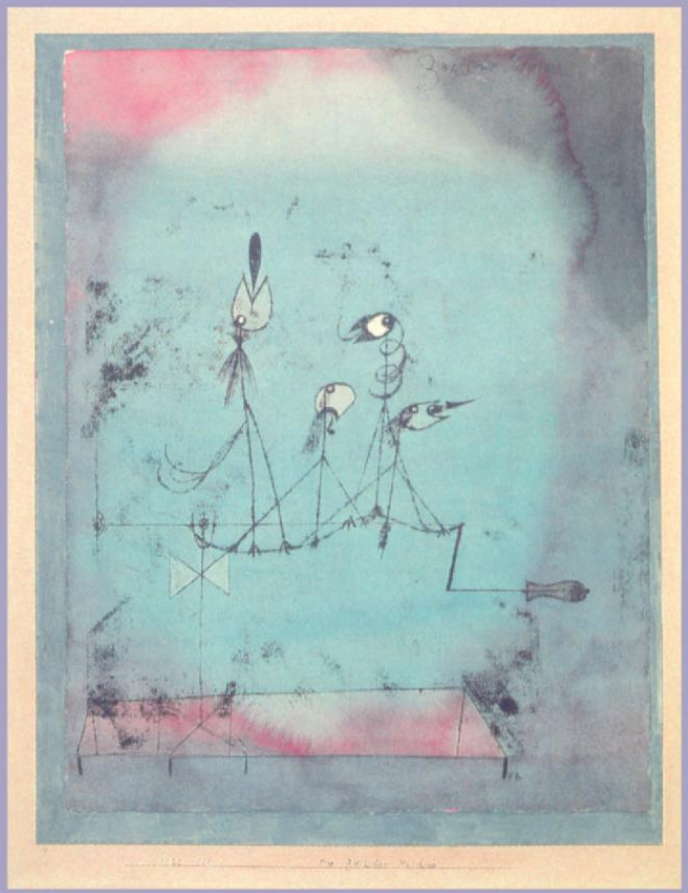
LOGICAL FOUNDATIONS of ARTIFICIAL INTELLIGENCE



MICHAEL R. GENESERETH
NILS J. NILSSON

Logical Foundations of Artificial Intelligence

Michael R. Genesereth and Nils J. Nilsson
Stanford University

Morgan Kaufmann Publishers, Inc.

Editor and President *Michael B. Morgan*
Production Manager *Jennifer M. Ballentine*
Cover Designer *Irene Imfeld*
Composition *Arthur Ogawa and Laura Friedsam*
Book Design *Beverly Kennon-Kelley*
Copy Editor *Lyn Dupré*
Graduate Assistant *Jamison Gray*
Production Assistant *Todd R. Armstrong*

Library of Congress Cataloging-in-Publication Data

```
Genesereth, Michael, 1948-
   Logical foundations of artificial intelligence.

   Bibliography: p.
   Includes index.
   1. Artificial intelligence.    I. Nilsson, Nils,
1933-     .    II. Title.
Q335.G37  1986         006.3         87-5461
ISBN 0-934613-31-1
```

Reprinted with corrections May, 1988

Morgan Kaufmann Publishers, Inc.
P.O. Box 50490, Palo Alto, CA 94303

Printed in the United States of America

ISBN: 0-934613-31-1

90 89 88 5 4 3 2

Contents

This page intentionally left blank

Typographical Conventions

(1) Elements of a conceptualization—objects, functions, and relations—are written in italics, as in the following example:

The extension of the *on* relation is the set $\{\langle a, b\rangle, \langle b, c\rangle, \langle d, e\rangle\}$.

(2) Expressions and subexpressions in predicate calculus are written in boldface, "typewriter" font, as in:

$$\texttt{(}\forall\texttt{x Apple(x))} \vee \texttt{(}\exists\texttt{x Pear(x))}$$

(3) We use lowercase Greek letters as metavariables ranging over predicate-calculus expressions and subexpressions. The use of these variables is sometimes mixed with actual predicate-calculus expressions, as in the following:

$$(\phi(\alpha) \vee \texttt{P(A)} \Rightarrow \psi)$$

Sometimes, for mnemonic clarity, we use Roman characters, in mathematical font, as metavariables for relation constants and object constants, as in the following sample text:

Suppose we have a relation constant P and an object constant A such that $P(A) \Rightarrow \texttt{P} \wedge \texttt{Q(B)}$.

(4) We use uppercase Greek letters to denote sets of predicate calculus formulas, as in:

> If there exists a proof of a sentence ϕ from a set Δ of premises and the logical axioms using Modus Ponens, then ϕ is said to be *provable* from Δ (written as $\Delta \vdash \phi$).

Since clauses are sets of literals, we also use uppercase Greek letters as variables ranging over clauses, as in:

> Suppose that Φ and Ψ are two clauses that have been standardized apart.

