest may be computed from the network. In doing so, we use an object-oriented style of discussion and view the causal polytree of the Bayesian network as a *computational architecture*. The vertices of the polytree are viewed as *autonomous objects* which hold some *private data* and are able to perform some computations. Recall that with each vertex is associated a set of local probabilities; these probabilities constitute the private data the object holds. The arcs of the causal polytree are taken as *communication channels*: the vertices are only able to communicate with their direct neighbours.

Now suppose that we are interested in the probabilities of the values of the variable V_0 after some evidence has been processed. It will be evident that, in terms of the graphical part of the Bayesian network, these probabilities cannot be computed from the private data the vertex holds; they are dependent upon the information from its upper and lower graphs

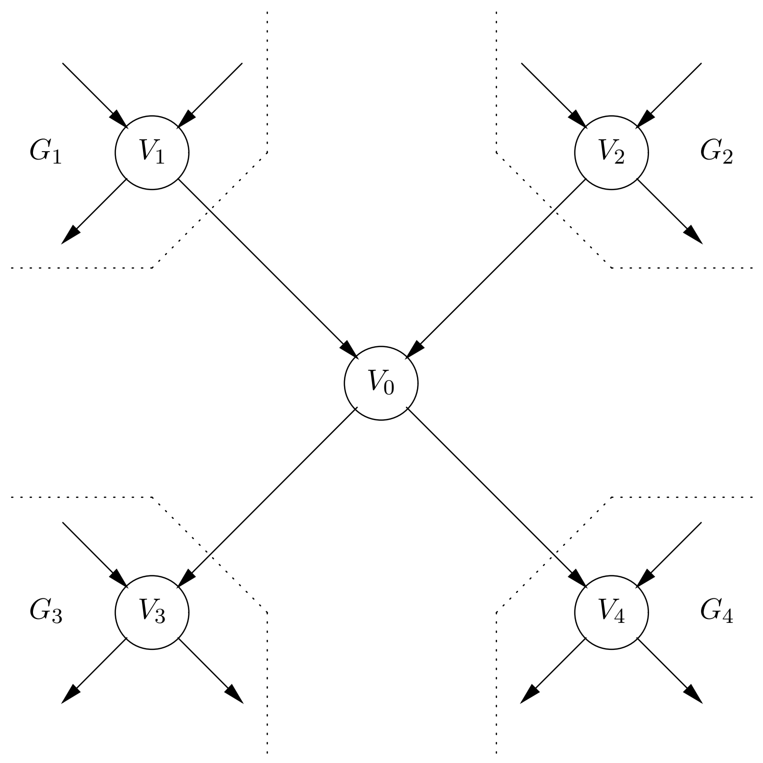


Figure 5.10: A part of a causal polytree.

as well. We shall see, however, that the neighbours of V_0 are able to provide V_0 with all information necessary for computing the probabilities of its values locally.

We introduce one more notational convention. After several pieces of evidence have been entered into the network and processed, some of the statistical variables have been *instantiated* with a value and some have not. Now, consider the configuration template $C_V(G) = V_1 \wedge \cdots \wedge V_n$ of the vertex set $V(G) = \{V_1, \ldots, V_n\}$, $n \geq 1$, in such a situation: we have that in the template some variables have been filled in. We shall use the notation $\tilde{c}_V(G)$ to denote the instantiated part of the template. If, for example, we have the configuration template $C = V_1 \wedge V_2 \wedge V_3$ and we know that the variable V_2 has adopted the value *true* and that the variable V_3 has the value *false*, and we do not know as yet the value of V_1, then $\tilde{c} = v_2 \wedge \neg v_3$.

We return to our example.

EXAMPLE 5.16

Consider the causal polytree from Figure 5.10 once more. We are interested in the probabilities of the values of the variable V_0. It can easily be proven, using Bayes' theorem and the independency relationships shown in the polytree, that these probabilities may be computed according to the following formula:

$$\begin{aligned} P(V_0 \mid \tilde{c}_V(G)) \quad = \quad & \alpha \cdot P(\tilde{c}_{V(G_3)} \mid V_0) \cdot P(\tilde{c}_{V(G_4)} \mid V_0) \\ & \cdot \big[P(V_0 \mid v_1 \wedge v_2) \cdot P(v_1 \mid \tilde{c}_{V(G_1)}) \cdot P(v_2 \mid \tilde{c}_{V(G_2)}) \\ & + P(V_0 \mid \neg v_1 \wedge v_2) \cdot P(\neg v_1 \mid \tilde{c}_{V(G_1)}) \cdot P(v_2 \mid \tilde{c}_{V(G_2)}) \end{aligned}$$

$$+P(V_0 \mid v_1 \wedge \neg v_2) \cdot P(v_1 \mid \tilde{c}_{V(G_1)}) \cdot P(\neg v_2 \mid \tilde{c}_{V(G_2)}) +$$
$$P(V_0 \mid \neg v_1 \wedge \neg v_2) \cdot P(\neg v_1 \mid \tilde{c}_{V(G_1)}) \cdot P(\neg v_2 \mid \tilde{c}_{V(G_2)}))\big]$$

where α is normalization factor chosen so as to guarantee $P(v_0 \mid \tilde{c}_V(G)) = 1 - P(\neg v_0 \mid \tilde{c}_V(G))$. We take a closer look at this formula. Note that the probabilities $P(v_0 \mid v_1 \wedge v_2)$, $P(v_0 \mid \neg v_1 \wedge v_2)$, $P(v_0 \mid v_1 \wedge \neg v_2)$, and $P(v_0 \mid \neg v_1 \wedge \neg v_2)$ necessary for computing the updated probabilities of the values of V_0 have been associated with V_0 initially: V_0 holds these probabilities as private data. So, if V_0 were to obtain the probabilities $P(\tilde{c}_{V(G_i)} \mid v_0)$ and $P(\tilde{c}_{V(G_i)} \mid \neg v_0)$ from its successors V_i, and the probabilities $Pr(v_j \mid \tilde{c}_{V(G_j)})$ and $Pr(\neg v_j \mid \tilde{c}_{V(G_j)})$ from each of its predecessors V_j, then V_0 would be able to locally compute the probabilities of its values.

In the previous example we have seen that the vertex V_0 has to receive some specific probabilities from its successors and predecessors before it is able to compute locally the probabilities of its own values. The vertex V_0 has to receive from each of its successors a so-called *diagnostic evidence parameter*: the diagnostic evidence parameter that the successor V_i sends to V_0 is a function λ_{V_i} defined by $\lambda_{V_i}(v_0) = P(\tilde{c}_{V(G_i)} \mid v_0)$ and $\lambda_{V_i}(\neg v_0) = P(\tilde{c}_{V(G_i)} \mid \neg v_0)$. The vertex V_0 furthermore has to receive from each of its predecessors a *causal evidence parameter*: the causal evidence parameter that the predecessor V_j sends to V_0 is a function π_{V_0} defined by $\pi_{V_0}(v_j) = P(v_j \mid \tilde{c}_{V(G_j)})$ and $\pi_{V_0}(\neg v_j) = P(\neg v_j \mid \tilde{c}_{V(G_j)})$. These evidence parameters may be viewed as being associated with the arcs of the causal polytree; Figure 5.11 shows the parameters associated with the causal polytree from Figure 5.10. Note that the π and λ parameters may be viewed as *messages* sent between objects.

Until now we have not addressed the question how a vertex computes the evidence parameters to be sent to its neighbours. We therefore turn our attention to evidence propagation. Suppose that evidence becomes available that a certain variable $V_i \in V(G)$ has adopted a certain value, say *true*. Informally speaking, the following happens. This evidence forces that variable V_i to update his private data: it will be evident that the updated probabilities for the values of V_i are $P(v_i) = 1$ and $P(\neg v_i) = 0$, respectively. From its local knowledge about the updated probability function, V_i then computes the proper π and λ parameters to be sent to its neighbours. V_i's neighbours subsequently are forced to update their local knowledge about the probability function and to send new parameters to their neighbours in turn. This way evidence, once entered, is spread through the Bayesian network.

EXAMPLE 5.17

Consider the causal polytree from Example 5.11 once more. The vertex V_0 computes the following causal evidence parameter to be sent to its successor V_3:

$$\begin{aligned}\pi_{V_3}(V_0) \;=\; & \alpha \cdot \lambda_{V_4}(V_0) \cdot [P(V_0 \mid v_1 \wedge v_2) \cdot \pi_{V_0}(v_1) \cdot \pi_{V_0}(v_2) \\ & +P(V_0 \mid \neg v_1 \wedge v_2) \cdot \pi_{V_0}(\neg v_1) \cdot \pi_{V_0}(v_2) \\ & +P(V_0 \mid v_1 \wedge \neg v_2) \cdot \pi_{V_0}(v_1) \cdot \pi_{V_0}(\neg v_2) \\ & +P(V_0 \mid \neg v_1 \wedge \neg v_2) \cdot \pi_{V_0}(\neg v_1) \cdot \pi_{V_0}(\neg v_2)]\end{aligned}$$

where α again is a normalization factor. In computing this causal evidence parameter, V_0 uses its private data and the information it obtains from its neighbours V_1, V_2, and

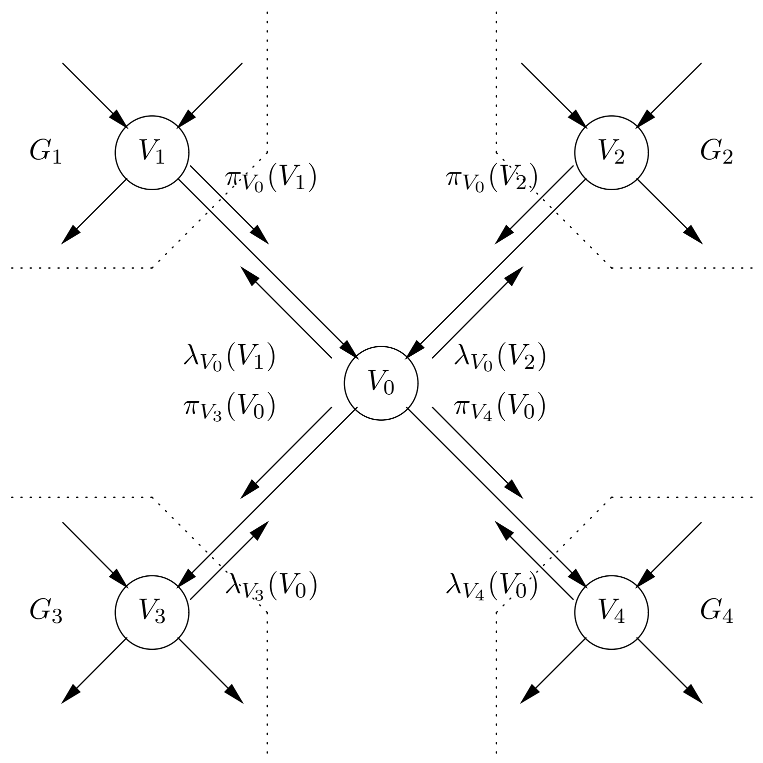


Figure 5.11: The π and λ parameters associated with the causal polytree.

V_4. Note that, if due to some new evidence for example the information $\lambda_{V_4}(V_0)$ has changed, then this change is propagated from V_4 through V_0 to V_3.

The vertex V_0 furthermore computes the following diagnostic evidence parameter to be sent to its predecessor V_1:

$$\begin{aligned}\lambda_{V_0}(V_1) \;=\; & \alpha \cdot \lambda_{V_3}(v_0) \cdot \lambda_{V_4}(v_0) \cdot [P(v_0 \mid V_1 \wedge v_2) \cdot \pi_{V_0}(v_2) \\ & + P(v_0 \mid V_1 \wedge \neg v_2) \cdot \pi_{V_0}(\neg v_2)] \\ & + \alpha \cdot \lambda_{V_3}(\neg v_0) \cdot \lambda_{V_4}(\neg v_0) \cdot [P(\neg v_0 \mid V_1 \wedge v_2) \cdot \pi_{V_0}(v_2) \\ & + P(\neg v_0 \mid V_1 \wedge \neg v_2) \cdot \pi_{V_0}(\neg v_2)]\end{aligned}$$

where α once more is a normalization factor.

We add to this example that the vertices V_i having no predecessors send a causal evidence parameter defined by $\pi_{V_j}(V_i) = P(V_i)$ to their successors V_j; furthermore, the vertices V_i having no successors initially send a diagnostic evidence parameter defined by $\lambda_{V_i}(V_j) = 1$ to their successors V_j.

We now have discussed the way a piece of evidence, once entered, is propagated through the causal polytree. We observe that any change in the joint probability distribution in response to a new piece of evidence spreads through the polytree in a single pass. This statement can readily be verified by observing that any change in the causal evidence parameter π associated with a specific arc of the causal polytree does not affect the diagnostic evidence parameter

λ on the same arc (and vice versa), since in computing the diagnostic evidence parameter $\lambda_{V_k}(V_0)$ associated with the arc (V_0, V_k) the causal evidence parameter $\pi_{V_k}(V_0)$ associated with the same arc is not used. So, in a causal polytree a perturbation is absorbed without reflection at the 'boundary' vertices, that is, vertices with either one outgoing or one incoming arc.

It remains to be discussed how a piece of evidence may be entered into the network. This is done rather elegantly: if evidence has become available that the variable V_i has the value *true* (or *false*, alternatively), then a dummy successor W of V_i is temporarily added to the polytree sending a diagnostic parameter $\lambda_W(V_i)$ to V_i such that $\lambda_W(v_i) = 1$ and $\lambda_W(\neg v)_i = 0$ (or vice versa if the value $false$ has been observed).

5.6.4 The reasoning method of Lauritzen and Spiegelhalter

In the previous section we have seen that propagating a piece of evidence concerning a specific statistical variable to the other variables in the graphical part of a Bayesian network will generally involve going against the directions of the arcs. This observation, amongst other ones, motivated S.L. Lauritzen and D.J. Spiegelhalter to transform an initially assessed Bayesian network into an equivalent undirected graphical and probabilistic representation of the problem domain. Their scheme for evidence propagation is defined on this new representation. The scheme has been inspired to a large extent by the existing statistical theory of *graphical models* (probabilistic models that can be represented by an undirected graph). In this theory, the class of so-called decomposable graphs has proven to be an important subclass of graphs. Before we define the notion of a decomposable graph, we introduce several other notions.

Definition 5.22 *Let $G = (V(G), E(G))$ be an undirected graph where $E(G)$ is a finite set of unordered pairs (V_i, V_j), $V_i, V_j \in V(G)$, called edges. A* cycle *is a path of length at least one from V_0 to V_0, $V_0 \in V(G)$. A cycle is* elementary *if all its vertices are distinct. A* chord *of an elementary cycle $V_0, V_1, \ldots, V_k = V_0$ is an edge (V_i, V_j), $i \neq (j \pm \mathrm{mod}(k+1)$.*

We now are ready to define the notion of a decomposable graph.

Definition 5.23 *An undirected graph is* decomposable *if all elementary cycles of length $k >= 4$ have a chord.*

It can be shown that a probability function on such a graph may be expressed in terms of local probability functions, called *marginal* probability functions, on small sets of variables. We shall see that a representation of the problem domain in a decomposable graph and an associated representation of the probability function then allows for an efficient scheme for evidence propagation, in which the computations to be performed are local to these small sets of variables.

In order to be able to fully exploit the theory of graphical models, Lauritzen and Spiegelhalter propose a transformation of the initially assessed Bayesian network in which the graphical representation of the Bayesian network is transformed into a decomposable graph, and in which from the probabilistic part of the network a new representation of the probability function in terms of the resulting decomposable graph is obtained. The resulting representation of the problem domain is a new type of Bayesian network, which will henceforth be called a *decomposable Bayesian network*. We shall only describe the transformation of the initially assessed Bayesian network into such a decomposable Bayesian network informally.

The transformation of the original acyclic directed graph G into a decomposable graph involves three steps:

(1) Add arcs to G in such a way that no vertex in $V(G)$ has non-adjacent predecessors.

(2) Subsequently, drop the directions of the arcs.

(3) Finally, cut each elementary cycle of length four or more short by adding a chord.

It will be evident that the resulting graph is decomposable. Note that the result obtained is not unique.

EXAMPLE 5.18

Consider the Bayesian network from the example of Section 5.6.1 once more. The transformation of the graphical part of this Bayesian network into a decomposable graph is demonstrated in Figure 5.12. We consider the transformation steps in further detail. First of all, we have to add new arcs to the graph such that no vertex has non-adjacent predecessors. Now observe that in figure 5.8 the vertex V_6 has two predecessors: the vertices V_2 and V_4. Since there does not exist an arc between V_2 and V_4, we have that the predecessors of V_6 are nonadjacent. We therefore add an arc between V_2 and V_4. Note that we also have to add an arc between the vertices V_5 and V_6. Since we will drop all directions in the second transformation step, the directions of the added arcs are irrelevant. From subsequently dropping the directions of the arcs, we obtain an undirected graph. The resulting graph, however, is still not decomposable, since it has an elementary cycle of length 4 without any shortcut: V_3, V_4, V_6, V_5, V_3. We cut this cycle short by adding an edge between the vertices V_4 and V_5. Note that addition of an edge between V_3 and V_6 would have yielded a decomposable graph as well.

We now have obtained an undirected graphical representation of the problem domain. With this undirected graph, an 'undirected' representation of the probability function is associated. We confine ourselves to a discussion of this new representation, without describing how it is actually obtained from the initially assessed probabilities. It should however be evident that the new representation can be obtained from the original one, since the initial probabilities define a unique probability function.

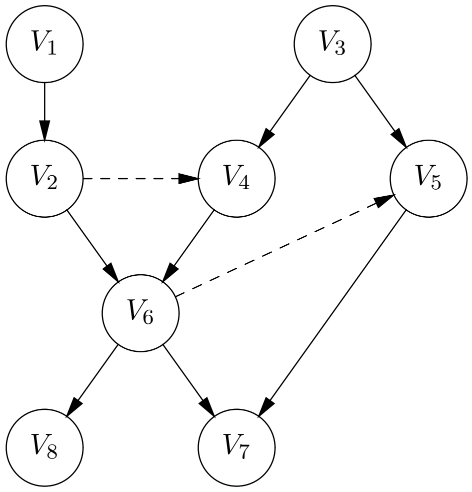
We shall see that the probability function can be expressed in terms of marginal probability functions on the cliques of the decomposable graph. We define the notion of a clique.

Definition 5.24 *Let $G = (V(G), E(G))$ be an undirected graph. A* clique *of G is a subgraph $H = (V(H), E(H))$ of G such that for any two distinct vertices $V_i, V_j \in V(H)$ we have that $(V_i, V_j) \in E(H)$. H is called a* maximal clique *of G if there does not exist a clique H' of G differing from H such that H is a subgraph of H'.*

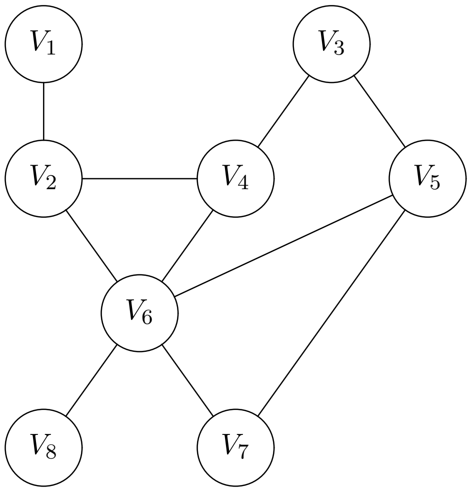
In the sequel, we shall take the word clique to mean a maximal clique.