(5) We use ordinary mathematical (not typewriter) font for writing metalogical formulas *about* predicate-calculus statements, as in:

> If σ is an object constant, then $\sigma^I \epsilon |I|$.

Sometimes, metalogical formulas might contain predicate-calculus expressions:

$$\mathtt{A}^I = a$$

(6) We use an uppercase script $\mathcal{T}$ to denote a predicate-calculus "theory."

(7) Algorithms and programs are stated in typewriter font:

```
Procedure Resclution (Gamma)
    Repeat Termination(Gamma) ==> Return(Success),
           Phi <- Choose(Gamma), Psi <- Choose(Gamma),
           Chi <- Choose(Resolvents(Phi,Psi)),
           Gamma <- Concatenate(Gamma,[Chi])
    End
```

(8) We use the notation $\{\mathtt{x/A}\}$ to denote the substitution in which the object constant `A` is substituted for the variable `x`. We use lowercase Greek letters for variables ranging over substitutions, as in:

> Consider the combination of substitutions $\sigma\rho$.

(9) Lowercase ps and qs are used to denote probabilities:

$$p(\mathtt{P} \wedge \mathtt{Q})$$

(10) Sets of possible worlds are denoted by uppercase script letters, such as $\mathcal{W}$.

(11) Vectors and matrices are denoted by boldface capital letters, such as $\mathbf{V}$ and $\mathbf{P}$.

(12) We also use boldface capital letters (and sequences of capital letters) to denote modal operators, such as $\mathbf{B}$ and $\mathbf{K}$.

CHAPTER 1
Introduction

ARTIFICIAL INTELLIGENCE (AI) is the study of intelligent behavior. Its ultimate goal is a theory of intelligence that accounts for the behavior of naturally occurring intelligent entities and that guides the creation of artificial entities capable of intelligent behavior. Thus, AI is both a branch of science and a branch of engineering.

As *engineering*, AI is concerned with the concepts, theory, and practice of building intelligent machines. Examples of machines already within the reach of AI include *expert systems* that give advice about specialized subjects (such as medicine, mineral exploration, and finance), question-answering systems for answering queries posed in restricted but large subsets of English and other natural languages, and theorem-proving systems for verifying that computer programs and digital hardware meet stated specifications. Ahead lie more flexible and capable robots, computers that can converse naturally with people, and machines capable of performing much of the world's "knowledge work."

As *science*, AI is developing concepts and vocabulary to help us to understand intelligent behavior in people and in other animals. Although there are necessary and important contributions to this same scientific goal by psychologists and by neuroscientists, we agree with the statement made by the sixteenth-century Italian philosopher Vico: *Certum quod factum* (one is certain of only what one builds). Aerodynamics, for

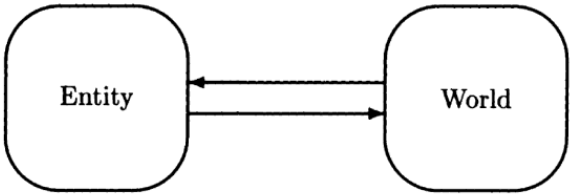


Figure 1.1 Entity and environment.

example, matured as it did because of its concern for flying machines; then it also helped us to explain and understand flight in animals. Thus, notwithstanding its engineering orientation, an ultimate goal of AI is a comprehensive theory of intelligence as it occurs in animals as well as in machines.

Note that in talking about the behavior of an intelligent entity in its environment, we have implicitly divided the world into two parts. We have placed an envelope around the entity, separating it from its environment, and we have chosen to focus on the transactions across that envelope. (See Figure 1.1.) Of course, a theory of intelligence must not only describe these transactions but must also give a clear picture of the structure of the entity responsible for those transactions. An important concept in this regard is that of *knowledge*. Intelligent entities seem to anticipate their environments and the consequences of their actions. They act as if they know, in some sense, what the results would be. We can account for this anticipatory behavior by assuming that intelligent entities themselves possess knowledge of their environments.

What more can we say about such knowledge? What forms can it take? What are its limits? How do entities use knowledge? How is knowledge acquired? Unfortunately, we cannot say much to answer these questions insofar as they pertain to natural, biological organisms. Even though we are beginning to learn how neurons process simple signals, our understanding of how animal brains—which are composed of neurons—represent and process knowledge about the world is regretfully deficient.

The situation is rather different when we turn our attention to artifacts, such as computer systems, capable of rudimentary intelligent behavior. Although we have not yet built machines approaching human levels of intelligence, nevertheless we can talk about how such machines can be said to possess knowledge. Because we design and build these machines, we ought to be able to decide what it means for them to *know* about their environments.

There are two major ways we can think about a machine having knowledge about its world. Although our ideas about the distinction

between these two points of view are still being clarified, it seems that, in some of our machines, the knowledge is implicit; in other machines, it is represented explicitly.

We would be inclined to say, for example, that the mathematical knowledge built into a computer program for inverting matrices is implicit knowledge, "stored," as it is, in the sequence of operations performed by the program. Knowledge represented in this way is manifest in the actual running or execution of the matrix-inverting program. It would be difficult to extract it from the text of the computer code itself for other uses. Computer scientists have come to call knowledge represented in this way *procedural knowledge*, because it is inextricably contained in the very procedures that use it.

On the other hand, consider a tabular database of salary data. In this case, we would be inclined to say that the knowledge is explicit. Programs designed to represent their knowledge explicitly have turned out to be more versatile in performing the complex tasks that we usually think of as requiring intelligence. Particularly useful explicit representations of knowledge are those that can be interpreted as making declarative statements. We call knowledge represented in this way *declarative knowledge* because it is contained in declarations about the world. Such statements typically are stored in symbol structures that are accessed by the procedures that use this knowledge.

There are several reasons to prefer declaratively represented knowledge when designing intelligent machines. One advantage is that such knowledge can be changed more easily. To make a small change to a machine's declarative knowledge, usually we need to change just a few statements. Even small adjustments to procedural knowledge, on the other hand, may require extensive changes to the program. Knowledge represented declaratively can be used for several different purposes, even purposes not explicitly anticipated at the time the knowledge is assembled. The knowledge base itself does not have to be repeated for each application, nor does it have to be specifically designed for each application. Declarative knowledge often can be extended, beyond that explicitly represented, by *reasoning* processes that derive additional knowledge. Finally, declarative knowledge can be accessed by *introspective* programs, so that a machine can answer questions (for itself or for others) about what it knows. A price is paid for these advantages, however. Using declarative knowledge usually is more costly and slower than is directly applying procedural knowledge. We give up efficiency to gain flexibility.

It is tempting to speculate about the roles of these two kinds of knowledge in biological organisms. Many insects and other not-very-brainy creatures seem so well attuned to their environments that it is difficult to avoid saying that they have a great deal of knowledge about their worlds. A spider, for example, must use quite a bit of knowledge about materials and structures in spinning a web. Once we understand such creatures

better, it seems likely that we will conclude that the knowledge they have evolved about their special niches is procedural. On the other hand, when a human mechanical engineer is consciously thinking about a new bridge design, it seems likely that he refers to declaratively represented knowledge about materials and structures. Admittedly, humans often (perhaps even usually) use procedural knowledge also. The tennis knowledge *used* by a champion player seems procedural, whereas that *taught* by an excellent teacher seems declarative. Perhaps when the distinctions between declarative and procedural knowledge are more clearly understood by computer scientists, they will indeed help biologists and psychologists characterize the knowledge of animals.

In any case, intelligent machines will need both procedural and declarative knowledge. Thus, it is difficult to see how we can study them properly without involving *all* of computer science. The most flexible kinds of intelligence, however, seem to depend strongly on declarative knowledge, and AI has concerned itself more and more with that subject. Our emphasis on declarative knowledge in this book should not be taken to imply that we think procedural knowledge unimportant. For example, when declarative knowledge is used over and over again for the same specific purpose, it would be advisable to compile it into a procedure tailored for that purpose. Nevertheless, the study of representing and using declarative knowledge is such a large and important subject in itself that it deserves book-length treatment.

The book is divided roughly into four parts. In the first five chapters, we present the main features of what is commonly called the *logicist* approach to AI. We begin by describing *conceptualizations* of the subject matter about which we want our intelligent systems to have knowledge. Then we present the syntax and semantics of the *first-order predicate calculus*, a declarative language in which we can write sentences about these conceptualizations. We then formalize the process of inference. Finally, we discuss a simple but powerful inference procedure called *resolution*, and we show how it can be used in reasoning systems.

In the next three chapters, we broaden the logical approach in various ways to deal with several inadequacies of strict logical deduction. First, we describe methods that allow *nonmonotonic reasoning*; i.e., reasoning in which tentative conclusions can be derived. Next, we discuss extensions that permit systems to learn new facts. Then, we show how to represent and reason with knowledge that is not certain.

In the next two chapters, we expand our language and its semantics by introducing new constructs, called *modal operators*, that facilitate representing and reasoning about knowledge about what other agents know or believe. Then, we show how the whole process of writing predicate-calculus sentences to capture conceptualizations can be turned in on itself at the *metalevel* to permit sentences about sentences and about reasoning processes.

In the final three chapters, we concern ourselves with agents that can perceive and act in the world. We first discuss the representation of knowledge about states and actions. Then, we show how this knowledge can be used to derive plans to achieve goals. Finally, we present a framework that allows us to relate sensory knowledge and inferred knowledge and that allows us to say how this knowledge affects an intelligent agent's choice of actions.