EXAMPLE 5.19

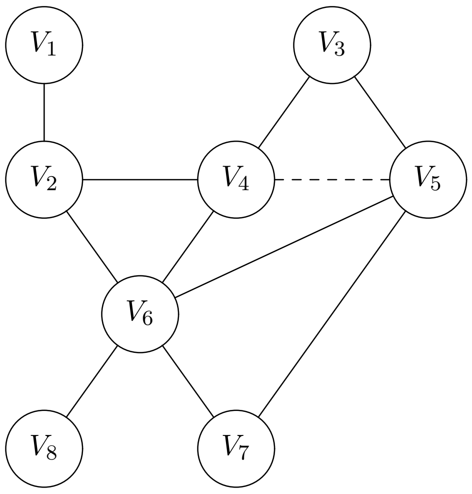
Consider the decomposable graph from Figure 5.12 once more. The reader can easily verify that this graph contains six cliques.



(a) Add arcs such that no vertex has non-adjacent predecessors



(b) Drop the directions of the arcs



(c) Cut elementary cycles short

Figure 5.12: Construction of the decomposable graph.

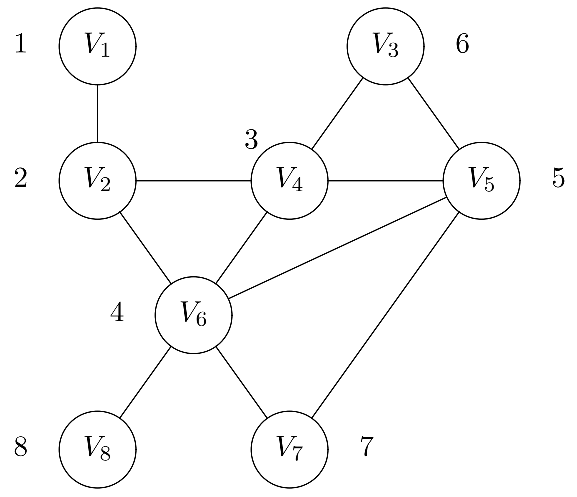


Figure 5.13: An ordering of the vertices obtained from maximum cardinality search.

To arrive at the new representation of the probability function, we obtain an ordering of the vertices and of the cliques of the decomposable graph. Its vertices are ordered as follows:

(1) Assign an arbitrary vertex the number 1.

(2) Subsequently, number the remaining vertices in increasing order such that the next number is assigned to the vertex having the largest set of previously numbered neighbours.

We say that the ordering has been obtained from *maximum cardinality search*. After the vertices of the decomposable graph have been ordered, the cliques of the graph are numbered in the order of their highest numbered vertex.

EXAMPLE 5.20

Consider the decomposable graph $G = (V(G), E(G))$ as shown in Figure 5.12 once more. The vertices of G are ordered using maximum cardinality search. An example of such an ordering is shown in Figure 5.13. The six cliques of the graph subsequently are numbered in the order of their highest numbered vertex. Let Cl_i be the clique assigned number $i, i = 1, \ldots, 6$. Then, we have obtained the following ordering (for ease of exposition we identify a clique with its vertex set):

$$\begin{aligned}
\mathrm{Cl}_1 &= \{V_1, V_2\} \\
\mathrm{Cl}_2 &= \{V_2, V_4, V_6\} \\
\mathrm{Cl}_3 &= \{V_4, V_5, V_6\} \\
\mathrm{Cl}_4 &= \{V_3, V_4, V_5\} \\
\mathrm{Cl}_5 &= \{V_5, V_6, V_7\} \\
\mathrm{Cl}_6 &= \{V_6, V_8\}
\end{aligned}$$

We consider the ordering $\mathrm{Cl}_1, \ldots, \mathrm{Cl}_m$, $m \geq 1$, of the cliques of a decomposable graph G in further detail. Let $V(\mathrm{Cl}_i)$ denote the vertex set of clique Cl_i, $i = 1, ..., m$. The ordering now has the following important property: for all $i \geq 2$ there is a $j < i$ such that $V(\mathrm{Cl}_i) \cap (V(\mathrm{Cl}_1) \cup \cdots \cup V(\mathrm{Cl}_{i-1})) \subset V(\mathrm{Cl}_j)$. In other words, the vertices a clique has in common with the lower numbered cliques are all contained in one such clique. This property is known as the *running intersection property.* This property now enables us to write the probability function on the decomposable graph as the product of the marginal probability functions on its cliques, divided by a product of the marginal probability functions on the clique intersections:

$$P(C_{V(G)}) = \prod_{i=1}^{m} \frac{P(C_{V(\mathrm{Cl}_i)})}{P(C_{S_i})}$$

where S_i is the set of vertices Cl_i has in common with the lower numbered cliques.

EXAMPLE 5.21

Consider the decomposable graph G shown in Figure 5.13 once more. The probability function on G may be expressed as

$$\begin{aligned} P(V_1 \wedge \cdots \wedge V_8) \quad = \quad & P(V_1 \wedge V_2) \cdot \frac{P(V_2 \wedge V_4 \wedge V_6)}{P(V_2)} \cdot \frac{P(V_4 \wedge V_5 \wedge V_6)}{P(V_4 \wedge V_6)} \\ & \cdot \frac{P(V_3 \wedge V_4 \wedge V_5)}{P(V_4 \wedge V_5)} \cdot \frac{P(V_5 \wedge V_6 \wedge V_7)}{P(V_5 \wedge V_6)} \cdot \frac{P(V_6 \wedge V_8)}{P(V_6)} \end{aligned}$$

The initially assessed Bayesian network has now been transformed into a decomposable Bayesian network. The scheme for evidence propagation proposed by Spiegelhalter and Lauritzen operates on this decomposable Bayesian network. We emphasize that for a specific problem domain the transformation has to be performed only once: each consultation of the system proceeds from the obtained decomposable Bayesian network.

Recall that for making probabilistic statements concerning the statistical variables discerned in a problem domain we have to associate with a decomposable Bayesian network a method for computing probabilities of interest from it and a method for propagating evidence through it. As far as computing probabilities from a decomposable Bayesian network is concerned, it will be evident that any probability which involves only variables occurring in one and the same clique can simply be computed locally from the marginal probability function on that clique.

The method for evidence propagation is less straightforward. Suppose that evidence becomes available that the statistical variable V has adopted a certain value, say v. For ease of exposition, we assume that the variable V occurs in one clique of the decomposable graph only. Informally speaking, propagation of this evidence amounts to the following. The vertices and the cliques of the decomposable graph are ordered anew, this time starting with the instantiated vertex. The ordering of the cliques then is taken as the order in which the evidence is propagated through the cliques. For each subsequent clique, the updated marginal probability function is computed locally using the computation scheme shown below; we use

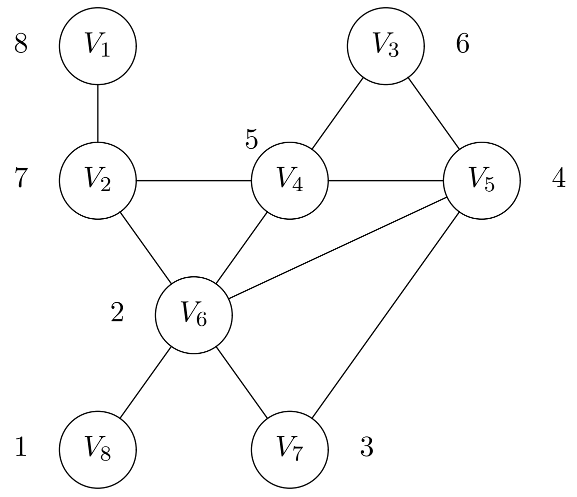


Figure 5.14: An ordering of the vertices starting with V_8.

P to denote the initially given probability function and P^* to denote the new probability function after updating. For the first clique in the ordering we simply compute:

$$P^*(C_{V(\mathrm{Cl}_1)}) = P(C_{V(\mathrm{Cl}_1)} \mid v)$$

For the remaining cliques, we compute the updated marginal probability function using:

$$\begin{aligned} P^*(C_{V(\mathrm{Cl}_i)}) &= P(C_{V(\mathrm{Cl}_i)} \mid v) \\ &= P(C_{V(\mathrm{Cl}_i)\setminus S_i} \mid C_{S_i} \wedge v) \cdot P(C_{S_i} \mid v) \\ &= P(C_{V(\mathrm{Cl}_i)\setminus S_i} \mid C_{S_i}) \cdot P^*(C_{S_i}) \\ &= P(C_{V(\mathrm{Cl}_i)}) \cdot \frac{P^*(C_{S_i})}{P(C_{S_i})} \end{aligned}$$

where S_i once more is the set of vertices Cl_i has in common with the lower numbered cliques. So, an updated marginal probability function is obtained by multiplying the 'old' marginal probability function with the quotient of the 'new' and the 'old' marginal probability function on the appropriate clique-intersection.

We look once more at our example.

EXAMPLE 5.22

Consider the decomposable graph from Figure 5.12 and its associated probability function once more. Suppose that we obtain the evidence that the variable V_8 has the value *true*. Using maximum cardinality search, we renumber the vertices of the graph starting with the vertex V_8. Figure 5.14 shows an example of such an ordering. From this new ordering of the vertices we obtain an ordering of the six cliques of the graph (once more, we identify a clique with its vertex set):

$$\begin{aligned} \mathrm{Cl}_1 &= \{V_6, V_8\} \\ \mathrm{Cl}_2 &= \{V_5, V_6, V_7\} \\ \mathrm{Cl}_3 &= \{V_4, V_5, V_6\} \\ \mathrm{Cl}_4 &= \{V_3, V_4, V_5\} \end{aligned}$$

$$\begin{aligned}\mathrm{Cl}_5 &= \{V_2, V_4, V_6\}\\ \mathrm{Cl}_6 &= \{V_1, V_2\}\end{aligned}$$

The impact of the evidence on the first clique is

$$P^*(V_6) = P(V_6 \mid v_8)$$

For the second clique we find:

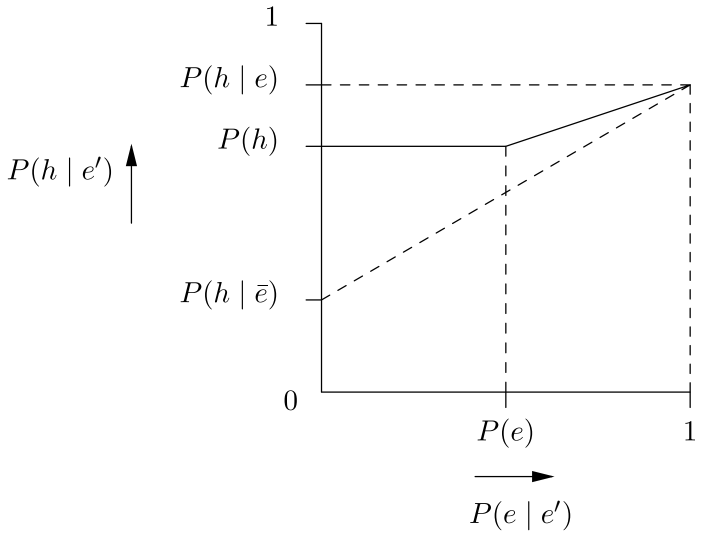
$$P^*(V_5 \wedge V_6 \wedge V_7) = P(V_5 \wedge V_6 \wedge V_7) \cdot \frac{P^*(V_6)}{P(V_6)}$$

For the remaining cliques we obtain similar results.

After the marginal probability functions have been updated locally, the instantiated vertex is removed from the graph, and the updated marginal probability functions are taken as the marginal probability functions on the cliques of the remaining graph. The process may now simply be repeated for a new piece of evidence.

Exercises

(5.1) The subjective Bayesian method uses a linear interpolation function as a combination function for propagating uncertain evidence. Recall that this interpolation function consists of two distinct linear functions, each defined on half of the domain of the combination function. Instead of the function employed in PROSPECTOR as discussed in Section 5.3.2, we could use for example the function shown in the figure below.



Describe the effect of applying the production rule **if** e **then** h **fi** on the prior probability of h in case this function is used as the combination function for uncertain evidence.

(5.2) Prove by means of counterexamples that the combination functions for composite evidence in the subjective Bayesian method are not correct when viewed from the perspective of probability theory.

(5.3) Write a PROLOG or LISP program implementing the subjective Bayesian method. You can depart from the program for top-down inference discussed in Chapter 3 (of Principles of Intelligent Systems).

(5.4) A particular rule-based system employs the certainty factor model for modelling the uncertainty that goes with the problem domain of concern. Let the following three production rules be given (only the names of the attributes in the conditions and conclusions are shown):

> **if** b **or** c **then** $f_{0.3}$ **fi**
> **if** f **and** g **then** $a_{0.8}$ **fi**
> **if** d **or** e **then** $a_{0.2}$ **fi**

Furthermore, suppose that the attributes b, c, d, e, and g have been established with the certainty factors 0.2, 0.5, 0.3, 0.6, and 0.7, respectively. The attribute a is the goal attribute of top-down inference. Give the inference network resulting from top-down inference with these facts and production rules. Compute the certainty factor which results for the attribute a.

(5.5) Consider the following frame of discernment: $\Theta = \{a, b, c\}$. Let the basic probability assignment m be defined by $m(\{a\}) = 0.3$, $m(\{a, b\}) = 0.4$, $m(\{a, b, c\}) = 0.2$, $m(\{a, c\}) = 0.1$; the remaining basic probability numbers all equal 0. Compute $\text{Bel}(\{a, c\})$.

(5.6) Let Θ be a frame of discernment. Prove that for each $x \subseteq \Theta$ we have that $\text{Pl}(x) \geq \text{Bel}(x)$.

(5.7) Let $\Theta = \{a, b, c, d\}$ be a frame of discernment. Give an example of a basic probability assignment on Θ that defines a probability function on Θ at the same time.

(5.8) Consider the frame of discernment $\Theta = \{a, b, c\}$ and the following two basic probability assignments m_1 and en m_2:

$$m_1(x) = \begin{cases} 0.3 & \text{if } x = \Theta \\ 0.6 & \text{if } x = \{a, c\} \\ 0.1 & \text{if } x = \{b, c\} \\ 0 & \text{otherwise} \end{cases}$$

$$m_2(x) = \begin{cases} 0.8 & \text{if } x = \Theta \\ 0.2 & \text{if } x = \{b\} \\ 0 & \text{otherwise} \end{cases}$$

Construct the intersection tableau for the function $m_1 \oplus m_2$ using Dempster's rule of combination.

9. Consider the frame of discernment $\Theta = \{a, b, c\}$ and the following basic probability assignments m_1 and m_2:

$$m_1(x) = \begin{cases} 0.3 & \text{if } x = \Theta \\ 0.6 & \text{if } x = \{a, c\} \\ 0.1 & \text{if } x = \{a, b\} \\ 0 & \text{otherwise} \end{cases}$$

$$m_2(x) = \begin{cases} 0.8 & \text{if } x = \Theta \\ 0.2 & \text{if } x = \{a\} \\ 0 & \text{otherwise} \end{cases}$$

Why is it not necessary in this case to normalize? Compute the value of $\mathrm{Bel}_1 \oplus \mathrm{Bel}_2(\{a\})$.

(5.10) Consider the following medical information:

> Metastatic cancer is a possible cause of a brain tumor, and is also an explanation for increased total serum calcium. In turn, either of these could explain a patient falling into a coma. Severe headache is also possibly associated with a brain tumour.

Suppose that we use a Bayesian network to represent this information. Give the graphical part of the Bayesian network. Which probabilities have been associated with the graph?

(5.11) Consider the causal polytree from Figure 5.9 and an associated set of probabilities. Suppose that we apply the method of J.H. Kim and J. Pearl for evidence propagation. Try to find out how evidence spreads through the network if entered in one of the vertices.

(5.12) Consider the Bayesian network obtained in Exercise 10 once more. We transform this Bayesian network into a decomposable Bayesian network as described in Section 5.6.4.

(a) Give the resulting decomposable graph. Which cliques do you discern?

(b) Give the new representation of the originally given probability function.

(c) What happens if we obtain the evidence that a specific patient is suffering from severe headaches?

Chapter 6

Model-based Reasoning

Diagnosis is commonly viewed as the interpretation of case-specific findings in the context of knowledge from a problem domain to obtain an indication of the presence and absence of defects or faults, and also of the nature of the problem. Computer-aided diagnosis was among the first applications investigated when digital computers became available more than four decades ago. It still remains an important research area, in which several new developments have taken place in the last decade. Diagnosis is the subject of this Chapter, where in particular we focus on *model-based* approaches.

It is customary to distinguish between diagnostic systems based on symbolic, or qualitative, reasoning technology, and those based on probability theory and statistics, although some systems offer a mixture of the two approaches. In this paper, we focus on systems based on symbolic reasoning technology.

Until recently, however, no theoretical framework was available to formally describe and compare the various underlying principles. At a conceptual level, it was evident that the knowledge bases of some of the systems captured models of structure and behaviour in a domain. Such systems have been called *model-based* or '*first principles*' systems. The knowledge bases of other systems, however, did not embody an explicit model of structure and behaviour, but rather consisted of encoded human expertise in solving particular problems in the underlying domain. Currently, the term *empirical associations* is often employed to denote such knowledge. The classical example of such a system is MYCIN.

The model-based approach to diagnosis has been successfully applied to fault finding in electronic circuits; in particular Johan de Kleer has done a considerable amount of work in this area. The study of simple electronic circuits has yielded much insight into the nature of the diagnostic process. More importantly, one of the first formal theories of diagnosis emerged from this research: the theory of consistency-based diagnosis as proposed by Ray Reiter. *Consistency-based diagnosis* offers a logic-based framework to formally describe diagnosis of abnormal behaviour in a device or system, using a model of normal structure and functional behaviour. Basically, consistency-based diagnosis amounts to finding faulty device components that account for a discrepancy between predicted normal device behaviour and observed (abnormal) behaviour. The predicted behaviour is inferred from a formal model of normal structure and behaviour of the device.

Where consistency-based diagnosis traditionally employs a model of *normal* behaviour, *abduction* has been the principal model-based technique for describing and analysing diagnosis using a model of *abnormal* behaviour in terms of cause-effect relationships. Early work on abduction has been done by Harry Pople and David Poole. In *abductive diagnosis*, diagnostic problem solving consists of establishing a diagnosis using cause-effect relationships with a set of observed findings (effects) as the starting point. In abduction, a system reasons from effects to causes, instead of from causes to effects. Because the reasoning from causes to effects can be accomplished using logical deduction, in a sense abductive reasoning is carried out in a direction reverse to that of deduction.