1.1 Bibliographical and Historical Remarks

The quest to build machines that think like people has a long tradition. Gardner [Gardner 1982] attributed to Leibniz the dream of "a universal algebra by which all knowledge, including moral and metaphysical truths, can some day be brought within a single deductive system." Frege, one of the founders of modern symbolic logic, proposed a notational system for mechanical reasoning [Frege 1879]. When digital computers were first being developed in the 1940s and 1950s, several researchers wrote programs that could perform elementary reasoning tasks, such as proving mathematical theorems, answering simple questions, and playing board games such as chess and checkers. In 1956, several of these researchers attended a workshop on AI at Dartmouth College, organized by McCarthy (who, incidentally, suggested the name *Artificial Intelligence* for the field) [McCorduck 1979]. (McCorduck's book is an interesting, nontechnical history of early AI work and workers.) Many of the important first papers about AI are contained in the collection *Computers and Thought* [Feigenbaum 1963].

From AI's very beginnings, people have pursued many approaches to the discipline. One, based on building parallel machines that could *learn* to recognize patterns, occupied many AI researchers during the 1960s and continues as one strand of what has come to be called *connectionism.* See [Nilsson 1965] for an example of some of the early work using this approach, and [Rumelhart 1986] for a collection of connectionist papers.

The computational manipulation of arbitrary symbolic structures (as opposed to operations on numbers) is at the heart of much work in AI. The idea that symbol manipulation is a sufficient process for explaining intelligence was forcefully stated in the *physical symbol system hypothesis* of Newell and Simon [Newell 1976]. The need for manipulating symbols led to the development of special computer languages. LISP, invented by McCarthy in the late 1950s [McCarthy 1960], continues to be the most popular of these languages. PROLOG [Colmerauer 1973, Warren 1977], stemming from ideas proposed by Green [Green 1969a], Hayes [Hayes 1973b], and Kowalski [Kowalski 1974, Kowalski 1979a] is rapidly gaining adherents. Much of the work in AI still is characterized mainly by the use of sophisticated symbol manipulation to perform complex reasoning tasks.

One articulation of the symbol-manipulating approach uses *production systems*, a term that has been used rather loosely in AI. Production systems derive from a computational formalism proposed by Post [Post 1943] based on string-replacement rules. The closely related idea of a *Markov algorithm* [Markov 1954, Galler 1970] involves imposing an order on the replacement rules and using this order to decide which applicable rule to apply next. Newell and Simon [Newell 1972, Newell 1973] used string-modifying production rules, with a simple control strategy, to model certain types of human problem-solving behavior. An earlier textbook by Nilsson [Nilsson 1980] used production systems as an organizing theme. More recently, the OPS family of symbol-manipulating computer-programming languages has been based on production rules [Forgy 1981, Brownston 1985]. Work on SOAR by Laird, Newell, and Rosenbloom [Laird 1987] and on *blackboard systems* by a variety of researchers [Erman 1982, Hayes-Roth 1985] can be regarded as following the production system approach.

Another important aspect of AI is *heuristic search*. Search methods are described as a control strategy for production systems in [Nilsson 1980]. Pearl's book [Pearl 1984] gave a thorough mathematical treatment of heuristic search, and his review article summarized the subject [Pearl 1987]. Lenat's work [Lenat 1982, Lenat 1983a, Lenat 1983b] on the nature of heuristics resulted in systems that exploit general heuristic properties in specific problems.

The view taken toward AI in this book follows the theme hinted at by Leibniz and Frege and then substantially elaborated and developed into specific proposals by McCarthy [McCarthy 1958 (the *advice taker* paper), McCarthy 1963]. It is based on two related ideas. First, the knowledge needed by intelligent programs can be expressed as declarative sentences in a form that is more or less independent of the uses to which that knowledge might later be put. Second, the reasoning performed by intelligent programs involves logical operations on these sentences. Good accounts of the importance of logic in AI, for representation and for reasoning, have been written by Hayes [Hayes 1977], Israel [Israel 1983], Moore [Moore 1982, Moore 1986], and Levesque [Levesque 1986].

Several people, however, have argued that logic has severe limitations as a foundation for AI. McDermott's article contained several cogent criticisms of logic [McDermott 1987a], whereas Simon emphasized the role of search in AI [Simon 1983]. Many AI researchers have stressed the importance of specialized procedures and of procedural (as opposed to declarative) representations of knowledge (see, for example, [Winograd 1975, Winograd 1980]). Minsky has claimed that intelligence in humans is the result of the interaction of a very large and complex assembly of loosely connected subunits operating much like a *society* but within a single individual [Minsky 1986].