Logical deduction, however, also has its place in the picture, because it has been used to formalise reasoning with the logical analogues of empirical associations. In the context of diagnosis, reasoning with empirical associations is often referred to as *heuristic classification*.

Although much work has now been done to formalise diagnosis, it has been difficult to capture the concept of diagnosis in a precise, formal and also general way, leaving room for various types of diagnosis. Both consistency-based diagnosis and abductive diagnosis have been looked upon as core concepts for formal frameworks of diagnosis, but, as we shall see, other formalisations are also possible. A formal framework of diagnosis offers means to formally describe and analyse various notions of diagnosis. The frameworks described in the literature are either logic-based or based on set theory.

The formalisation of diagnosis is the subject reviewed in this chapter. The structure of this chapter is as follows. First, the nature of the diagnostic process is sketched. Next, the various core approaches to diagnosis described in the literature are reviewed. Finally, the various approaches to diagnosis are compared to each other, and a number of frameworks that offer means for the general description of diagnosis are discussed.

6.1 Diagnostic problem solving

Discovering what is wrong in a particular situation is one of the central activities in real life; this process is usually called *diagnosis* or *diagnostic problem solving*. The process may be viewed as the selective gathering and interpretation of information as evidence for or against the presence or absence of one or more defects in a system. This informal definition reveals that the following aspects are of central importance to diagnostic problem solving. Firstly, the *gathering* of information, and secondly, the *interpretation* of the gathered information for determining what is wrong, for example with a patient or a device. In medicine, defects are disorders of a patient; in technical domains, defects are faults of a device. In medicine, the information-gathering process is usually carried out in a systematic, structured fashion, because there are an enormous number of diagnostic tests available to the clinician, that cannot all be carried out. Furthermore, some diagnostic tests cause discomfort to the patient, or carry even some risk of causing disease or death. By restricting the selection of diagnostic tests in early diagnosis to those that do no harm or cause little discomfort to the patient, as is common practice in medical diagnosis, diagnostic tests are performed only when necessary. In technical fields, it is sometimes impossible to gather certain information because of time constraints, costs involved, or physical impossibility. Although the information-gathering process is a characteristic feature of diagnosis, the interpretation of information as evidence for or against a diagnostic solution is a more fundamental aspect of diagnostic problem solving.

The information-gathering process together with related aspects, such as the process of generating, and accepting or rejecting diagnostic hypotheses are sometimes referred to as the *dynamic* aspects of diagnostic problem solving. They yield specific problem-solving behaviour. Establishing an actual diagnostic solution requires knowledge of what constitutes a

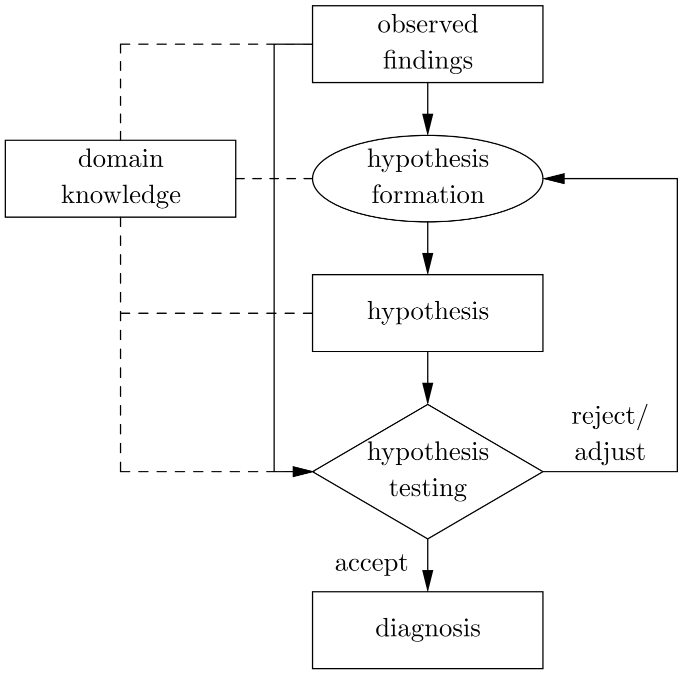


Figure 6.1: Diagnostic problem solving and the empirical cycle.

diagnosis of a particular problem; the various aspects involved are sometimes referred to as the *static* aspects of diagnostic problem solving. This chapter focuses on these static aspects of diagnostic problem solving.

In general, diagnostic problem solving, like many other forms of problem solving, may be described using the scientific notion of the *empirical cycle*, which describes the framework underlying empirical research. It states that empirical research encompasses: (1) formulating a *hypothesis*, (2) *testing* that hypothesis, and (3) *rejecting* the hypothesis when it fails to pass the tests, or *accepting* the hypothesis when it successfully passes the tests.[1] The process may start again with (1), in which case the formulation of a new hypothesis possibly involves *adjusting* a hypothesis previously rejected. In Figure 6.1, this view of diagnostic problem solving as an instance of the empirical cycle is depicted. Testing involves the application of procedures for the verification and falsification of a hypothesis using *observed findings* and domain knowledge. In general, a hypothesis may be a complex structure or mechanism. In diagnostic problem solving, however, a hypothesis is usually taken to be a collection of 'defects', where each defect is assumed to be either present or absent. This simplification may not always be justified, for example because the defects may be interrelated to each other in some particular way, which could be part of the hypothesis. For example, a hypothesis may be whether or not a process A is causally related to a process B. Nevertheless, this simplification is invariably made in diagnostic systems, and seems acceptable in the light of developed applications. A *diagnosis* may be conceived as an accepted hypothesis concerning a particular defect or collection of defects; the results of diagnostic tests correspond to the observed findings.

The literature on diagnosis more or less follows the terminology and structure of the empirical cycle. For example, views diagnostic problem solving as three fundamental subproblems:

[1]Popperians may read instead: 'not rejecting the hypothesis so long as it has not been falsified by a test'.

(1) Hypothesis generation (or hypothesis formation);

(2) Hypothesis testing;

(3) Hypothesis discrimination.

The subproblem of hypothesis discrimination concerns selecting from the hypotheses accepted on the basis of a measure of plausibility. This process may entail collecting additional observed findings.

The basic framework of diagnostic problem solving as the empirical cycle can be refined in several ways. For example, there may be an ordering on the set of hypotheses, such as an ordering from generic to specific, or an ordering by the value of a real-valued utility function associated with the hypotheses. A class of defects may be taken as a generic hypothesis, and a specific defect may be viewed as a specific hypothesis. Such orderings are especially useful in guiding the problem-solving process, information gathering included. For example, the process may be decomposed into several stages working from generic towards more specific hypotheses, or from hypotheses with high associated utility to those with low associated utility. It is well-known that guiding the problem-solving process, using information collected at earlier stages, may be quite effective in reducing the number of tests to be performed. It may also result in a step-wise reduction in the number of defects to be considered, due to the rejection of specific hypotheses motivated by the earlier rejection of more generic hypotheses. This approach to handling hypotheses and observable findings is an example of a so-called (diagnostic) *problem-solving strategy*. Problem-solving strategies are beyond the scope of this book, because they belong to the dynamics of diagnosis.

6.2 Conceptual basis of diagnosis

Although the description of diagnostic problem solving given in Section 6.1 carries much of the flavour of the process of diagnosis, it is still an imprecise description and, in fact, several formal theories have been proposed to capture the concept of diagnosis more precisely. In doing so, however, researchers became aware that there are actually various *conceptual models* of diagnosis, determined by the kind of knowledge involved. As mentioned above, diagnosis concerns the interpretation of observed findings in the context of knowledge from a problem domain. A good starting point for describing diagnosis at a conceptual level are the various types of knowledge that play a role in diagnostic applications.

The knowledge embodied in a diagnostic system may be based on one or more of the following descriptions:

(1) A description of the *normal* structure and functional behaviour of a system.

(2) A description of *abnormal* functional behaviour of a system; abnormal structure is usually not taken into account.

(3) An enumeration of defects and collections of observable findings for every possible defect concerned, without the availability of explicit knowledge concerning the (abnormal) functional behaviour of the system.

(4) An enumeration of findings for the normal situation.

These types of knowledge may coexist in real-life diagnostic systems, but it is customary to emphasise their distinction in conceptual and formal theories of diagnosis. Similar classifications of types of knowledge appear in the literature on diagnosis, although often no clear distinction is made between the conceptual, formal and implementation aspects of diagnostic systems. For example, some researchers distinguish diagnostic rule-based systems, by which they mean diagnostic systems based on knowledge of the third type mentioned above, from diagnostic systems incorporating knowledge of structure and behaviour, i.e. knowledge of the first and second type mentioned above. However, rule-based systems with a sufficiently expressive production-rule formalism can be used to implement any diagnostic system, including those based on knowledge of structure and behaviour.

An observed finding that has been gathered in diagnosing a problem is often said to be either a 'normal finding', i.e. a finding that matches the normal situation, or an 'abnormal finding', i.e. a finding that does not match the normal situation. Based on the four types of knowledge mentioned above, and the two sorts of findings, three different conceptual models of diagnosis are usually distinguished; they will be called:

- *Deviation-from-Normal-Structure-and-Behaviour diagnosis*, abbreviated to *DNSB diagnosis*,
- *Matching-Abnormal-Behaviour diagnosis*, abbreviated to *MAB diagnosis*, and
- *Abnormality-Classification diagnosis*, abbreviated to *AC diagnosis.*

Below, we shall discuss the relationship between these three conceptual models of diagnosis and the four types of knowledge mentioned above. A formal theory of diagnosis has been proposed for each of these conceptual models of diagnosis. In the remainder of this section, each of the three conceptual models of diagnosis will be discussed, and the corresponding formal theory of diagnosis is mentioned. The formal theories of diagnosis are discussed in depth in Section 6.3.

DNSB diagnosis. For diagnosis based on knowledge concerning normal structure and behaviour, little or no explicit knowledge is available about the relationships between defects of the system, on the one hand, and findings to be observed when certain defects are present, on the other hand. Hence, DNSB diagnosis typically employs knowledge of the first and fourth types mentioned above. From a practical point of view, the primary motivation for investigating this approach to diagnosis is that in many domains little knowledge concerning abnormality is available, which is certainly true for new human-developed artifacts. For example, for a new device that has just been released from the factory, experience with respect to the faults that may occur when the device is in operation is lacking. Thus, the only conceivable way in which initially such faults can be handled is by looking at the normal structure and functional behaviour of the device. Yet, even if knowledge concerning abnormal behaviour is available, exhaustive description may be sometimes too cumbersome compared with a model of normal behaviour.

For the purpose of diagnosis, the actual behaviour of a physical device, called *observed behaviour*, is compared with the results of a model of normal structure and behaviour of the device, which may be taken as *predicted behaviour.* Both types of behaviour can be characterised by findings. If there is a *discrepancy* between the observed and the predicted behaviour, diagnostic problem solving amounts to isolating the components in the device that are not properly functioning, using a model of the normal structure and behaviour of

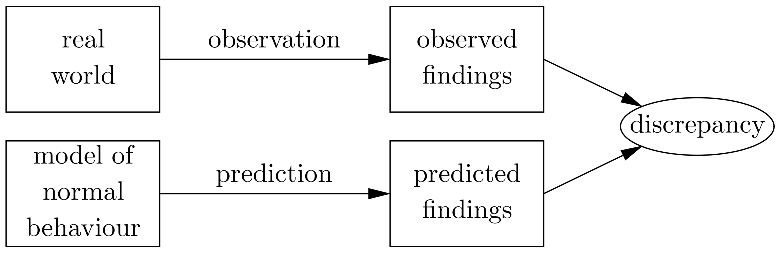


Figure 6.2: Deviation-from-normal-structure-and-behaviour (DNSB) diagnosis.

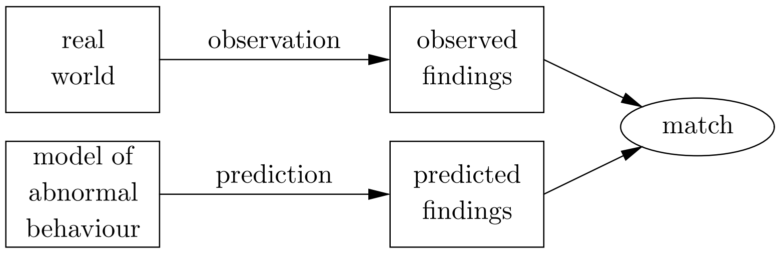


Figure 6.3: Matching-abnormal-behaviour (MAB) diagnosis.

the device. In doing so, it is assumed that the model of normal structure and behaviour is sufficiently accurate and correct. Figure 6.2 depicts DNSB diagnosis in a schematic way. DNSB diagnosis is frequently erroneously called model-based diagnosis in the literature, as if it were the only instance of model-based diagnosis. It is also called consistency-based dagnosis, but in this book this term is reserved for the corresponding formal theory of diagnosis. DNSB diagnosis has been developed in the context of troubleshooting in electronic circuits. A well-known program that supports DNSB diagnosis, and includes various strategies to do so efficiently, is the *General Diagnostic Engine* (GDE) as proposed by Johan de Kleer.

Above, we have reviewed the conceptual basis of diagnosis based on a model of normal structure and behaviour, which we have called DNSB diagnosis. The formal counterpart of DNSB diagnosis, called *consistency-based diagnosis*, originates from work by Ray Reiter; consistency-based diagnosis will be discussed in detail below. As far as known to the author, DNSB diagnosis-like approaches have been used in medical applications on a limited scale; there is more work in which DNSB diagnosis has been applied to solve technical problems.

MAB diagnosis. For diagnosis based on knowledge of abnormal behaviour, diagnostic problem solving amounts to simulating the abnormal behaviour using an explicit model of that behaviour. Hence, in MAB diagnosis the use of knowledge of abnormal behaviour (the second type mentioned above) is emphasised. By assuming the presence of certain defects, some observable abnormal findings can be predicted. It can be investigated which of these assumed defects account for the observed findings by *matching* the predicted abnormal findings with those observed. In Figure 6.3, MAB diagnosis is depicted schematically. In most applications of MAB diagnosis, the domain knowledge that is used for diagnosis consists of causal relationships. Two, strongly related, formal counterparts of MAB diagnosis have been proposed in the literature. The first formal theory, referred to as the *set-covering theory of diagnosis*, is based on set theory: causal knowledge is expressed as mathematical relations, used for diagnosis. This theory originates from work by James Reggia and others. The second theory is based on logic.

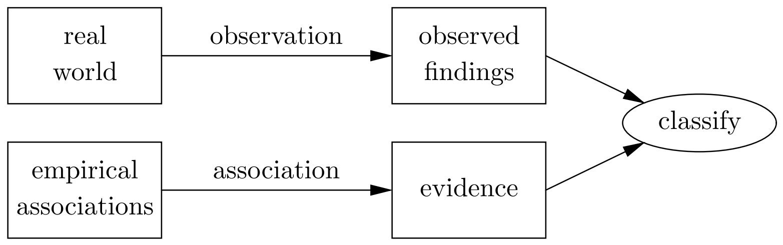


Figure 6.4: Abnormality-classification (AC) diagnosis.

Based on the type of reasoning employed to formalise MAB diagnosis, i.e. reasoning from effects to causes instead of from causes to effects, this theory of diagnosis is also referred to as *abductive diagnosis*. Theorist, developed by David Poole, and CHECK, developed by Luca Console, are two systems supporting MAB diagnosis.

AC diagnosis. Where DNSB and MAB diagnosis employ a model of normal or abnormal structure and behaviour for the purpose of diagnosis, the third conceptual model of diagnosis uses neither. The knowledge employed in this conceptual model of diagnosis consists of the enumeration of more or less typical evidence that can be observed, i.e. observable findings, when a particular defect or defect category is present (the third type of knowledge mentioned above). For example, sneezing is a finding that may be typically observed in a disorder like common cold. This form of knowledge has been referred to as *empirical associations* above (the phrase '*compiled knowledge*' is also employed). Diagnostic problem solving amounts to establishing which of the elements in a finite set of defects have associated findings that account for as many of the findings observed as possible, as is shown in Figure 6.4. The enumeration of findings for the normal situation (knowledge of the fourth type mentioned above) is sometimes also used in AC diagnosis, together with knowledge of the third type; then, observed findings are classified in terms of present and absent defects. The main goal of AC diagnosis, however, remains the classification of observed findings in terms of abnormality. AC diagnosis is often referred to in the literature as *heuristic classification*, although this term is broader, since it also includes a reasoning strategy. The MYCIN system is the classical system in which this conceptual approach to diagnosis has been adopted. AC diagnosis can be characterised in terms of logical deduction in a straightforward way. We shall refer to this formalisation of AC diagnosis as *hypothetico-deductive diagnosis*.

A comparison of the three conceptual models of diagnosis is given in Table 6.1. Obviously, the various models of diagnosis discussed above can also be combined. To solve real-life

	DNSB	**MAB**	**AC**
Type of knowledge	normal structure and behaviour	causal model of abnormality	empirical associations
Formalisation	consistency-based diagnosis	abductive and set-covering diagnosis	hypothetico-deductive diagnosis
Examples of systems	GDE	Theorist/CHECK	EMYCIN

Table 6.1: Comparison of typical conceptual models of diagnosis.

diagnostic problems in a domain, it is likely that a mixture of conceptual models of diagnosis as distinguished above will be required. Since the resulting systems use various types of knowledge, e.g. both knowledge of structure and behaviour, and empirical associations, the result is known as diagnosis with *multiple models* as suggested by Peter Struss.

Although in the literature it is emphasised that the conceptual models of diagnosis discussed embody different forms of diagnosis, they have much in common. For example, the type of knowledge used in DNSB diagnosis can be viewed as an implicit, or intensional, version of the type of knowledge used in AC diagnosis (if restricted to normality classification), which is an explicit or extensional type of knowledge; the associations between normal observable findings and the absence of defects are hidden in the specified normal behaviour in DNSB diagnosis. DNSB and MAB diagnostic problem solving are based on some kind of simulation of behaviour; such simulation of behaviour is absent in AC diagnosis.

6.3 Formal theories of diagnosis

There have been several attempts to formalise the various conceptual models of diagnosis discussed above; most, but not all, of these formalisations are based on logic. The most important formal theories of diagnosis will be reviewed below.

6.3.1 Consistency-based diagnosis

The formal theory of diagnosis originally proposed by Ray Reiter was motivated by the desire to provide a formal underpinning of diagnostic problem solving using knowledge of the normal structure and behaviour of technical devices, i.e. DNSB diagnosis. The theory of diagnosis may be viewed as the logical foundation of earlier work in DNSB diagnosis by Johan de Kleer et al, Brown and colleagues, Randy Davis, and Michael Genesereth. The logical formalisation uses results from earlier work by Ray Reiter, and John McCarthy on nonmonotonic reasoning. We shall sometimes refer to this theory of diagnosis as Reiter's formal theory of diagnosis.