Notwithstanding the various criticisms of logic, there does seem to be an emerging consensus among researchers that logical tools are important, at

the very least, for helping us to analyze and understand AI systems. Newell [Newell 1982] made that point in his article about the *knowledge level.* The work of Rosenschein and Kaelbling on *situated automata* is a good example of an approach to AI that acknowledges the analytic utility of logic while pursuing an alternative implementational strategy [Rosenschein 1986]. The assertion that predicate calculus and logical operations can also usefully serve directly in the implementation of AI systems as a representation language and as reasoning processes, respectively, is a much stronger claim.

Several thinkers have claimed that none of the techniques currently being explored will ever achieve true, human-level intelligence. Prominent among these are the Dreyfuses, who argued that symbol manipulation operations are not the foundation of intelligence [Dreyfus 1972, Dreyfus 1981, Dreyfus 1986] (although their suggestions about what might be needed seem compatible with the claims of the connectionists). Winograd and Flores argued, mainly, that whatever mechanistic processes are involved in thinking, they are probably too complicated to be fully expressed in artificial machines designed and built by human engineers [Winograd 1986]. Searle attempted to distinguish between *real* thought and mere *simulations* of thought by rule-like computations [Searle 1980]. He also seemed to claim that computer-like machines built of silicon, for example, will not do, although machines built according to different principles out of protein might. Taking a somewhat different tack, Weizenbaum argued that, even if we could build intelligent machines to perform many human functions, it might be unethical to do so [Weizenbaum 1976].

There are several other good AI textbooks. Most of them differ from this one in that they do not emphasize logic as much as we do, and they describe applications of AI such as natural-language processing, expert systems, and vision. The books by Charniak and McDermott, Winston, and Rich are three such texts [Charniak 1984, Winston 1977, Rich 1983]. The book by Boden [Boden 1977] treats some of the philosophical issues related to AI. In addition to these books, the reader might also refer to encyclopedic collections of short articles about key ideas in AI [Shapiro 1987, Barr 1982, Cohen 1982].

Many important articles describing AI research appear in the journal *Artificial Intelligence.* In addition, there are several other relevant journals, including the *Journal of Automated Reasoning, Machine Learning,* and *Cognitive Science.* Several articles are reprinted in special collections. The American Association for Artificial Intelligence and other national organizations hold annual conferences with published proceedings [AAAI 1980]. The International Joint Council on Artificial Intelligence holds biannual conferences with published proceedings [IJCAI 1969]. Technical notes and memoranda published by the several university and industrial laboratories performing research in AI are available in microfilm from Scientific DataLink (a division of Comtex Scientific Corporation) in New York.

For an interesting summary of the opinions of several AI reseachers about the status of the field during the mid-1980s see [Bobrow 1985]. Trappl's book contains articles about the social implications of AI [Trappl 1986].

Exercises

1. *Structure and behavior.* It is common in discussing the design of artifacts to distinguish between the structure of a device (i.e., its parts and their interconnections) and its behavior (i.e., its external effects).

   a. Give a brief description of a thermostat. Describe both its external behavior and its internal structure. Explain how its structure achieves its behavior.

   b. Is it possible to determine the purpose of an artifact unambiguously, given its behavior? Provide examples to justify your answer.

   c. In his paper "Ascribing Mental Qualities to Machines," John McCarthy[McCarthy 1979b] suggests that it is convenient to talk about artifacts (such as thermostats and computers) as having mental qualities (such as beliefs and desires). For example, according to McCarthy, a thermostat *believes* it is too hot, too cold, or just right, and *desires* that it be just right. Try to adopt McCarthy's viewpoint and indicate the beliefs and desires you think an alarm clock possesses.

2. *Missionaries and cannibals.* Three missionaries and three cannibals seek to cross a river. A boat is available that can hold two people and can be navigated by any combination of missionaries and cannibals involving one or two people. If at any time the missionaries on either bank of the river or en route on the river are outnumbered by cannibals, the cannibals will indulge their anthropophagic tendencies and do away with the missionaries.

   a. Find the simplest schedule of crossings that will permit all the missionaries and cannibals to cross the river safely.

   b. State at least three facts about the world you used in solving the problem. For example, you had to know that a person can be in only one place at a time.

   c. Describe the steps that you took to solve the problem. For each step, record the facts or assumptions you used and the conclusions you drew. The purpose of this part of the problem is to get you to think about the process of solving a problem, not just to arrive at the final solution. Do just enough to get a feel for this distinction.

CHAPTER 2
Declarative Knowledge

AS WE HAVE ALREADY ARGUED, intelligent behavior depends on the knowledge an entity has about its environment. Much of this knowledge is descriptive and can be expressed in *declarative* form. The goal of this chapter is to elucidate the issues involved in formally expressing declarative knowledge.

Our approach to formalizing knowledge is much the same as that of scientists who describe the physical world; in fact, our language is similar to that used to state results in mathematics and the natural sciences. The difference is that in this book we are concerned with the issues of formalizing knowledge, rather than with discovering the knowledge to be formalized.