Reiter's theory of diagnosis was later extended by De Kleer et al; in this section, both formalisations will be introduced in a single, logical framework. Where appropriate, the differences between Reiter's original proposal and the extensions proposed in De Kleer will be indicated. This formal theory of diagnosis is often referred to as the *consistency-based theory of diagnosis*, or *consistency-based diagnosis* for short.

The logical specification of knowledge concerning structure and behaviour in Reiter's theory is a triple $\mathcal{S} = (\mathrm{SD}, \mathrm{COMPS}, O)$, called a *system*, where

- SD denotes a finite set of formulae in first-order predicate logic, specifying normal structure and behaviour, called the *system description*;
- COMPS denotes a finite set of constants (nullary function symbols) in first-order logic, denoting the *components* of the system;
- O denotes a finite set of formulae in first-order predicate logic, denoting *observations*, i.e. observed findings.

It is, in principle, possible to specify normal as well as abnormal (faulty) behaviour within a system description SD, but originally SD was designed to comprise a logical specification of normal behaviour of the modelled system only, thus yielding the intended formalisation of

DNSB diagnosis. The essential part of a formal model of normal structure and behaviour of a system consists of logical axioms of the form

$$\neg \mathit{Abnormal}(c) \to o_{norm} \tag{6.1}$$

where $c \in \text{COMPS}$, and o_{norm} denotes a finding that may be observed if the component c is normal, i.e. is nondefective. The observable finding o_{norm} need not be unique. Axioms of the above form are provided for each component $c \in \text{COMPS}$. These axioms will be referred to as *normality axioms*. It is assumed that the finding o_{norm} may be observed in reality when component c of the device, that has been modelled in logic, is operating normally. Such an observed finding is called a *normality observation*. The subscript *norm* is used to emphasise that a particular finding represents a normal result; in Section 6.3.2 and further, the subscript *ab* is used to indicate an abnormal finding. These subscripts are only used for clarity and have no additional meaning; they will often be omitted. The predicate symbol '*Abnormal*' is sometimes referred to as the *fault mode* (also behavioural mode) of the component. The literal '$\mathit{Abnormal}(c)$' denotes the component c to be defective if satisfied. Other predicate names, such as 'OK', '*Correct*', are also employed in the literature, with similar intended meaning and use as the negation of an '*Abnormal*' literal.

Diagnostic problem solving is formalised as a method for finding the source of inconsistency in the logical description of the (normal) functioning of a system when supplied with observed findings, where some of the observed findings are the result of a system defect in reality. Hence, inconsistency formalises the notion of discrepancy in DNSB diagnosis as indicated in Figure 6.2. If it is assumed that the atom $\mathit{Abnormal}(c)$ is *false*, i.e. the component c is functioning normally, inconsistency will arise given the observed finding $\neg o_{norm}$ with logical implication (6.1). This result is interpreted in Reiter's theory as an indication that the defect may be localised in component c. This gives rise to the hypothesis that component c is defective, i.e. $\mathit{Abnormal}(c)$ is *true*, and the inconsistency is resolved if the assumption that $\mathit{Abnormal}(c)$ is *false* was its only source. This effect of relaxing logical constraints is sometimes referred to as *constraint suspension*.

Adopting the definition from De Kleer a diagnosis in the theory of consistency-based diagnosis can be defined as follows.

Definition 6.1 *(consistency-based diagnosis) Let $\mathcal{S} = (\text{SD}, \text{COMPS}, O)$ be a system. Let*

$$H_P = \{\mathit{Abnormal}(c) \mid c \in \text{COMPS}\}$$

be the set of all positive 'Abnormal' literals, and

$$H_N = \{\neg \mathit{Abnormal}(c) \mid c \in \text{COMPS}\}$$

be the set of all negative 'Abnormal' literals. Furthermore, let $H \subseteq H_P \cup H_N$ be a set, called a hypothesis*, such that*

$$H = \{\mathit{Abnormal}(c) \mid c \in D\} \cup \{\neg \mathit{Abnormal}(c) \mid \text{COMPS} \setminus D\}$$

for some $D \subseteq \text{COMPS}$. Then, the hypothesis H is a (consistency-based) diagnosis *of $\mathcal{S}$ if the following condition, called the* consistency condition*, holds:*

$$\text{SD} \cup H \cup O \nvDash \bot \tag{6.2}$$

i.e. $\text{SD} \cup H \cup O$ is consistent.

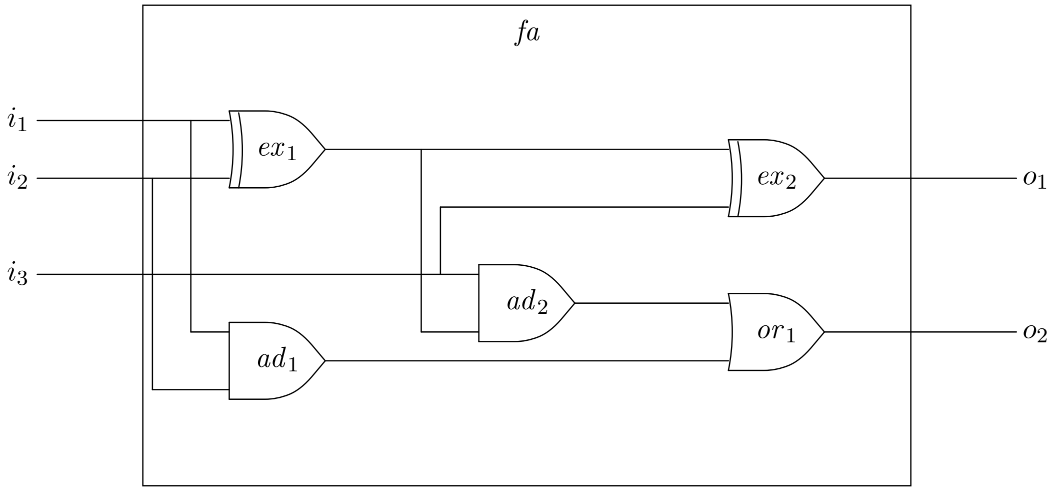


Figure 6.5: Full adder.

Here, $\not\models$ stands for the negation of the logical entailment relation, and $\perp$ represents 'falsum'. The consistency condition (6.2) captures DNSB diagnosis in terms of consistency-based diagnosis under the assumption that the axioms in SD provide a completely accurate and correct representation of a physical system. A diagnosis is just a hypothesis that is accepted. In the formalisation by De Kleer et al, each literal $Abnormal(c) \in H$ is interpreted as being defective; a literal $\neg Abnormal(c) \in H$ indicates component c to be nondefective. In the original theory by Reiter the set D above is taken as a diagnosis, with the extra requirement that D is minimal with respect to set inclusion. Then, each component c in a diagnosis D for which $Abnormal(c)$ is *true* is interpreted as being defective. According to expression (6.2), taking $D =$ COMPS leads to the trivial diagnosis that all components are defective (or the defective components are among the set of all components). Reiter, therefore, incorporated in the original theory the requirement that the set D must be a minimal set with respect to set inclusion, fulfilling the consistency condition. However, later it was recognised that minimality according to set inclusion is merely a measure of plausibility, which may not be appropriate when knowledge of abnormal behaviour is also included in the system description SD, and the minimality criterion was left out of the basic definition of De Kleer and others. Moreover, other measures of plausibility in the context of abduction may also apply. The application of the formal theory by Reiter is illustrated by a classical example.

EXAMPLE 6.1

Consider the logical circuit depicted in Figure 6.5, which represents a full adder, i.e. a circuit that can be used for the addition of two bits with carry-in and carry-out bits. The components X_1 and X_2 represent exclusive-OR gates, A_1 and A_2 represent AND gates, and R_1 represents an OR gate.

The system description consists of the following axioms:

$$\begin{aligned}
\forall x(\text{ANDG}(x) \wedge \neg Abnormal(x) &\rightarrow out(x) = and(in1(x), in2(x))) \\
\forall x(\text{XORG}(x) \wedge \neg Abnormal(x) &\rightarrow out(x) = xor(in1(x), in2(x))) \\
\forall x(\text{ORG}(x) \wedge \neg Abnormal(x) &\rightarrow out(x) = or(in1(x), in2(x)))
\end{aligned}$$

which describe the (normal) behaviour of each individual component (gate), and

$$\begin{aligned} out(X_1) &= in2(A_2) \\ out(X_1) &= in1(X_2) \\ out(A_2) &= in1(R_1) \\ in1(A_2) &= in2(X_2) \\ in1(X_1) &= in1(A_1) \\ in2(X_1) &= in2(A_1) \\ out(A_1) &= in2(R_1) \end{aligned}$$

which gives information about the connections between the components, i.e. information about the normal structure, including some electrical relationships. Finally, the various gates are defined:

$$\begin{array}{l} \mathrm{ANDG}(A_1) \\ \mathrm{ANDG}(A_2) \\ \mathrm{XORG}(X_1) \\ \mathrm{XORG}(X_2) \\ \mathrm{ORG}(R_1) \end{array}$$

Appropriate axioms for a Boolean algebra are also assumed to be available.

Now, let us assume that

$$O = \{in1(X_1) = 1, in2(X_1) = 0, in1(A_2) = 1, out(X_2) = 0, out(R_1) = 0\}$$

Note that $out(R_1) = 1$ is predicted using the model of normal structure and behaviour in Figure 6.5, which is in contrast with the observed output $out(R_1) = 0$. Assuming that $H = \{\neg Abnormal(c) \mid c \in \mathrm{COMPS}\}$, it follows that

$$\mathrm{SD} \cup H \cup O$$

is inconsistent. This confirms that some of the output signals observed differ from those expected under the assumption that the circuit is functioning normally. Using Formula (6.2), a possible diagnosis is, for instance,

$$\begin{aligned} H' = \{&Abnormal(X_1), \neg Abnormal(X_2), \neg Abnormal(A_1), \\ &\neg Abnormal(A_2), \neg Abnormal(R_1)\} \end{aligned}$$

since

$$\mathrm{SD} \cup H' \cup O$$

is consistent. In terms of Reiter's original definition, the corresponding diagnosis would be $D' = \{X_1\}$. Note that, given the diagnosis H', no output is predicted for the circuit; the assumption $Abnormal(X_1)$ completely blocks transforming input into output by the modelled circuit, because

$$\mathrm{SD} \cup H' \cup O \backslash \{out(X_2) = 0\} \nvDash out(X_2) = 0$$

In a sense, this is too much, because there was no discrepancy between the predicted and observed output of gate X_2. Nevertheless, the hypothesis H' is a diagnosis according to Definition 6.1.

It is interesting to look at consistency-based diagnosis in a more intuitive way. What the theory actually expresses is that if components that may be defective are removed from a system or device, and the resulting newly predicted behaviour, or no behaviour at all, does not contradict the observed behaviour, then a diagnosis has been established. This is a rather crude approach to diagnosis. Imagine that we have a formal model of an electrical device, including its electric plug, then simulating the removal of the plug from its socket, thus recovering consistency, will provide us with a diagnosis for a defective system. According to the theory, the plug will be identified as the culprit, which, of course, is absurd if the device was in operation prior to the removal of the plug, although incorrectly. Kurt Konolige refers to diagnoses produced by consistency-based diagnosis as *excuses*, to reflect that it may not be possible to explain such diagnoses in terms of cause-effect relationships.

In addition to a definition of consistency-based diagnosis De Kleer introduces the concepts of partial diagnosis and kernel diagnosis. A *partial diagnosis* is an abbreviated representation for a set of diagnoses that have certain '*Abnormal*' and '$\neg$*Abnormal*' literals in common. For example, in addition to H' in Example 1,

$$\begin{aligned}H'' = \{&\mathit{Abnormal}(X_1), \mathit{Abnormal}(X_2), \neg\mathit{Abnormal}(A_1),\\ &\neg\mathit{Abnormal}(A_2), \neg\mathit{Abnormal}(R_1)\}\end{aligned}$$

is also a diagnosis. The two diagnoses H' and H'' can be abbreviated as the partial diagnosis

$$P = \{\mathit{Abnormal}(X_1), \neg\mathit{Abnormal}(A_1), \neg\mathit{Abnormal}(A_2), \neg\mathit{Abnormal}(R_1)\}$$

which explicitly indicates that the actual status of component X_2 is irrelevant, adopting for the other components the status mentioned in the partial diagnosis P. Note that a partial diagnosis is not a real diagnosis according to Definition 6.1, because not all components are assigned unique '*Abnormal*' modes. A *kernel diagnosis* is simply a partial diagnosis that is minimal with respect to set inclusion.

Above, it was assumed that a system description SD is expressed using standard logic, using standard, monotonic logical entailment to define the notion of consistency-based diagnosis, but this is not essential. We may as well use some nonmonotonic logic. However, even when restricted to standard, monotonic logic, the notion of consistency-based diagnosis is nonmonotonic: observing additional findings may result in cancelling prior diagnoses.

EXAMPLE 6.2 ______

Reconsider the system description SD from Example 1. Assume that

$$O = \{\mathit{in1}(X_1) = 1, \mathit{in2}(X_1) = 0, \mathit{in1}(A_2) = 1, \mathit{out}(X_2) = 0\}$$

Then, the diagnosis is equal to

$$\begin{aligned}H'' = \{&\neg\mathit{Abnormal}(X_1), \neg\mathit{Abnormal}(X_2), \neg\mathit{Abnormal}(A_1),\\ &\neg\mathit{Abnormal}(A_2), \neg\mathit{Abnormal}(R_1)\}\end{aligned}$$

or, in Reiter's original notation: $D'' = \varnothing$ (there are no faults). Observing $out(R_1) = 0$ yields, among others, the diagnosis mentioned in Example 1 ($D' = \{X_1\}$ in Reiter's notation), but H'' ($D'' = \varnothing$) is not longer a diagnosis.

Reiter has also given an analysis of consistency-based diagnosis in terms of default logic. A system description SD and a set of observed findings O are supplemented with default rules of the form

$$\frac{: \neg Abnormal(c)}{\neg Abnormal(c)}$$

for each component c, yielding a default theory. A default rule as above expresses that $\neg Abnormal(c)$ may be assumed for component c, if assuming $\neg Abnormal(c)$ does not give rise to inconsistency. Hence, in computing an extension of the resulting default theory, these default rules will only be applied under the condition that they do not violate consistency, which is precisely the effect of the consistency condition (6.2). This mapping of a system $\mathcal{S}$ to default logic offers an object-level characterisation of the meta-level description of consistency-based diagnosis given in Definition 6.1.

In Section 6.4, the application of Reiter's theory to the logical formalisation of MAB diagnosis will be discussed. The techniques proposed by Reiter are not the only possible ways to formalise DNSB and MAB diagnosis; David Poole has proposed other logical techniques for the same purpose in terms of his Theorist framework of hypothetical reasoning. This work, however, bears great resemblance to the work by Reiter with respect to DNSB diagnosis, and to the work by Console and Torasso with respect to MAB diagnosis, which will be discussed in the following section. The Theorist framework is discussed in Section 6.4.

6.3.2 Abductive diagnosis

The formalisation of MAB diagnosis has been extensively studied by Luca Console and Pietro Torasso. In their theory, the abnormal behaviour of a system is specified in terms of abnormal states and resulting abnormal findings. Normal findings may also be included, but these are less useful for diagnosis, since an abnormal state is often causally related to a large number of normal findings. Diagnostic problem solving is formally described as the problem of accounting for a given set of observed findings, referred to in the theory as manifestations, by the simulation of abnormal behaviour. The simulation process is accomplished by deduction with logical axioms, describing abnormal behaviour, and assumed (abnormal) states.

The logical axioms are Horn clauses of the following form and meaning

$$State_1 \wedge \cdots \wedge State_n \;\rightarrow\; f \tag{6.3}$$

$$State_1 \wedge \cdots \wedge State_n \;\rightarrow\; State \tag{6.4}$$

$$State_1 \wedge \cdots \wedge State_n \;\rightarrow\; d \tag{6.5}$$

where $State$ and $State_i$, $i = 1, \ldots, n$, are positive literals representing part of the internal state of a modelled system, d is a *defect* (or disorder), and f is an *observable finding*. In a number of articles, extension to general Horn clauses (Horn clauses with negation as failure) is proposed. For simplicity's sake, we shall adopt the Horn-clause restriction in this section. It is assumed that the set of Horn clauses is hierarchical, i.e. no cyclic dependencies among atoms in clauses are allowed (which contrasts with the situation in logic programming, where cyclic dependencies are almost the rule). In the original abductive theory of diagnosis by Console and Torasso a finding appearing in the conclusion of a logical implication represents an *abnormal* finding. In later papers, owever, *normal* findings are also allowed. Recall that, when necessary for clarity, abnormal findings are denoted by f_{ab}; similarly, normal findings are denoted by f_{norm}.

A state literal is employed for the simulation of the occurrence of (abnormal) behaviour using the logical specification. It corresponds to a parameter with a value. For example, if the parameter *pressure*(*blood*) can take values *decreased*, *normal* and *increased*, then

$$\mathit{pressure}(\mathit{blood}) = \mathit{increased}$$

corresponds to a state. The intuitive meaning of formulae of the form (6.3) is: 'presence of $\mathit{State}_1, \ldots, \mathit{State}_n$ *causes* the (abnormal) finding f', i.e. if $\mathit{State}_1, \ldots, \mathit{State}_n$ hold in the system, (abnormal) finding f must be observed. Formulae of the form (6.4) express that a collection of states is causally related to another state, i.e. if the states $\mathit{State}_1, \ldots, \mathit{State}_n$ occur then *State* occurs as well. Axioms that conform to the two axiom schemata above are sometimes referred to as *abnormality axioms*. Note that the notion of causality is expressed in the theory using logical implication. Logical implication is employed to express a causal relationship between states and observable findings, and between states and states. Axioms of the form (6.5) can be viewed as *classification axioms* because they classify a collection of states as a particular defect. The idea originates from the CASNET system. If sufficient state literals are assumed or derived to satisfy the antecedent of an axiom of the form, a defect d can be derived. In the theory by Console and Torasso, a defect is actually defined in terms of a collection of states. This can be expressed by using a bi-implication ($\leftrightarrow$) instead of an implication, as in axiom schema. However, when adopting this formalisation for diagnosis, the implications from right to left ($\leftarrow$) are not involved. Classification axioms are not an essential ingredient of the theory of diagnosis by Console and Torasso; they are merely used to attach diagnostic labels to collections of states. Note that in the classification axioms, logical implication is used to express a classification instead of a causal relationship, as in the abnormality axioms. Due to the manifold uses of logical implication, the theory provides no clear logical meaning for the various relationships, including causality, underlying the theory of diagnosis. To express the theory in terms of defects and findings only, thus enabling us to analyse the essentials of the theory, states are identified with defects. Thus, axioms of the form (6.4) and (6.5) are collapsed into one axiom schema; the classification axioms are given no further consideration. In the following, it shall be assumed that axioms are of the following two forms:

$$d_1 \wedge \cdots \wedge d_n \quad \rightarrow \quad f \tag{6.6}$$

$$d_1 \wedge \cdots \wedge d_n \quad \rightarrow \quad d \tag{6.7}$$

where d, d_i, $i = 1, \ldots, n$, represent defects. We shall try to convey the essentials of the theory, using the uniform terminology and notation adopted in this book, thus deviating in some respects from the original papers.

Console and Torasso also provide a mechanism in their logical formalisation to weaken the causality relation. To this end, literals α are introduced into the premises of the axioms of the form (6.6) and (6.7), which can be used to block the deduction of an observable finding f or defect d if the defects d_i, $i = 1, \ldots, n$, hold true, by assuming the literal α to be false.

The weakened axioms have the following form:

$$d_1 \wedge \cdots \wedge d_n \wedge \alpha_f \rightarrow f \tag{6.8}$$
$$d_1 \wedge \cdots \wedge d_n \wedge \alpha_d \rightarrow d \tag{6.9}$$

The literals α are called *incompleteness-assumption literals*, abbreviated to *assumption literals*. Axioms of the form (6.6) – (6.9) are now taken as the (abnormality) axioms.

In the following, let $\Sigma = (\Delta, \Phi, \mathcal{R})$ stand for a *causal specification* in the theory of diagnosis by Console and Torasso, where:

- Δ denotes a set of possible defect and assumption literals;
- Φ denotes a set of possible (positive and negative) observable finding literals;
- $\mathcal{R}$ ('Causal Model') stands for a set of logical (abnormality) axioms of the form (6.6) – (6.9).

Subsets of the set Δ will be called *hypotheses*. A causal specification can then be employed for the prediction of observable findings in the sense of Figure 6.3.

Definition 6.2 *(prediction) Let $\Sigma = (\Delta, \Phi, \mathcal{R})$ be a causal specification. Then, a hypothesis $H \subseteq \Delta$ is called a* prediction *for a set of observable findings $E \subseteq \Phi$ if*

(1) $\mathcal{R} \cup H \vDash E$, *and*

(2) $\mathcal{R} \cup H$ *is consistent.*

Hence, the notion of prediction formalises the arrow in the lower half of Figure 6.3; the resulting set of findings E corresponds to the predicted (observable) findings in the same figure.

An *abductive diagnostic problem* $\mathcal{P}$ is now defined as a pair $\mathcal{P} = (\Sigma, E)$, where $E \subseteq \Phi$ is called a *set of observed findings*. A set of observed findings corresponds to the box in the upper half of Figure 6.3.

Formally, a solution to an abductive diagnostic problem $\mathcal{P}$ can be defined as follows.

Definition 6.3 *(solution) Let $\mathcal{P} = (\Sigma, E)$ be an abductive diagnostic problem, where $\Sigma = (\Delta, \Phi, \mathcal{R})$ is a causal specification with $\mathcal{R}$ a set of abnormality axioms of the form (6.6) – (6.9), and $E \subseteq \Phi$ a set of observed findings. A hypothesis $H \subseteq \Delta$ is called a* solution *to $\mathcal{P}$ if:*

(1) $\forall f \in E : \mathcal{R} \cup H \vDash f$ (covering condition)*;*

(2) $\forall f \in E^c : \mathcal{R} \cup H \nvDash \neg f$ (consistency condition)

where E^c is defined by:

$$E^c = \{\neg f \in \Phi \mid f \in \Phi, f \notin E,\ f \textit{ is a positive literal}\}$$

In the work by Console and Torasso, the set $\mathcal{R} \cup H$ is called a 'world' if H is a prediction; the set $\mathcal{R} \cup H$ is called a 'final world' if H is a solution to an abductive diagnostic problem. Note that the sets E and E^c are disjoint, and that if $f \in E$ then $\neg f \notin E^c$. The set E^c stands for findings assumed to be false, because they have not been observed (and are therefore assumed to be absent). But any finding may also be unknown. Thus, rather than providing a single definition, Console and Torasso provide in their articles several alternatives for this set E^c. The definition provided in Definition 6.3 above is just one of the alternatives.

Condition (1) is called the covering condition, because it requires that each observed finding is accounted for by a solution H. Note that any solution to a diagnostic problem $\mathcal{P} = (\Sigma, E)$ is a prediction for E according to Definition 6.2. Condition (2) is called the consistency condition, because it can be restated as follows

$$\mathcal{R} \cup H \cup E^c \nvDash \bot$$

A set of defects in a prediction H is also called a set of *perturbations*; the term *abducibles* is also employed for literals that may be assumed as part of diagnostic problem solving.

In the original formulation of the theory only those defects (states) are admitted to H which do not appear in the conclusions of implications; such defects are called *initial defects* (initial states in the original theory). The covering condition defined above ensures that sufficient defects and assumption literals are assumed to account for all given observed findings. The consistency condition helps to ensure that not too many defect and assumption literals are assumed. Although it is only necessary to include an assumption literal α in a solution for implications $d \wedge \alpha_f \to f$ and $d \wedge \alpha_{d'} \to d'$ if the defect d is deducible from the assumed (initial) defects and assumption literals, Definition 6.3 does not always prevent their inclusion in a solution.

An entire solution H may be taken as a diagnosis, but in one of his papers, Luca Console considers a diagnosis to consist of the defect literals in a solution H only.

Definition 6.4 *(abductive diagnosis) Let $\mathcal{P} = (\Sigma, E)$ be an abductive diagnostic problem, where $\Sigma = (\Delta, \Phi, \mathcal{R})$ is a causal specification. Let H be a solution to $\mathcal{P}$. Then, the set of all defects $D \subseteq H$ is called an* (abductive) diagnosis *of $\mathcal{P}$.*

Recall that a diagnosis is obtained by applying the classification axioms; a distinction is therefore made between a solution H for which the covering and consistency conditions are satisfied, i.e. the set of defect and assumption literals contained in a 'final world' – this world is called a *causal explanation* – and the set of defects resulting from an explanation, which is called a diagnosis (originally, a solution). However, from a formal point of view, the distinction is not essential.

EXAMPLE 6.3

Consider the causal specification $\Sigma = (\Delta, \Phi, \mathcal{R})$, with

$$\Delta = \{\mathit{fever}, \mathit{influenza}, \mathit{sport}, \alpha_1, \alpha_2\}$$

and

$$\Phi = \{\mathit{chills}, \mathit{thirst}, \mathit{myalgia}, \neg\mathit{chills}, \neg\mathit{thirst}, \neg\mathit{myalgia}\}$$

'Myalgia' means painful muscles. The following set of logical formulae R, representing medical knowledge concerning influenza and sport, both 'disorders' with frequent occurrence, is given:

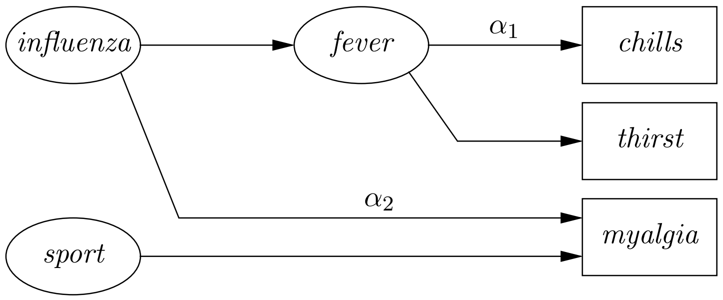


Figure 6.6: A knowledge base with causal relations.

$$\begin{aligned}
&\mathit{fever} \wedge \alpha_1 \rightarrow \mathit{chills}\\
&\mathit{influenza} \rightarrow \mathit{fever}\\
&\mathit{fever} \rightarrow \mathit{thirst}\\
&\mathit{influenza} \wedge \alpha_2 \rightarrow \mathit{myalgia}\\
&\mathit{sport} \rightarrow \mathit{myalgia}
\end{aligned}$$

For example, $\mathit{influenza} \wedge \alpha_2 \rightarrow \mathit{myalgia}$ means that influenza *may cause* myalgia; $\mathit{influenza} \rightarrow \mathit{fever}$ means that influenza *always causes* fever. For illustrative purposes, a causal knowledge base as given above is often depicted as a labelled, directed graph G, which is called a *causal net*, as shown in Figure 6.6. Suppose that the abductive diagnostic problem $\mathcal{P} = (\Sigma, E)$ must be solved, where the set of observed findings $E = \{\mathit{thirst}, \mathit{myalgia}\}$. Then, $E^c = \{\neg \mathit{chills}\}$. There are several solutions to this abductive diagnostic problem (for which the consistency and covering conditions are fulfilled):

$$\begin{aligned}
H_1 &= \{\mathit{influenza}, \alpha_2\}\\
H_2 &= \{\mathit{influenza}, \mathit{sport}\}\\
H_3 &= \{\mathit{fever}, \mathit{sport}\}\\
H_4 &= \{\mathit{fever}, \mathit{influenza}, \alpha_2\}\\
H_5 &= \{\mathit{influenza}, \alpha_2, \mathit{sport}\}\\
H_6 &= \{\mathit{fever}, \mathit{influenza}, \mathit{sport}\}\\
H_7 &= \{\mathit{fever}, \mathit{influenza}, \alpha_2, \mathit{sport}\}
\end{aligned}$$

The following diagnoses correspond to these solutions:

$$\begin{aligned}
D_1 &= \{\mathit{influenza}\}\\
D_2 &= \{\mathit{influenza}, \mathit{sport}\}\\
D_3 &= \{\mathit{fever}, \mathit{sport}\}\\
D_4 &= \{\mathit{fever}, \mathit{influenza}\}\\
D_5 &= \{\mathit{fever}, \mathit{influenza}, \mathit{sport}\}
\end{aligned}$$

For example, the diagnosis $D_4 = \{\mathit{fever}, \mathit{influenza}\}$ means that the patient has influenza with associated fever. Restricting to initial defects would yield the solutions H_1, H_2 and H_5 and the diagnoses D_1 and D_2. Finally, note that, for example, the hypothesis $H = \{\alpha_1, \alpha_2, \mathit{fever}, \mathit{influenza}\}$ is incompatible with the consistency condition.

Because in this theory of diagnosis, the observable findings are logically entailed by the assumption of the presence of certain states, and the reasoning goes in a sense in a direction reverse to that of the logical implication, i.e. from the consequent to the premise, the theory is often referred to as the *abductive theory of diagnosis*, or *abductive diagnosis* for short.

Several researchers have noted a close correspondence between abduction and the predicate completion of a logical theory, as originally proposed by K. Clark in connection with negation as finite failure in logic programming. Consider the following example.

EXAMPLE 6.4

Suppose that sport and influenza are two 'disorders'; this may be expressed in predicate logic as follows:

$$\begin{array}{l} \mathit{Disorder}(\mathit{sport}) \\ \mathit{Disorder}(\mathit{influenza}) \end{array}$$

The following logical implication is equivalent to the conjunction of the two literals above:

$$\forall x((x = \mathit{sport} \lor x = \mathit{influenza}) \to \mathit{Disorder}(x))$$

assuming the presence of the logical axioms for equality, and also assuming that constants with different names are not equal. Suppose that sport and influenza are the *only* possible disorders. This can be expressed by adding the following logical implication:

$$\forall x(\mathit{Disorder}(x) \to (x = \mathit{sport} \lor x = \mathit{influenza})) \tag{6.10}$$

to the implication above. For example, adding $\mathit{Disorder}(\mathit{asthma})$ to logical implication (6.10) yields an inconsistency, because *asthma* is neither equal to *sport* nor equal to *influenza*: the conclusion

$$\mathit{asthma} = \mathit{sport} \lor \mathit{asthma} = \mathit{influenza}$$

cannot be satisfied. Now, suppose that the literal $\mathit{Disorder}(\mathit{asthma})$ is removed, but that '*asthma*' remains a valid constant symbol. Then, $\neg\mathit{Disorder}(\mathit{asthma})$ is a logical consequence of formula (6.10); this formula 'completes' the logical theory by stating that disorders not explicitly mentioned are assumed to be false. Formula (6.10) is called a *completion formula*.

The characterisation of abduction as deduction in a completed logical theory is natural, because computation of the predicate completion of a logical theory amounts to adding the only-if parts of the formulae to the theory, i.e. it 'reverses the arrow' which is exactly what happens when abduction is applied to derive conclusions. After all, abductive reasoning is reasoning in a direction reverse to logical implication. In an intuitive sense, predicate completion expresses that the only possible causes (defects) for observed findings are those appearing in the abnormality axioms; assumption literals are taken as implicit causes. Where the characterisation of abduction by means of the covering and consistency conditions may be viewed as a meta-level description of abductive diagnosis, the predicate completion can be taken as the object-level characterisation, i.e. in terms of the original axioms in $\mathcal{R}$. However, in contrast to the predicate completion in logic programming, predicate completion should only pertain to literals appearing as a consequence of the logical axioms in $\mathcal{R}$, i.e. finding literals and defect literals that can be derived from other defects and assumption literals. This set of defects and observable findings is called the set of *non-abducible* literals, denoted by A; the set $\Delta\backslash A$ is then called the set of *abducible* literals.

Let us denote the axiom set $\mathcal{R}$ by

$$\begin{aligned}\mathcal{R} = \{&\varphi_{1,1} \rightarrow a_1, \ldots, \varphi_{1,n_1} \rightarrow a_1,\\ &\qquad\vdots\\ &\varphi_{m,1} \rightarrow a_m, \ldots, \varphi_{m,n_m} \rightarrow a_m\}\end{aligned}$$

where $A = \{a_i \mid 1 \leq i \leq m\}$ is the set of non-abducible (finding or defect) literals and each $\varphi_{i,j}$ denotes a conjunction of defect literals, possibly including an assumption literal. The predicate completion of $\mathcal{R}$ with respect to the non-abducible literals A, denoted by $\mathrm{COMP}[\mathcal{R}; A]$ is defined as follows:

$$\begin{aligned}\mathrm{COMP}[\mathcal{R}; A] = \mathcal{R} \cup \{&a_1 \rightarrow \varphi_{1,1} \vee \cdots \vee \varphi_{1,n_1},\\ &\qquad\vdots\\ &a_m \rightarrow \varphi_{m,1} \vee \cdots \vee \varphi_{m,n_m}\}\end{aligned}$$

The predicate completion of $\mathcal{R}$ makes explicit the fact that the only causes of non-abducible literals (findings and possibly also defects) are the defects and assumption literals given as a disjunct in the consequent. For example,

$$f_{ab} \rightarrow d_1 \vee \cdots \vee d_n$$

indicates that only the defects from the set $\{d_1, \ldots, d_n\}$ can be used to explain the observed finding f_{ab}.

Predicate completion of abnormality axioms with respect to a set of non-abducible literals can now be used to characterise diagnosis. Let ψ and ψ' be two logical formulae. It is said that ψ is *more specific than* ψ' iff $\psi \vDash \psi'$. Using the predicate completion of a set of abnormality axioms $\mathcal{R}$, we now have the following definition.

Definition 6.5 *(solution formula) Let $\mathcal{P} = (\Sigma, E)$ be an abductive diagnostic problem and let* $\mathrm{COMP}[\mathcal{R}; A]$ *be the predicate completion of $\mathcal{R}$ with respect to A, the set of non-abducible literals in $\mathcal{P}$. A* solution formula *S for $\mathcal{P}$ is defined as the most specific formula consisting only of abducible literals, such that*

$$\mathrm{COMP}[\mathcal{R}; A] \cup E \cup E^c \vDash S$$

where E^c is defined as in Definition 6.3.

Hence, abductive diagnosis is transformed to hypothetico-deductive diagnosis (cf. Section 6.3.4). A solution formula is obtained by applying the set of equivalences in $\mathrm{COMP}[\mathcal{R}; A]$ to a set of observed findings E, augmented with those findings not observed, E^c, yielding a logical formula that includes all possible solutions according Definition 6.3, given the equivalences in $\text{COMP}[\mathcal{R}; A]$. The following theorem reveals an important relationship between the meta-level characterisation of abductive diagnosis, as presented in Definition 6.3, and the object-level characterisation of diagnosis in Definition 6.5.[2]

THEOREM 8 *Let $\mathcal{P} = (\Sigma, E)$ be an abductive diagnostic problem, where $\Sigma = (\Delta, \Phi, \mathcal{R})$ is a causal specification. Let E^c be defined as in Definition 6.3, and let S be a solution formula for $\mathcal{P}$. Let $H \subseteq \Delta$ be a set of abducible literals, and let I be an interpretation of $\mathcal{P}$, such that for each abducible literal $a \in \Delta$: $\models_I a$ iff $a \in H$. Then, H is a solution to $\mathcal{P}$ iff $\models_I S$.*

Proof: ($\Rightarrow$): The set of defect and assumption literals H is a solution to $\mathcal{P}$, hence, for each $f \in E$: $\mathcal{R} \cup H \models f$, and for each $f' \in E^c$: $\mathcal{R} \cup H \not\models \neg f'$. The solution formula S is the result of rewriting observed findings in E and non-observed findings in E^c using the equivalences in $\text{COMP}[\mathcal{R}; A]$ to a formula merely consisting of abducibles. Assume that S is in conjunctive normal form. Conjuncts in S are equivalent to observed findings $f \in E$, that are logically entailed by $\mathcal{R} \cup H$, or to non-observed findings $\neg f \in E^c$ that are consistent with $\mathcal{R} \cup H$. Hence, an interpretation I for which $\models_I H$, that falsifies each abducible in $\Delta \backslash H$, satisfying every $f \in E$ and each $\neg f \in E^c$ that has been rewritten, must satisfy this collection of conjuncts, i.e. S.
($\Leftarrow$): If S is in conjunctive normal form, S must be the result of rewriting observed findings $f \in E$ and non-observed findings in E^c to (negative or positive) abducibles, using the equivalences in $\text{COMP}[\mathcal{R}; A]$. Since an interpretation I that satisfies H and S must also satisfy each finding $f \in E$ and those $\neg f \in E^c$ that have been rewritten to S, it follows that I can be chosen such that $\models_I E^c$, i.e. H must be a solution to $\mathcal{P}$. $\Diamond$

This theorem reveals an important property of the abductive theory of diagnosis. Sometimes, a solution to an abductive diagnostic problem is capable of satisfying a solution formula in the technical, logical sense.