2.1 Conceptualization

The formalization of knowledge in declarative form begins with a *conceptualization.* This includes the objects presumed or hypothesized to exist in the world and their interrelationships.

The notion of an *object* used here is quite broad. Objects can be concrete (e.g., this book, Confucius, the sun) or abstract (e.g., the number 2, the set of all integers, the concept of justice). Objects can be primitive or composite (e.g., a circuit that consists of many subcircuits). Objects can even be fictional (e.g., a unicorn, Sherlock Holmes, Miss Right). In short, an object can be anything about which we want to say something.

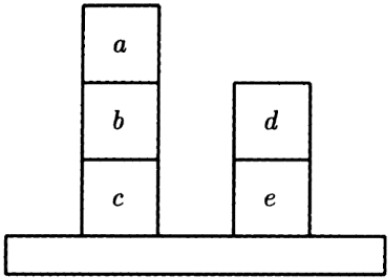


Figure 2.1 A scene from the Blocks World.

Not all knowledge-representation tasks require that we consider all the objects in the world; in some cases, only those objects in a particular set are relevant. For example, number theorists usually are concerned with the properties of numbers and usually are not concerned with physical objects such as resistors and transistors. Electrical engineers usually are concerned with resistors and transistors and usually are not concerned with buildings and bridges. The set of objects about which knowledge is being expressed is often called a *universe of discourse.*

As an example, consider the Blocks World scene in Figure 2.1. Most people looking at this figure interpret it as a configuration of toy blocks. Some people conceptualize the table on which the blocks are resting as an object as well; but, for simplicity, we ignore it here.

The universe of discourse corresponding to this conceptualization is the set consisting of the five blocks in the scene.

$$\{a, b, c, d, e\}$$

Although in this example there are finitely many elements in our universe of discourse, this need not always be the case. It is common in mathematics, for example, to consider the set of all integers, or the set of all real numbers, or the set of all n-tuples of real numbers, as universes with infinitely many elements.

A *function* is one kind of interrelationship among the objects in a universe of discourse. Although we can define many functions for a given set of objects, in conceptualizing a portion of the world we usually emphasize some functions and ignore others. The set of functions emphasized in a conceptualization is called the *functional basis set.*

For example, in thinking about the Blocks World, it would make sense to conceptualize the partial function *hat* that maps a block into the block

on top of it, if any such block exists. The tuples corresponding to this partial function are as follows:

$$\{\langle b, a\rangle, \langle c, b\rangle, \langle e, d\rangle\}$$

When concentrating on spatial relationships, we would probably ignore functions that do not have any spatial significance, such as the *rotate* function that maps blocks into blocks according to the alphabetic order of their labels.

$$\{\langle a, b\rangle, \langle b, c\rangle, \langle c, d\rangle, \langle d, e\rangle, \langle e, a\rangle\}$$

A *relation* is the second kind of interrelationship among objects in a universe of discourse. As we do with functions, in conceptualizing a portion of the world, we emphasize some relations and ignore others. The set of relations in a conceptualization is called the *relational basis set.*

In a spatial conceptualization of the Blocks World, there are numerous meaningful relations. For example, it makes sense to think about the *on* relation that holds between two blocks if and only if one is immediately above the other. For the scene in Figure 2.1, *on* is defined by the following set of tuples.

$$\{\langle a, b\rangle, \langle b, c\rangle, \langle d, e\rangle\}$$

We might also think about the *above* relation that holds between two blocks if and only if one is anywhere above the other.

$$\{\langle a, b\rangle, \langle b, c\rangle, \langle a, c\rangle, \langle d, e\rangle\}$$

The *clear* relation holds of a block if and only if there is no block on top of it. For the scene in Figure 2.1, this relation has the following elements.

$$\{a, d\}$$

The *table* relation holds of a block if and only if that block is resting on the table.

$$\{c, e\}$$

The generality of relations can be determined by comparing their elements. Thus, the *on* relation is less general than the *above* relation since, when viewed as a set of tuples, it is a subset of the *above* relation. Of course, some relations are empty (e.g., the *unsupported* relation), whereas others consist of all n-tuples over the universe of discourse (e.g., the *block* relation).

It is worthwhile to note that, for a finite universe of discourse, there is an upper bound on the number of possible n-ary relations. In particular, for a universe of discourse of size b, there are b^n distinct n-tuples. Every n-ary relation is a subset of these b^n tuples. Therefore, an n-ary relation must be one of at most $2^{(b^n)}$ possible sets.

Formally, a *conceptualization* is a triple consisting of a universe of discourse, a functional basis set for that universe of discourse, and a relational basis set. For example, the following triple is one conceptualization of the world in Figure 2.1.

$$\langle \{a, b, c, d, e\}, \{hat\}, \{on, above, clear, table\} \rangle$$

Note that, although we have have written the *names* of objects, functions, and relations here, the conceptualization consists of the objects, functions, and relations themselves.