EXAMPLE 6.5

Reconsider the set of logical axioms given in Example 3. The predicate completion of $\mathcal{R}$ is equal to

$$\begin{aligned}
&\text{COMP}[\mathcal{R}; \{\mathit{chills}, \mathit{thirst}, \mathit{myalgia}, \mathit{fever}\}]\\
&= \mathcal{R} \cup \{\mathit{chills} \rightarrow \mathit{fever} \wedge \alpha_1,\\
&\qquad \mathit{fever} \rightarrow \mathit{influenza},\\
&\qquad \mathit{thirst} \rightarrow \mathit{fever},\\
&\qquad \mathit{myalgia} \rightarrow (\mathit{influenza} \wedge \alpha_2) \vee \mathit{sport}\}\\
&= \{\mathit{chills} \leftrightarrow \mathit{fever} \wedge \alpha_1,\\
&\qquad \mathit{fever} \leftrightarrow \mathit{influenza},\\
&\qquad \mathit{thirst} \leftrightarrow \mathit{fever},\\
&\qquad \mathit{myalgia} \leftrightarrow (\mathit{influenza} \wedge \alpha_2) \vee \mathit{sport}\}
\end{aligned}$$

Note that

[2]Contrary to our treatment, a solution H of an abductive problem $\mathcal{P}$ is sometimes defined by SLD resolution with the negation as finite failure rule, i.e. SLDNF resolution, such that $\mathcal{R} \cup H \vdash_{\text{SLDNF}} E \cup E^c$, i.e. the covering and consistency conditions are merged.

$$\text{COMP}[\mathcal{R}; \{\textit{chills}, \textit{thirst}, \textit{myalgia}, \textit{fever}\}] \cup E \cup E^c \vDash (\textit{influenza} \wedge \alpha_2) \vee (\textit{influenza} \wedge \textit{sport})$$

given that $E = \{\textit{thirst}, \textit{myalgia}\}$ and $E^c = \{\neg\textit{chills}\}$. Although

$$\text{COMP}[\mathcal{R}; \{\textit{chills}, \textit{thirst}, \textit{myalgia}, \textit{fever}\}] \cup E \cup E^c \vDash \neg(\textit{fever} \wedge \alpha_1)$$

the formula $\neg(\textit{fever} \wedge \alpha_1)$, which is a logical consequence of $\neg\textit{chills}$ and $\textit{chills} \leftrightarrow (\textit{fever} \wedge \alpha_1)$, is not part of the solution formula $S \equiv (\textit{influenza} \wedge \alpha_2) \vee (\textit{influenza} \wedge \textit{sport})$, because the literal *fever* is non-abducible. It holds, in accordance with Theorem 8, that

$$\vDash_I H_i \;\Rightarrow\; \vDash_I (\textit{influenza} \wedge \alpha_2) \vee (\textit{influenza} \wedge \textit{sport})$$

for $i = 1, 2, 5$, where H_i is a solution given in Example 3 consisting only of abducible literals, for suitable interpretations I. Here, it even holds that $H_i \vDash S$, because S does not contain any negative defects or assumption literals entailed by non-observed findings in E^c.

Although the theory by Console and Torasso is restricted to reasoning with causal domain knowledge, other types of knowledge, referred to as *contextual information* by Console and Torasso, is also dealt with in the theory. Contextual information is incorporated to render the causal relation conditional on certain findings, e.g. in

$$d \wedge f \rightarrow f'$$

the finding literal f acts as a condition with regard to the causal relation between the defect d and the finding f'. For example, in a medical setting, many causal relations are age-specific; hence, the observed (normal) finding '$\textit{age} \circ v$', where $\circ$ denotes an ordering predicate and v an integer, could be employed to express such conditional causality.

Above we have defined abductive diagnosis using propositional logic. The definition in terms of predicate logic reveals some additional subtleties, yielding various alternative definition for the set of findings not observed and assumed to be absent, E^c. Findings f are denoted in predicate logic using a predicate symbol p, indicating a particular group of findings or a test. For example, in '*Sign*(*fever*)', the predicate symbol '*Sign*' denotes a group of patient findings; in '*Serum_copper*(*patient*, *high*)', the predicate symbol '*Serum_copper*' indicates the result of a diagnostic test. The consequences of using predicate logic to define abductive diagnosis will be briefly introduced by means of the following example.

EXAMPLE 6.6

Consider the following (partial) set of abnormality axioms $\mathcal{R}$, expressed in first-order predicate logic as follows:

$$\begin{array}{rcl}
\textit{Disorder}(\textit{influenza}) & \rightarrow & \textit{Symptom}(\textit{cough}) \\
\textit{Disorder}(\textit{influenza}) & \rightarrow & \textit{Sign}(\textit{fever}) \\
\textit{Disorder}(\textit{pulmonary\_embolism}) & \rightarrow & \textit{Blood\_chemistry}(O_2\textit{-level}, \textit{low})
\end{array}$$

where the (ground) literals $\mathit{Symptom}(\mathit{cough})$, $\mathit{Sign}(\mathit{fever})$ and $\mathit{Blood\_chemistry}(O_2\text{-}\mathit{level}, \mathit{low})$ stand for observable findings, and the '*Disorder*' literals represent defects. The finding literals f, representing abnormal observable findings, are taken from the following set of positive finding literals:

$$\begin{aligned}\Phi_P = \{&\mathit{Symptom}(\mathit{cough}), \mathit{Symptom}(\mathit{headache}),\\ &\mathit{Sign}(\mathit{fever}), \mathit{Sign}(\mathit{hypertension}),\\ &\mathit{Blood\_chemistry}(O_2\text{-}\mathit{level}, \mathit{low}), \mathit{Blood\_chemistry}(\mathit{Sodium}, \mathit{low})\}\end{aligned}$$

and the set of negative finding literals is equal to

$$\begin{aligned}\Phi_N = \{&\neg\mathit{Symptom}(\mathit{cough}), \neg\mathit{Symptom}(\mathit{headache}),\\ &\neg\mathit{Sign}(\mathit{fever}), \neg\mathit{Sign}(\mathit{hypertension}),\\ &\neg\mathit{Blood\_chemistry}(O_2\text{-}\mathit{level}, \mathit{low}), \neg\mathit{Blood\_chemistry}(\mathit{Sodium}, \mathit{low})\}\end{aligned}$$

with $\Phi = \Phi_P \cup \Phi_N$. Now, let $E = \{\mathit{Sign}(\mathit{fever}), \mathit{Blood\_chemistry}(O_2\text{-}\mathit{level}, \mathit{low})\}$ be a set of observed findings. Usually, it is assumed that $E \subseteq \Phi_P$, because only positive findings can be accounted for by Horn clauses in $\mathcal{R}$. The set $E^c \subseteq \Phi$, representing the findings not observed, is constructed in accordance with Definition 6.3. In the present case, the set E^c is equal to

$$\begin{aligned}E^c = \{&\neg\mathit{Symptom}(\mathit{cough}), \neg\mathit{Symptom}(\mathit{headache}),\\ &\neg\mathit{Sign}(\mathit{hypertension}),\\ &\neg\mathit{Blood\_chemistry}(\mathit{Sodium}, \mathit{low})\}\end{aligned}$$

Thus, test results denoted by the predicate symbol '*Symptom*' are assumed to be absent. Note that when applying this version of the consistency definition, obtained by the definition of E^c, the defect $\mathit{Disorder}(\mathit{influenza})$ cannot be part of any diagnosis, because this would clash with the consistency condition. Although, on first thought, the set

$$\{\mathit{Disorder}(\mathit{pulmonary\_embolism})\}$$

may seem to represent a diagnosis, it turns out that there exists no diagnosis at al. The reason is that

$$\mathcal{R} \cup \{\mathit{Disorder}(\mathit{pulmonary\_embolism})\} \nvDash \mathit{Sign}(\mathit{fever})$$

i.e., the covering condition fails to hold.

In a second, alternative version of the theory, the consistency condition is reformulated, by adopting another definition for the set E^c, as follows. The set $E^c \subseteq \Phi_N$ is defined by:

$$E^c = \{\neg\pi(t) \in \Phi_N \mid \pi(s) \in E, t \neq s\}$$

where π stands for a predicate symbol, and t and s are constants. The consistency condition remains the same, but its effects on the computation of a diagnosis differs, because of the altered definition of E^c. For the example diagnostic problem, the set E^c is equal to

$$E^c = \{\neg\mathit{Sign}(\mathit{hypertension}), \neg\mathit{Blood\_chemistry}(\mathit{Sodium}, \mathit{low})\}$$

Note that the literals $\neg Symptom(cough)$ and $\neg Symptom(headache)$ are missing from this set, because none of the literals in the set of observed findings E has '*Symptom*' as predicate symbol. Thus, the test results with respect to test '*Symptom*' are assumed to be unknown. A diagnosis in this case is $H = \{Disorder(influenza), Disorder(pulmonary\_embolism)\}$, because

$$\mathcal{R} \cup H \vDash \{Sign(fever), Blood\_chemistry(O_2\text{-}level, low)\}$$

(in fact, the literal $Symptom(cough)$ is also entailed), and E^c is consistent with $\mathcal{R}$ and H. Note that $H = \{Disorder(influenza), Disorder(pulmonary\_embolism)\}$ yields an inconsistency if taken as a hypothesis using the first version of the consistency condition.

The intuitive basis of the two versions of the consistency condition in abductive diagnosis, yielded by different logical interpretations of findings not observed, can be clarified in terms of diagnostic problem solving as follows.[3] In the first version of the consistency condition, it is assumed that all findings associated with a defect, present in the real world, will be observed. If a finding is not included among the findings in the set of observed findings, it is assumed to be absent; absent findings are denoted by negative literals. The basic assumption is that all findings of defects that are absent will not be observed, i.e. are absent (if unique for the defect), hence, it can safely be assumed that all findings not observed are negative. Although this may not be justified in diagnostic problem solving – it could be more natural to take the findings as unknown – the assumption of the negative literals has the technical advantage of blocking the inclusion of defects that are not present in the real world according to the theory, because some observable finding associated with the defect is not included in the set of observed findings. This is precisely the effect required. Now, if, as in the example above, only part of the unique findings of a defect occurs among the set of observed findings, there must be something wrong, either with the abnormality axioms $\mathcal{R}$, or with the set of observed findings. It seems therefore justified that no diagnosis is established in this case. However, this result is only valid if one accepts as a basic assumption that every possible cause (defect) of a finding is included in the set of abnormality axioms $\mathcal{R}$, which also constituted the basis of the predicate completion discussed above (at the risk of ambiguity with respect to database theory, one might call this the closed world assumption of abduction).

The second version of the consistency condition in abductive diagnosis is similar to the first version, except that it is assumed that if no information concerning a specific diagnostic test is available,– recall that every test corresponds to a different predicate symbol – it is assumed to be unknown. Now, if some defect d is included in a solution H and

$$\mathcal{R} \cup \{d\} \vDash f$$

where $f \notin E$, this means that the model predicts that if the test is actually carried out, the finding f will be observed. If it is not observed, or turns out to be false, i.e. $\neg f$, some action needs to be undertaken, but no specific ideas concerning this situation appear in the papers of Console and Torasso. However, if the test has been carried out, i.e. there exists some finding f' with the same predicate symbol as f, and $f \notin E$, then again no diagnosis exists, because $\neg f \in E^c$ would hold.

[3]We remark that this interpretation is the author's own, no such interpretation appears in the papers by Console and Torasso.

The abductive theory of diagnosis discussed above may be viewed as a formalisation of particular parts of the diagnostic system shell CHECK. This system can be used to build hybrid diagnostic systems for domains in which causal, hierarchical and heuristic knowledge coexist. As far as known to the author, CHECK has been used as an experimental platform on which various prototype systems have been developed, including diagnosis of automobile engine failure and diagnosis of liver disease.

6.3.3 Set-covering theory of diagnosis

Instead of choosing logic as the language for MAB diagnosis, as discussed above, others have adopted set theory as their formal language. This approach to the formalisation of diagnosis is referred to as the *set-covering theory of diagnosis*, or *parsimonious covering theory.* The treatment of the set-covering theory of diagnosis in the literature deals only with the modelling of restricted forms of abnormal behaviour of a system.

The specification of the knowledge involved in diagnostic problem solving consists of the enumeration of all findings that may be present (and observed) given the presence of each individual defect distinguished in the domain; the association between each defect and its associated set of observable findings is interpreted as an uncertain *causal relation* between the defect and each of the findings in the set of observable findings. Instead of the terms 'defect' and 'finding' the terms 'disorder' and 'manifestation' are employed in descriptions of the set-covering theory of diagnosis. In the following, we have chosen to uniformly employ the terms 'defect' and 'finding' instead. The basic idea of the theory with respect to diagnosis is that each finding in the set of observed findings in a given diagnostic situation must be causally related to at least one present defect; the collected set of present defects thus obtained can be taken as a diagnosis. As with the theory of diagnosis by Console and Torasso, this reasoning method is usually viewed as being abductive in nature, because the reasoning goes from findings to defects, using causal knowledge from defects to findings.

More formally, the triple $\mathcal{N} = (\Delta, \Phi, C)$ is called a *causal net* in the set-covering theory of diagnosis, where

- Δ is a set of possible *defects*,
- Φ is a set of elements called *observable findings*, and
- C is a binary relation

  $$C \subseteq \Delta \times \Phi$$

  called the *causation relation.*

A *diagnostic problem* in the set-covering theory of diagnosis is then defined as a pair $\mathcal{D} = (\mathcal{N}, E)$, where $E \subseteq \Phi$ is a *set of observed findings.* It is assumed that all defects $d \in \Delta$ are potentially present in a diagnostic problem, and all findings $f \in \Phi$ will be observed when present. In addition, all defects $d \in \Delta$ have a causally related observable findings $f \in \Phi$, and vice versa, i.e. $\forall d \in \Delta\, \exists f \in \Phi : (d, f) \in C$, and $\forall f \in \Phi\, \exists d \in \Delta : (d, f) \in C$. No explicit distinction is made in the theory between positive (present), negative (absent) and unknown defects, and positive (present), negative (absent) and unknown findings. The causation relation is often depicted by means of a labelled, directed acyclic graph, which, as $\mathcal{N}$, is called a *causal net.*

Let $\wp(X)$ denote the power set of the set X. It is convenient to write the binary causation relation C as two functions. Since in the next section, such functions are intensively employed, we adopt a notation that slightly generalises the notation originally proposed.[4] The first function

$$e : \wp(\Delta) \rightarrow \wp(\Phi)$$

called the *effects function*, is defined as follows; for each $D \subseteq \Delta$:

$$e(D) = \bigcup_{d \in D} e(\{d\}) \tag{6.11}$$

where

$$e(\{d\}) = \{f \mid (d, f) \in C\}$$

and the second function

$$c : \wp(\Phi) \rightarrow \wp(\Delta)$$

called the *causes function*, is defined as follows; for each $E \subseteq \Phi$:

$$c(E) = \bigcup_{f \in E} c(\{f\})$$

where

$$c(\{f\}) = \{d \mid (d, f) \in C\}$$

Hence, knowledge concerning combinations of findings and defects is taken as being composed of knowledge concerning individual defects or findings, which is not acceptable in general. This is a strong assumption, because it assumes that no interaction occurs between defects.

A causal net can now be redefined, in terms of the effects function e above, as a triple $\mathcal{N} = (\Delta, \Phi, e)$.

Given a set of observed findings, diagnostic problem solving amounts to determining sets of defects – technically the term *cover* is employed – that account for *all* observed findings. Formally, a diagnosis is defined as follows.

Definition 6.6 *(set-covering diagnosis) Let $\mathcal{D} = (\mathcal{N}, E)$ be a diagnostic problem, where $\mathcal{N} = (\Delta, \Phi, e)$ is a causal net and E denotes a set of observed findings. Then, a* (set-covering) diagnosis *of $\mathcal{D}$ is a set of defects $D \subseteq \Delta$, such that:*

$$e(D) \supseteq E \tag{6.12}$$

In the set-covering theory of diagnosis the technical term ‘cover’ is employed instead of ‘diagnosis’; ‘diagnosis’ will be the name adopted in this book. Due to the similarity of condition (6.12) with the covering condition in the abductive theory of diagnosis, this condition is called the *covering condition* in the set-covering theory of diagnosis. Actually, set-covering diagnosis can be mapped to abductive diagnosis in a straightforward way, thus revealing that

[4]In the original definition of set-covering diagnosis, function e for singleton sets is called *effects*, and is defined for elements only. They also define an associated function *Effects*, which is defined on sets of defects, in terms of the *effects* function. This function is identical to our function e. Hence, the *effects* function is superfluous. Similarly, the functions corresponding to the function c are called *causes* and *Causes*.

set-covering diagnosis is more restrictive than abductive diagnosis. Just by mapping each function value

$$e(\{d\}) = \{f_1, \ldots, f_n\}$$

to a collection of logical implications, taken as abnormality axioms $\mathcal{R}$ of a causal specification $\Sigma = (\Delta, \Phi, \mathcal{R})$, of the following form:

$$\begin{aligned} d \wedge \alpha_{f_1} &\rightarrow f_1 \\ d \wedge \alpha_{f_2} &\rightarrow f_2 \\ &\vdots \\ d \wedge \alpha_{f_n} &\rightarrow f_n \end{aligned}$$

abductive diagnosis for such restricted causal specifications and set-covering diagnosis coincide.

Since it is assumed that $e(\Delta) = \Phi$ is satisfied, i.e. any finding $f \in \Phi$ is a possible causal effect of at least one defect $d \in \Delta$, there exists a diagnosis for any set of observed findings E, because

$$e(\Delta) \supseteq E$$

always holds (explanation existence theorem).

A set of defects D is said to be an *explanation* of a diagnostic problem $\mathcal{D} = (\mathcal{N}, E)$, with E a set of observed findings, if D is a diagnosis of E and D satisfies some additional criteria. Various criteria, in particular so-called *criteria of parsimony*, are in use. The basic idea is that among the various diagnoses of a set of observable findings, those that satisfy certain criteria of parsimony are more likely than others. Let $\mathcal{D} = (\mathcal{N}, E)$ be a diagnostic problem, then some of the criteria are:

- *Minimal cardinality*: a diagnosis D of E is an explanation of $\mathcal{D}$ iff it contains the minimum number of elements among all diagnoses of E;
- *Irredundancy*: a diagnosis D of E is an explanation of $\mathcal{D}$ iff no proper subset of D is a diagnosis of E;
- *Relevance*: a diagnosis D of E is an explanation of $\mathcal{D}$ iff $D \subseteq c(E)$;
- *Most probable diagnosis*: a diagnosis D of E is an explanation of $\mathcal{D}$ iff $P(D|E) \geq P(D'|E)$ for any diagnosis D' of E.

In addition, some researchers define the concept of minimal-cost diagnosis. A diagnosis D of a set of observed findings E is called a *minimal-cost explanation* of $\mathcal{D}$ iff

$$\sum_{d \in D} cost(d) \leq \sum_{d \in D'} cost(d)$$

for each diagnosis D' of E, where *cost* is a function associating real values with defects $d \in \Delta$. The cost of a diagnosis may be anything, varying from financial costs to some subjective feeling of importance expressed by numbers. However, Eugene Charniak choose as a semantics of cost function information for the negative logarithm of probabilities. Under this interpretation, a minimal-cost diagnosis is identical to a most probable diagnosis.

Although not every diagnosis is an explanation, any diagnosis may be seen as a solution to a diagnostic problem, where diagnoses which represent explanations conform to more strict conditions than diagnoses that do not. The term 'explanation' refers to the fact that a diagnosis in the set-covering theory of diagnosis can be stated, and thus be explained, in terms of cause-effect relationships. A better choice, in our opinion, would have been the adoption of the term 'explanation' for what is now called 'cover' in the theory, and to refer to what are now called 'explanations' by the name of 'parsimonious explanations'. To avoid confusion, the term 'explanation' will not be used in the sequel. Instead, we shall speak of a 'minimal-cardinality diagnosis', an 'irredundant diagnosis', a 'minimal-cost diagnosis' and so on.

For minimal cardinality, a diagnosis which consists of the smallest number of defects among all diagnoses is considered the most plausible diagnosis. Minimal cardinality is a suitable parsimony criterion in domains in which large combinations of defects are unlikely to occur. For example, in medicine, it is generally more likely that a patient has a single disorder than more than one disorder. Irredundancy expresses that it is not possible to leave out a defect from an explanation without losing the capability of explaining the complete set of observed findings, i.e.

$$e(D) \not\supseteq E$$

for each $D \subset D'$, where D' is an irredundant diagnosis. The relevance criterion states that every defect in an explanation has at least one observable finding in common with the set of observed findings. This seems an obvious criterion, but note that the notion of uncertain causal relation employed in the set-covering theory of diagnosis does not preclude situations in which a defect is present, although none of its causally related observable findings have been observed. These three definitions of the notion of explanation are based on general set-theoretical considerations. In contrast, the most probable diagnosis embodies some knowledge of the domain, in particular with respect to the strengths of the causal relationships. We shall not deal with such probabilistic extensions of the set-covering theory of diagnosis any further.

EXAMPLE 6.7

Consider the causal net $\mathcal{N} = (\Delta, \Phi, C)$, where the effects function e is defined by the causation relation C, i.e.

$$e(D) = \bigcup_{d \in D} e(\{d\})$$

where

$$e(\{d\}) = \begin{cases} \{cough, fever, sneezing\} & \text{if } d = influenza \\ \{cough, sneezing\} & \text{if } d = common\ cold \\ \{fever, dyspnoea\} & \text{if } d = pneumonia \end{cases}$$

It states, for example, that a patient with influenza will be coughing, sneezing and have a fever; a patient with a common cold will show the same findings, except fever, and a patient with pneumonia will have a fever and dyspnoea (shortness of breath). The

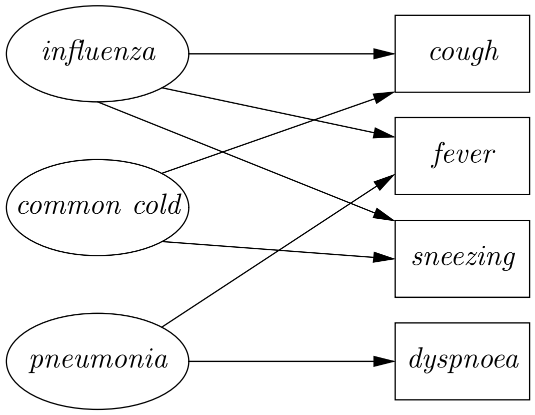


Figure 6.7: Causal net.

associated graph representation G_C of C is shown in Figure 6.7. It holds, among others, that

$$e(\{\mathit{influenza}, \mathit{common\ cold}\}) = \{\mathit{cough}, \mathit{fever}, \mathit{sneezing}\}$$

Based on the causal net C, the following causes function c is obtained:

$$c(E) = \bigcup_{o \in E} c(\{o\})$$

with

$$c(\{f\}) = \begin{cases} \{\mathit{influenza}, \mathit{common\ cold}\} & \text{if } f = \mathit{cough} \\ \{\mathit{influenza}, \mathit{pneumonia}\} & \text{if } f = \mathit{fever} \\ \{\mathit{influenza}, \mathit{common\ cold}\} & \text{if } f = \mathit{sneezing} \\ \{\mathit{pneumonia}\} & \text{if } f = \mathit{dyspnoea} \end{cases}$$

Suppose $\mathcal{D} = (\mathcal{N}, E)$ is a diagnostic problem, with $E = \{\mathit{cough}, \mathit{fever}\}$ a set of observed findings, then a diagnosis of $\mathcal{D}$ is

$$D_1 = \{\mathit{influenza}\}$$

but

$$\begin{aligned} D_2 &= \{\mathit{influenza}, \mathit{common\ cold}\} \\ D_3 &= \{\mathit{common\ cold}, \mathit{pneumonia}\} \end{aligned}$$

and $D_4 = \{\mathit{influenza}, \mathit{common\ cold}, \mathit{pneumonia}\}$ are also diagnoses for E. All of these diagnoses are relevant diagnoses, because

$$c(\{\mathit{cough}, \mathit{fever}\}) \supseteq D_i$$

where $i = 1, \ldots, 4$. Irredundant diagnoses of E are D_1 and D_3. There is only one minimal cardinality diagnosis, viz. $D_1 = \{\mathit{influenza}\}$. Now suppose that $E = \{\mathit{cough}\}$, then for example $D = \{\mathit{influenza}, \mathit{pneumonia}\}$ would not have been a relevant diagnosis, because

$$c(\{\mathit{cough}\}) = \{\mathit{influenza}, \mathit{common\ cold}\} \not\supseteq D$$

Other, more domain-specific, definitions of the notion of explanation have only been developed recently. Such domain-specific knowledge can be effective in reducing the size of the set of diagnoses generated by a diagnostic system. For example, Tuhrim et al. demonstrated that the use of knowledge concerning the three-dimensional structure of the brain by means of a binary adjacency relation in a neurological diagnostic expert system, based on the set-covering theory of diagnosis, could increase the diagnostic accuracy of the system considerably.

Peng and Regia have also shown that the causation relation C can be extended for the representation of multi-layered causal nets, in which defects are causally connected to each other, finally leading to observable findings. By computation of the reflexive, transitive closure of the causation relation, $C^\star$, the basic techniques discussed above immediately apply. The reflexive closure makes it possible to enter defects as observed findings, which are interpreted as already established defects, yielding a slight extension to the theory treated above.

6.3.4 Hypothetico-deductive diagnosis

The third approach to diagnosis mentioned in Section 6.2, AC (Abnormality Classification) diagnosis, originates from work by Ted Shortliffe, Bruce Buchanan, William Clancey and Edward Feigenbaum in the MYCIN project. The knowledge incorporated in that expert system, and in similar systems for AC diagnosis, is based on the body of experience accumulated in handling a large number of cases, such as the patients a physician sees in medical practice. The knowledge is extracted from textbooks or human experts. We have called this type of knowledge empirical associations, i.e. the knowledge consists of associations between typical observable findings and defects; knowledge about the underlying mechanisms (if available) is not represented.

In most practical systems,, the formal counterparts of empirical associations are organised according to some underlying model distinguished in the collection of empirical associations. A typical example is a distinction between families of disorders and specific disorders, i.e. a taxonomy of disorders, that can be exploited in problem solving. Hence, expert systems based on empirical associations are model-based like the other systems discussed above, because they are also based on a model of the problem domain, although the nature of the model is different. It is possible to characterise AC diagnosis in a more formal way. We shall refer to this formal counterpart of AC diagnosis as *hypothetico-deductive diagnosis*.

A hypothetico-deductive diagnostic problem consists of a set of logical axioms, called an *empirical model* EM, of the form

$$c_1 \wedge \cdots \wedge c_n \to q \tag{6.13}$$

where c_i and q represent either negative or positive defects and findings, represented in logic as negative or positive literals, and if every c_i is a finding, then q should be a defect. Logical implication in the formalisation of empirical associations (6.13) may be viewed as a *classification relation*. A set of observed findings is represented as a set of ground literals, where each literal is of the finding type. For example, a typical logical axiom might be

$$f_1 \wedge \cdots \wedge f_m \to d$$

which expresses that a set of observable findings $E = \{f_1, \ldots, f_m\}$ represents necessary and sufficient evidence for establishing the presence of the defect d as part of a diagnosis. One difference between the theories of hypothetico-deductive diagnosis and abductive diagnosis is that, in hypothetico-deductive diagnosis, observed findings and defects need not be causally related to each other. Some of the findings may be interpreted as abnormal; other findings, such as, for example, age of a patient in a medical application, may not. The function of normal findings in empirical associations is similar to that of conditional causality introduced in Section 6.3.2, viz. to condition a particular piece of knowledge on a specific piece of evidence.

Now, let $\mathcal{B} = (\Delta, \Phi, \mathrm{EM})$ denote an *associational specification*, where:

- Δ denotes a set of (positive and negative) possible defects,
- Φ denotes a set of (positive and negative) observable findings, and
- EM denotes the logical representation of a set of empirical associations of the form (6.13).

A *hypothetical-deductive diagnostic problem* is then defined as a pair $\mathcal{H} = (\mathcal{B}, E)$, where $E \subseteq \Phi$ denotes a *set of observed findings.* A diagnosis based on empirical associations can be defined as follows.

Definition 6.7 *(hypothetico-deductive diagnosis) Let $\mathcal{H} = (\mathcal{B}, E)$ be a hypothetico-deductive diagnostic problem, where $\mathcal{B} = (\Delta, \Phi, \mathrm{EM})$ is an associational specification, and E is a set of observed findings. Let $\Theta \subseteq \Delta$ be a set of defects, called a* hypothesis*. Then, $D \subseteq \Theta$ is called a* (hypothetico-deductive) diagnosis *of $\mathcal{H}$ if*

$$D = \{d \in \Theta \mid \mathrm{EM} \cup E \vDash d\}$$

Note that, in contrast with the theories discussed above, a single hypothesis is initially given in hypothetico-deductive diagnosis; it stands for the defects that are initially given to be of interest. In the theory of hypothetico-deductive diagnosis, defects are logically entailed by the observed findings (usually implemented by a deductive calculus, hence the adjective hypothetico-*deductive*).

In contrast with the other theories of diagnosis, there are a large number of nonexperimental applications available that may be viewed as hypothetico-deductive diagnostic systems.

The technical characteristics of the various formal theories of diagnosis, discussed in the previous sections, are summarised in Table 6.2.

6.4 Frameworks of diagnosis

Having described the various formal theories of diagnosis, the question arises in what sense these theories are related to each other, and whether it is possible to develop generalisations based on these theories. Actually, several originators of theories of diagnosis have investigated the expressiveness of their theory for modelling other conceptual models of diagnosis than those for which the theory was originally designed. In this section, we summarise and comment on results found in the literature, and discuss various general frameworks of diagnosis.

Originator	**Knowledge base specification**	**Knowledge base interpretation**	**Diagnosis**
Reiter	functional relations	deduction	consistency
Console & Torasso	causality	abduction	covering
		deduction	consistency
Reggia et al.	causality	abduction	set covering
Bylander et al.	diagnostic relation	none	set covering
Shortliffe et al.	empirical associations	deduction	classification

Table 6.2: Comparison of formal theories of diagnosis.

6.4.1 Expressiveness of theories of diagnosis

Reiter has shown that the framework of consistency-based diagnosis provides enough descriptive power to capture the set-covering theory of diagnosis. In Reiter's formalisation, the normality axioms in the original theory of consistency-based diagnosis are changed into *abnormality axioms*, simply by replacing 'components' by 'defects'. These axioms have the following form

$$\neg Abnormal(d) \rightarrow \neg Present(d) \tag{6.14}$$

for each defect d, stating that under normal conditions defect d is not present, and

$$f_{ab} \rightarrow Present(d_1) \vee \cdots \vee Present(d_n) \tag{6.15}$$

for each observable abnormal finding f_{ab} and related defect d_i, $i = 1, \ldots, n$. Formulae of the form (6.14) express hypotheses, namely that a particular defect may be absent ($\neg Present(d)$) if it does not give rise to an inconsistency. As we have discussed in Section 6.3.2, formulae of the form (6.15) may be seen as the predicate completion of finding literals in formulae of the form

$$Present(d) \rightarrow f_{ab}$$

i.e. if $\mathcal{R}$ denotes the set of formulae of the last form, with defect literals $Present(d_1)$, ..., $Present(d_n)$ in the premise, then the predicate completion COMP$[\mathcal{R}; f_{ab}]$ with regard to the finding f_{ab} is equal to

$$\text{COMP}[\mathcal{R}; f_{ab}] = \mathcal{R} \cup \{f_{ab} \rightarrow Present(d_1) \vee \cdots \vee Present(d_n)\}$$

This states that the only causes of the finding f_{ab} to be present (and observed) are the defects $d_1, \ldots, d_n$. As discussed above, this same kind of knowledge is expressed, although implicitly, in the abductive theory of diagnosis; it is also expressed in the set-covering theory of diagnosis, but the differences between the reasoning methods employed (consistency-based reasoning, logical abduction, and set covering) dictate a different representation (syntax) in all three formal theories. Informally, in the consistency-based diagnosis formalisation of MAB diagnosis, diagnostic problem solving is carried out as follows. Given an observed finding f_{ab} associated with a defect d_i, $i = 1, \ldots, n$, a disjunction

$$Present(d_1) \vee \cdots \vee Present(d_n)$$

is deduced, which is reduced by cancelling out atoms using axiom (6.14), assuming certain defects not to be present, i.e. $Abnormal(d)$ is *false*, yielding a (subset minimal) diagnosis. The effect of axiom (6.14) corresponds to producing irredundant diagnoses in the set-covering theory of diagnosis, in the sense that a minimal diagnosis with respect to set inclusion is produced. Reiter shows that there exists a (subset minimal) diagnosis according to the consistency-based reformulation of the set-covering theory of diagnosis iff there exists an equivalent irredundant diagnosis in the set-covering theory (although at the time Reiter's result was published, the notion of irredundant diagnosis had not yet appeared in the literature).

Console and Torasso have studied the use of the consistency condition in abductive diagnosis for modelling DNSB diagnosis, i.e. diagnosis using a specification of a model of normal structure and behaviour in a way resembling the work of Reiter. By taking the empty set for the set of observed findings that must be covered, the covering condition in abductive diagnosis becomes

$$\mathcal{R} \cup H \vDash E'$$

where $E' = \varnothing$; a diagnosis is the result of satisfaction of the consistency condition only, because the covering condition is always satisfied in this case. Thus, consistency-based diagnosis in the sense of Reiter is obtained. However, the meaning of the logical axioms is entirely different from the meaning originally attached to the logical axioms, because they now represent normal behaviour of a device; d represents some normal state of a component of the device and a finding f in the conclusion of a Horn clause $d \rightarrow f$ represents a finding that may be observed when the component is in its normal state, i.e. f represents a normality finding f_{norm}. By varying between $E' = \varnothing$ and $E' = E$, for example by taking for E' the set of all abnormal findings f_{ab} occurring in E, DNSB and MAB diagnosis can be integrated within the same abductive framework. The resulting abductive framework is referred to as 'the spectrum of logical definitions of diagnosis'.

We may conclude by saying that generalisation of the formal theories of diagnosis discussed above has shown that there is no such thing as a unique formalisation of a conceptual model of diagnosis. Although the formal theories can be applied to formalise conceptual models of diagnosis other than those for which they were originally designed, the results often lack conceptual clarity.

6.4.2 Generalisation towards frameworks of diagnosis

The principal difficulty of developing a theory of diagnosis lies, undoubtedly, in the design of a mapping of some intuitively appealing conceptual model of diagnosis to a formal language, such as logic or set theory. We know beforehand that both logic and set theory are sufficiently expressive; so, this is not where the problem lies. The selection of an appropriate logic, or an appropriate fragment of set theory, however, is much more difficult. The insights gained from the formal theories discussed in Section 6.3 have facilitated researchers in coming up with more general frameworks of diagnosis.

David Poole and colleagues have developed a theory and an implementation of a form of hypothetical reasoning, called *Theorist*. Theorist may be used as a framework of diagnosis, but it is not restricted in any way to diagnostic problem solving. Moreover, there are no inherent relationships between Theorist and any of the conceptual models of diagnosis. The present implementation of the Theorist framework, however, is more or less tailored to abductive diagnosis.

```
default a1.
default a2.
default fever.
default influenza.
default sport.

fact chills  <- fever and a1.
fact fever   <- influenza.
fact thirst  <- fever.
fact myalgia <- influenza and a2.
fact myalgia <- sport.

constraint not chills.  % Ec
```

Figure 6.8: Specification of an abductive diagnostic problem in Theorist.

In Theorist, a diagnostic problem must be specified in terms of a set of *facts*, denoted by FACTS, a set of *hypotheses*, denoted by HYP, and a set of *constraints*, denoted by C. The set of facts FACTS and constraints C are collections of arbitrary closed formulae in first-order logic; hypotheses act as a kind of defaults that might become instantiated, and assumed to hold true, in the reasoning process. A set $\text{FACTS} \cup H$ is called an *explanation* of a closed formula g, where H is a set of ground instances of hypothesis elements in HYP, iff:

(1) $\text{FACTS} \cup H \vDash g$, and

(2) $\text{FACTS} \cup H \cup C \nvDash \bot$.

On first sight, the framework looks a lot like the framework of abductive diagnosis discussed in Section 6.3.2, but it is much more general, mainly due to the unrestricted nature of its elements. In terms of the abductive theory of diagnosis, we would have called H a solution, if the abnormality axioms $\mathcal{R}$ were taken as FACTS, the set of findings not observed E^c as constraints C, and the set of observed findings E as g. Obviously, because there is no fixed diagnostic interpretation in Theorist, the framework can be used as a basis for various other notions of diagnosis, such as consistency-based diagnosis (just take $g \equiv \top$). A similar framework of diagnosis has been proposed by Kurt Konolige; in this theory, called the *default causal net theory*, the partitioning of a logical theory into various meaningful elements is a bit different from that of Theorist. However, there are too many similarities with Theorist to justify an extensive description in this overview.

EXAMPLE 6.8

Figure 6.8 presents a specification of the abductive diagnostic problem from Example 3 in terms of the Theorist implementation, where `Ec` denotes the set of findings assumed to be absent, E^c, taken as constraints in Theorist. The following query:

```
explain thirst and myalgia.
```

yields the following results:

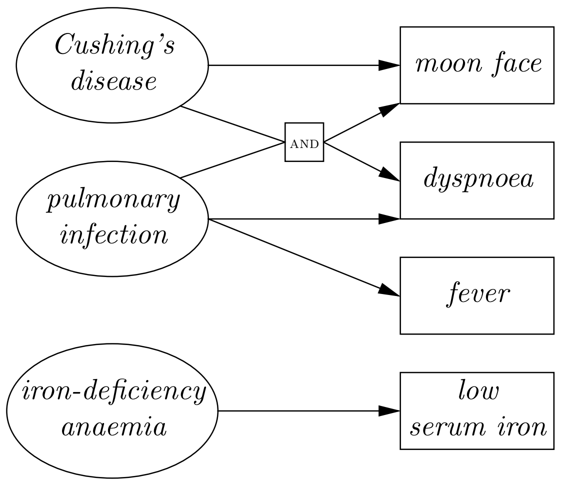


Figure 6.9: Nonmonotonic interaction between disorders.

```
Answer is thirst and myalgia
Theory is [a2,influenza,fever]

Answer is thirst and myalgia
Theory is [sport,fever]

Answer is thirst and myalgia
Theory is [a2,influenza]

Answer is thirst and myalgia
Theory is [sport,influenza]
```

Theories are solutions in the abductive theory of diagnosis. Only a subset of the solutions mentioned in Example 3 are computed, because in Theorist it is assumed that every observed finding need be explained only once by a diagnosis.