No matter how we choose to conceptualize the world, it is important to realize that there are other conceptualizations as well. Furthermore, there need not be any correspondence between the objects, functions, and relations in one conceptualization and the objects, functions, and relations in another.

In some cases, changing one's conceptualization of the world can make it impossible to express certain kinds of knowledge. A famous example of this is the controversy in the field of physics between the view of light as a wave phenomenon and the view of light in terms of particles. Each conceptualization allowed physicists to explain different aspects of the behavior of light, but neither alone sufficed. Not until the two views were merged in modern quantum physics were the discrepancies resolved.

In other cases, changing one's conceptualization can make it more difficult to express knowledge, without necessarily making it impossible. A good example of this, once again in the field of physics, is changing one's frame of reference. Given Aristotle's geocentric view of the universe, astronomers had great difficulty explaining the motions of the moon and other planets. The data were explained (with epicycles, etc.) in the Aristotelian conceptualization, although the explanation was extremely cumbersome. The switch to a heliocentric view quickly led to a more perspicuous theory.

This raises the question of what makes one conceptualization more appropriate than another for knowledge formalization. Currently, there is no comprehensive answer to this question. However, there are a few issues that are especially noteworthy.

One such issue is the *grain size* of the objects associated with a conceptualization. Choosing too small a grain can make knowledge formalization prohibitively tedious. Choosing too large a grain can make it impossible. As an example of the former problem, consider a conceptualization of the scene in Figure 2.1 in which the objects in the universe of discourse are the atoms composing the blocks in the picture. Each block is composed of enormously many atoms, so the universe of discourse is extremely large. Although it is in principle possible to describe the scene at this level of detail, it is senseless if we are interested in only the vertical relationship of the blocks made up of those atoms. Of course, for a chemist interested in the composition of blocks, the atomic view of the

scene might be more appropriate. For this purpose, our conceptualization in terms of blocks has too large a grain.

Finally, there is the issue of *reification* of functions and relations as objects in the universe of discourse. The advantage of this is that it allows us to consider properties of properties. As an example, consider a Blocks World conceptualization in which there are five blocks, no functions, and three unary relations, each corresponding to a different color. This conceptualization allows us to consider the colors of blocks but not the properties of those colors.

$$\langle \{a, b, c, d, e\}, \{\}, \{red, white, blue\} \rangle$$

We can remedy this deficiency by *reifying* various color relations as objects in their own right and by adding a partial function—such as *color*—to relate blocks to colors. Because the colors are objects in the universe of discourse, we can then add relations that characterize them; e.g., *nice*.

$$\langle \{a, b, c, d, e, red, white, blue\}, \{color\}, \{nice\} \rangle$$

Note that, in this discussion, no attention has been paid to the question of whether the objects in one's conceptualization of the world really exist. We have adopted neither the standpoint of *realism*, which posits that the objects in one's conceptualization really exist, nor that of *nominalism*, which holds that one's concepts have no necessary external existence. Conceptualizations are our inventions, and their justification is based solely on their utility. This lack of commitment indicates the essential ontological promiscuity of AI: Any conceptualization of the world is accommodated, and we seek those that are useful for our purposes.

2.2 Predicate Calculus

Given a conceptualization of the world, we can begin to formalize knowledge as sentences in a language appropriate to that conceptualization. In this section, we define a formal language called *predicate calculus*.

All the sentences in predicate calculus are strings of characters arranged according to precise rules of grammar. For example, we can express the fact that block a is above block b by taking a relation symbol such as `Above` and object symbols `A` and `B` and combining them with appropriate parentheses and commas, as follows.

```
Above(A,B)
```

One source of expressiveness in predicate calculus is the availability of logical operators that allow us to form complex sentences from simple ones without specifying the truth or falsity of the constituent sentences. For example, the following sentence using the operator `v` states that either

block a is above block b or block b is above block a, but it makes no commitment as to which is the case.

```
Above(A,B) ∨ Above(B,A)
```

The flexibility also stems from the availability of quantifiers and variables. The quantifier ∀ allows us to state facts about all the objects in our universe of discourse without enumerating them. For example, the first sentence in the following set states that every block that is on another block is above the other block. The quantifier ∃ allows us to assert the existence of an object with certain properties without identifying the object. For example, the second sentence states that there is a block that is both clear and on the table.

```
∀x∀y On(x,y) ⇒ Above(x,y)
∃x Clear(x) ∧ Table(x)
```

To use a language such as predicate calculus, we need to know both its syntax and its semantics. In this section, we describe the syntax of the language in detail; as we present each construct, we informally suggest the semantics. In the next section, we define the semantics of the language formally.