Although logic offers powerful tools for designing and studying notions of diagnosis, formalising diagnosis in logical terms may be cumbersome. In particular, when it is necessary to resort to non-standard logics, there is a lurking danger that the original subject of research (diagnosis) is taken over by the study of logic. Instead of taking logic as a language to develop a framework of diagnosis, we might also adopt set theory as our language of choice and generalise the set-covering theory of diagnosis. This approach, which is more straightforward than logical analysis, has been investigated by Peter Lucas. The approach is introduced by the following example.

EXAMPLE 6.9

Consider a medical diagnostic problem, where a patient may have Cushing's disease – a disease caused by a brain tumour producing hyperfunctioning of the adrenal glands – pulmonary infection and iron-deficiency anaemia. We shall not enumerate all symptoms and signs causally associated with these medical problems; it suffices to note that moon face is a sign associated with Cushing's disease, fever and dyspnoea (shortness of breath) are associated with pulmonary infection, and low levels of serum iron are characteristic for iron-deficiency anaemia. However, in a patient in whom Cushing's disease and pulmonary infection coexist there usually is no fever. This indicates that there exists an interaction between the two disorders, Cushing's disease and pulmonary infection, that is nonmonotonic, i.e. the co-occurrence of the two disorders produces fewer findings than the union of their associated observable findings. Figure 6.9 depicts this simple problem. Note that we can neither represent this knowledge by a causal specification (refraining from non-standard logic) as used in abductive diagnosis, nor in terms of an effects function as used in the set-covering theory of diagnosis.

Interactions among defects (disorders) can be expressed by means of a mapping of sets of defects to sets of observable findings. Such a mapping will be called an evidence function. More formally, let $\Sigma = (\Delta, \Phi, e)$ be a *diagnostic specification*, where, again, Δ denotes a set of possible defects (disorders), and Φ denotes a set of observable findings. Positive defects d (findings f) and negative defects $\neg d$ (findings $\neg f$) denote *present* defects (findings) and *absent* defects (findings), respectively. If a defect d or a finding f is not included in a set, it is assumed to be *unknown*. Let a set X_P denote a set of positive elements, and let X_N denote a set of negative elements, such that X_P and X_N are disjoint. It is assumed that $\Delta = \Delta_P \cup \Delta_N$ and $\Phi = \Phi_P \cup \Phi_N$. Now, an *evidence function* e is a mapping

$$e : \wp(\Delta) \to \wp(\Phi) \cup \{\perp\}$$

such that:

(1) for each $f \in \Phi$ there exists a set $D \subseteq \Delta$ with $f \in e(D)$ or $\neg f \in e(D)$ (and possibly both);

(2) if $d, \neg d \in D$ then $e(D) = \perp$;

(3) if $e(D) \neq \perp$ and $D' \subseteq D$ then $e(D') \neq \perp$.

If $e(D) \neq \perp$, it is said that $e(D)$ is the set of *observable findings* for D; otherwise, it is said that D is inconsistent. Inconsistency here means that a particular combination of defects is not allowed. According to the definition above, we may have that both $f \in e(D)$ and $\neg f \in e(D)$, which simply means that these findings may alternatively, e.g. at different times, occur given the combined occurrence of the defects in the set D.

For the medical knowledge depicted in Figure 6.9, it holds, among others, that:

$$\begin{array}{rcl} e(\{\textit{Cushing's disease}\}) & = & \{\textit{moon face}\} \\ e(\{\textit{pulmonary infection}\}) & = & \{\textit{fever}, \textit{dyspnoea}\} \\ e(\{\textit{Cushing's disease}, \textit{pulmonary infection}\}) & = & \{\textit{moon face}, \textit{dyspnoea}\} \end{array}$$

The property

$$\begin{array}{l} e(\{\textit{Cushing's disease}, \textit{pulmonary infection}\}) \not\supseteq e(\{\textit{Cushing's disease}\}) \cup \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad e(\{\textit{pulmonary infection}\}) \end{array}$$

formally expresses that the interaction between *Cushing's disease* and *pulmonary infection* is nonmonotonic.

Various semantic properties of a domain for which a diagnostic system must be built can be expressed precisely as interactions in terms of evidence functions. An example of a local interaction reflected in an evidence function is *causality*; it is formalised as $e(D') \subseteq e(D)$, with the following meaning: 'the set of defects D *causes* the set of defects D''.[5] This is the same sort of knowledge as used in abductive diagnosis (cf. Section 6.3.2). We may also have that defects exhibit no interactions at all, which is a *global* property, expressed as follows:

$$e(D) = \bigcup_{d \in D} e(\{d\})$$

for each consistent set $D \subseteq \Delta$. Observe that this evidence function corresponds to the effects function (6.11) in the set-covering theory of diagnosis. Other semantic properties (with respect to observable findings) can be defined in this fashion quite easily.

To employ an evidence function for the purpose of diagnosis, it must be interpreted with respect to the actually observed findings. The interpretation of an evidence function and the observed findings that is adopted, can be viewed as a notion of diagnosis applied to solve the diagnostic problem at hand.

More formally, let $\mathcal{P} = (\Sigma, E)$ be a *diagnostic problem*, where $\Sigma = (\Delta, \Phi, e)$ and $E \subseteq \Phi$ is a set of *observed findings*. Let R_Σ denote a *notion of diagnosis* R applied to Σ, then a mapping

$$R_{\Sigma, e_{|H}} : \wp(\Phi) \to \wp(\Delta) \cup \{u\}$$

will either provide a diagnostic solution for a diagnostic problem $\mathcal{P}$, or indicate that no solution exists, denoted by u (undefined). Here, H denotes a *hypothesis*, which is taken to be a set of defects ($H \subseteq \Delta$), and $e_{|H}$, called the *restricted evidence function* of e, is a restriction of e with respect to the power set $\wp(H)$:

$$e_{|H} : \wp(H) \to \wp(\Phi) \cup \{\perp\}$$

where for each $D \subseteq H$: $e_{|H}(D) = e(D)$. A restricted evidence function $e_{|H}$ can be thought of as the relevant part of a knowledge base with respect to a hypothesis H. An *R-diagnostic solution*, or *R-diagnosis* for short, with respect to a hypothesis $H \subseteq \Delta$, is now defined as the set

$$R_{\Sigma, e_{|H}}(E), \text{ where } R_{\Sigma, e_{|H}}(E) \subseteq H \text{ if a solution exists.}$$

The general idea is illustrated in Figure 6.10. To illustrate the flexibility of the framework, consider again the notion of *weak* causality as defined in the abductive theory of diagnosis, which is obtained by the addition of assumption literals α to individual abnormality axioms of a causal model $\mathcal{R}$.

EXAMPLE 6.10

[5]We do not claim that this property formalises causality; it only expresses the notion of causality in terms of diagnosis.

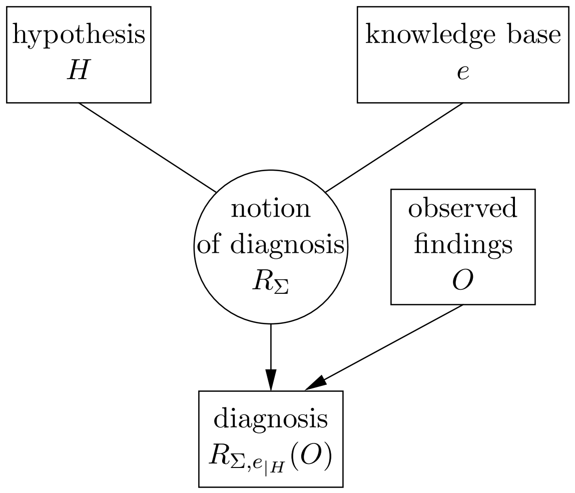

hypothesis H
knowledge base e
notion of diagnosis R_Σ
observed findings O
diagnosis $R_{\Sigma,e_{|H}}(O)$

Figure 6.10: Schema of notion of diagnosis, diagnostic problem and solution.

Consider the abductive problem $\mathcal{P} = (\Sigma, E)$, with causal specification $\Sigma = (\Delta, \Phi, \mathcal{R})$, where $\mathcal{R}$ is equal to:

$$\begin{aligned} \mathit{fever} \wedge \alpha_1 &\rightarrow \mathit{thirst} \\ \mathit{fever} \wedge \alpha_2 &\rightarrow \mathit{sweating} \\ \mathit{pneumonia} \wedge \alpha_3 &\rightarrow \mathit{fever} \\ \mathit{pulmonary\_embolism} \wedge \alpha_4 &\rightarrow \mathit{dyspnoea} \end{aligned}$$

and where *fever*, *pneumonia*, and *pulmonary_embolism* are defects (disorders). The resulting evidence function e is defined by the following restriction $\tilde{e}$ of the evidence function e:

$$\tilde{e}(D) = \begin{cases} \{\mathit{thirst}, \mathit{sweating}\} & \text{if } D = \{\mathit{fever}\} \\ \{\mathit{thirst}, \mathit{sweating}\} & \text{if } D = \{\mathit{pneumonia}\} \\ \{\mathit{dyspnoea}\} & \text{if } D = \{\mathit{pulmonary\_embolism}\} \\ \bot & \text{if } D = \{\neg \mathit{fever}, \mathit{pneumonia}\} \\ \varnothing & \text{if } D \text{ is a singleton set different from those above} \end{cases}$$

yielding a diagnostic specification $\Sigma = (\Delta, \Phi, e)$, where the function e is obtained from $\tilde{e}$ by taking the union of non-specified, consistent function values. For example,

$$e(\{\mathit{pneumonia}, \mathit{pulmonary\_embolism}\}) = \{\mathit{thirst}, \mathit{sweating}, \mathit{dyspnoea}\}$$

Given the definition of a diagnostic problem $\mathcal{P}$, it is possible to solve it using various notions of diagnosis. For example, the notion of diagnosis that corresponds to abductive diagnosis with weakly causal relations as introduced above, is called the notion of *weak-causality diagnosis*, denoted by WC. It is defined as follows:

$$\mathrm{WC}_{\Sigma,e_{|H}}(E) = \begin{cases} H & \text{if } e_{|H}(H) \supseteq E \\ u & \text{otherwise} \end{cases}$$

This notion of diagnosis is precisely the same as set-covering diagnosis, except that it is defined for general evidence functions, and not only for those evidence functions that are free of interaction.

EXAMPLE 6.11

Reconsider the previous example. Let the set of observed findings be equal to $E = \{\mathit{thirst}, \mathit{sweating}\}$, then the set $H = \{\mathit{fever}, \alpha_1, \alpha_2\}$ is an abductive solution to $\mathcal{P} = (\Sigma, E)$, because the covering and consistency conditions are satisfied; the associated diagnosis is $D = \{\mathit{fever}\}$. In terms of the set-theoretical framework, we have

$$\mathrm{WC}_{\Sigma, e_{|\{\mathit{fever}\}}}(E) = \{\mathit{fever}\}$$

Hence, the results of the (set-theoretical) notion of weak-causality diagnosis and the (logical) notion of abductive diagnosis with a weakly causal model $\mathcal{R}$ do indeed coincide.

Other notions of diagnosis, such as consistency-based diagnosis or a notion of diagnosis based on strongly causal knowledge, can be defined in a straightforward way. For example, where the notion of strong causality diagnosis is obtained in the theory of abductive diagnosis by doing away with incompleteness assumption literals, the same notion is obtained in the set-theoretical framework by replacing the $\supseteq$ relation in the definition of the function WC by equality $=$. The resulting notion of diagnosis expresses that all predicted observable findings must be observed, and vice versa.

It is also straightforward to define notions of diagnosis in terms of the set-theoretical framework that offer some approximating or refinement form of diagnosis. For example, the following notion of diagnosis, called *most general subset diagnosis*,

$$\mathrm{GS}_{\Sigma, e_{|H}}(E) = \begin{cases} \bigcup_{\substack{H' \subseteq H \\ e_{|H}(H') \subseteq E}} H' & \text{if } H \text{ is consistent, and} \\ & \quad \exists H' \subseteq H : e_{|H}(H') \subseteq E \\ u & \text{otherwise} \end{cases}$$

is more flexible than strong-causality diagnosis. Intuitively, a most general subset diagnosis is the smallest set of defects that includes all accepted subhypotheses of a given hypothesis, where an accepted subhypothesis concerns observable findings that all have been observed.

EXAMPLE 6.12

Reconsider Example 10. Let $E = \{\mathit{thirst}, \mathit{sweating}, \mathit{dyspnoea}\}$ be the set of observed findings. Then, we have that

$$\mathrm{WC}_{\Sigma, e_{|\{\mathit{fever}, \mathit{pneumonia}\}}}(E) = u$$

i.e. the observed findings in E cannot be accounted for using weak-causality diagnosis. However, it holds that

$$\mathrm{GS}_{\Sigma, e_{|\{\mathit{fever}, \mathit{pneumonia}\}}}(E) = \{\mathit{fever}, \mathit{pneumonia}\}$$

This expresses that at least part of the observed findings in E can be accounted for by the hypothesis $\{\mathit{fever}, \mathit{pneumonia}\}$.

Hypothetico-deductive diagnosis can be described using the set-theoretical framework as a specific form of most general subset diagnosis. Assuming for simplicity's sake that the associated evidence function exhibits no interaction, most general subset diagnosis expresses that a defect is accepted as part of a diagnosis if all its associated typical observable findings have been observed. With some slight extensions, it is also possible to model the effect of grouping various findings with respect to a defect, which is usually expressed in rule-based systems by defining more than one rule with the same conclusion.

Exercises

(6.1) Answer the following general questions concerning model-based reasoning:

a. Give a description of a problem for which DNSB diagnosis is the only applicable form of diagnosis. Explain your answer.

b. Why is the logical implication $\rightarrow$ often used in abductive diagnosis for the formalisation of the notion of causality? Which properties of logical implication closely fit the intuitive meaning of the notion of causality and which properties of logical implication are intuitively incompatible?

(6.2) The following questions concern the theory of abductive diagnosis:

a. Consider the causal specification $\Sigma = (\Delta, \Phi, \mathcal{R})$, where

- $\Delta = \{d_1, d_2, d_3, \alpha_1, \alpha_2\}$ denotes a set of defects (d_1, d_2, d_3) and assumption literals (α_1, α_2);
- $\Phi = \{f_1, f_2, f_3\}$ denotes a set of observable findings;
- $\mathcal{R} = \{d_1 \wedge \alpha_1 \rightarrow d_2,$
  $d_1 \rightarrow f_1,$
  $d_2 \wedge \alpha_2 \rightarrow f_2,$
  $d_2 \wedge d_3 \rightarrow f_3\}$
  denotes a model of abnormal behaviour.

Now, let $\mathcal{P} = (\Sigma, E)$ be a diagnostic problem, where $E = \{f_1, f_3\}$ is a set of observed findings.

Determine all abductive solutions and diagnoses for $\mathcal{P}$. Next, determine the solution formula following from the computation of the predicate completion of $\mathcal{R}$ and the set of observed findings E. Assume in this case that all elements in Δ, with the exception of d_2, are abducible. Finally, discuss the logical relationship between the solution formula and the abductive solutions for $\mathcal{P}$.

b. Suppose that DPF and DPF$'$ are diagnostic problem formulations in the spectrum of logical definitions of diagnosis with the same diagnostic problem $\mathcal{P}$, such that DPF $\sqsubset$ DPF$'$. This means that DPF 'precedes' DPF$'$. Now, prove that the set of diagnoses for DPF$'$ is a subset of the set of diagnoses for DPF when applying abductive diagnosis.

(6.3) Abductive diagnosis exercises:

a. Consider the abductive network $A = \langle D, M, C\rangle$ where

   - $D = \{d_1, d_2, d_3, d_4\}$ denotes a set of disorders;
   - $M = \{m_1, m_2, m_3, m_4\}$ denotes a set of manifestations;
   - $C = \{(d_1, m_1),$
     $(d_1, m_2),$
     $(d_2, m_2),$
     $(d_2, m_4),$
     $(d_3, m_2),$
     $(d_3, m_3),$
     $(d_4, m_3),$
     $(d_4, m_4)\}$
     denotes a causation relation.

   Now, let $P = \langle D, M, C, M^+\rangle$ be a diagnostic problem, where $M^+ = \{m_1, m_3\}$ is a set of observed manifestations.

   Determine all diagnoses given the irredundancy parsimony criterion for P using the covering algorithm of Peng and Reggia. Clearly indicate the various different steps taken by the algorithm.

b. Let $A = \langle D, M, C\rangle$ be an abductive network and let $P = \langle D, M, C, M^+\rangle$ be a diagnostic problem. Give an abductive network $A' = \langle D', M', C'\rangle$ with a minimal number of disorders and manifestations such that $\mathit{Sol}(P, \mathit{irredundancy}) = \mathit{Sol}(P', \mathit{irredundancy})$ where $P' = \langle D', M', C', M^+\rangle$.

(6.4) Consistency-based diagnosis and GDE:

a. Consider the diagnostic problem DP = (SYS, OBS) in the theory of consistency-based diagnosis (according to Reiter), where

   - SYS = (SD, COMPS) denotes a system specification, with
     * $\mathrm{SD} = \{\forall x((\mathrm{ANDG}(x) \wedge \neg \mathrm{AB}(x)) \rightarrow (\mathit{and}(\mathit{in}(1,x), \mathit{in}(2,x)) = \mathit{out}(x))),$
       $\forall x((\mathrm{ORG}(x) \wedge \neg \mathrm{AB}(x)) \rightarrow (\mathit{or}(\mathit{in}(1,x), \mathit{in}(2,x)) = \mathit{out}(x))),$
       $\mathit{in}(2, O_1) = \mathit{in}(2, O_2),$
       $\mathit{in}(2, A_2) = \mathit{in}(2, A_1),$
       $\mathit{out}(O_1) = \mathit{in}(1, A_1),$
       $\mathit{out}(O_1) = \mathit{in}(1, A_2),$
       $\mathit{out}(A_2) = \mathit{in}(1, O_2),$
       $\mathrm{ORG}(O_1), \mathrm{ORG}(O_2), \mathrm{ANDG}(A_1), \mathrm{ANDG}(A_2)\};$
     * $\mathrm{COMPS} = \{A_1, A_2, O_1, O_2\};$
   - $\mathrm{OBS} = \{\mathit{in}(1, O_1) = 1, \mathit{in}(2, O_1) = 0, \mathit{in}(2, A_2) = 1, \mathit{out}(A_1) = 0, \mathit{out}(O_2) = 0\}$ denotes a set of observations.

   Determine the set of all conflict sets for DP; indicate for each conflict set whether or not it is minimal. Use the computed conflict sets to compute the set of diagnoses for DP by means of the HS-DAG algorithm.

b. For the original hitting-set algorithm of Reiter, various different optimisations have been proposed with the aim of reducing the number of nodes of the generated hitting-set tree. Have one or more of these optimisations been realised in the algorithms of the General Diagnostic Engine (GDE) ? If so, in which parts of

GDE have the optimisations been realised; if not, in which parts of GDE may the optimisations be incorporated ?