The alphabet of our version of predicate calculus consists of the following *characters*. Spaces and carriage returns have no significance and are used solely for formatting.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0 . , ( ) { } [ ] + - * / ↑
∈ ∪ ∩ = < > ≤ ≥ ⊂ ⊃ ⊆ ⊇ ¬ ∧ ∨ ∀ ∃ ⇒ ⇐ ⇔
```

There are two types of *symbols* in predicate calculus: variables and constants. Constants are further subdivided into object constants, function constants, and relation constants.

A *variable* is any sequence of lowercase alphabetic and numeric characters in which the first character is lowercase alphabetic. As we mentioned, variables are used to express properties of objects in the universe of discourse without explicitly naming them.

An *object constant* is used to name a specific element of a universe of discourse. Every object constant is a sequence of alphabetic characters or

digits in which the first character is either uppercase alphabetic or numeric. The following are simple examples with obvious intended interpretations.

Confucius	Elephants	32456
Stanford	Justice	MCMXII
California	Resistor14	Twelve

A *function constant* is used to designate a function on members of the universe of discourse. Every function constant is either a functional operator ($+$, $-$, $*$, $/$, $\uparrow$, $\cap$, $\cup$) or a sequence of alphabetic characters or digits in which the first character is uppercase alphabetic. The following symbols are examples.

Age	Sin	Cardinality
Weight	Cos	President
Color	Tan	Salary

Every function constant has an associated *arity*, which indicates the number of arguments it is expected to take. For example, Sin normally takes a single argument, and $\uparrow$ takes two arguments. Symbols that stand for associative functions, such as $+$, take any number of arguments.

A *relation constant* is used to name a relation on the universe of discourse. Every relation constant is either a mathematical operator ($=$, $<$, $>$, $\leq$, $\geq$, $\in$, $\subset$, $\supset$, $\subseteq$, $\supseteq$) or a sequence of alphabetic characters or digits in which the first character is uppercase alphabetic; the following symbols are examples.

Odd	Parent	Above
Even	Relative	Between
Prime	Neighbor	Nearby

Like function constants, every relation constant has an associated arity. In addition, every n-ary function constant can also be used as an $(n+1)$-ary relation constant, as we will discuss. However, the converse is not necessarily true.

Note that the type and arity of an alphanumeric constant can be determined only by the way it is used in sentences; these properties cannot be determined on the basis of the symbol's constituent characters. Different people may use the same symbol in different ways.

In predicate calculus, a *term* is used as a name for an object in the universe of discourse. There are three types of terms: variables, object

constants, and functional expressions. We have already discussed variables and object constants.

A *functional expression* consists of an n-ary function constant π and n terms $\tau_1, \ldots, \tau_n$, arranged with parentheses and commas as follows.

$$\pi(\tau_1, \ldots, \tau_n)$$

For example, assuming that `Age` and `Cardinality` are both unary function constants and that `Log` is a binary function constant, the following expressions are all legal terms.

```
Age(Confucius)
Cardinality(Elephants)
Log(32456,2)
```

Although this syntax is quite general, it is somewhat cumbersome for writing expressions involving common mathematical functions. For this reason, the class of functional expressions also is defined to include terms in any of the following *infix* forms. In each case, the operator is the function constant and the surrounding terms designate its arguments.

$(\tau_1+\tau_2)$	$(\tau_1\uparrow\tau_2)$
$(\tau_1-\tau_2)$	$(\tau_1\cap\tau_2)$
$(\tau_1*\tau_2)$	$(\tau_1\cup\tau_2)$
(τ_1/τ_2)	$(\tau_1.\tau_2)$

The use of braces designates an unordered set of elements denoted by the enclosed terms. The use of square brackets denotes a sequence.

$$\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$$
$$[\sigma_1, \sigma_2, \ldots, \sigma_n]$$

From the definitions, it should be clear that functional expressions can be composed in combination with one another, as in the following examples.

```
Log(Cardinality(Elephants),2)
(2*(A↑3))
(Log(A)+Log(B))
```

In predicate calculus, facts are stated in the form of expressions called *sentences*, or sometimes *well-formed formulas* or *wffs*. There are three types of sentences: atomic sentences, logical sentences, and quantified sentences.

An *atomic sentence*, or *atom*, is formed from an n-ary relation constant ρ and n terms $\tau_1, \ldots, \tau_n$, by combining them as follows.

$$\rho(\tau_1, \ldots, \tau_n)$$

Writing atomic sentences involving mathematical relations in this form can be cumbersome. For this reason, the class of atomic sentences also is defined to include expressions in any of the following infix forms.

$$(\tau_1 = \tau_2) \qquad (\tau_1 \in \tau_2)$$
$$(\tau_1 < \tau_2) \qquad (\tau_1 \subset \tau_2)$$
$$(\tau_1 > \tau_2) \qquad (\tau_1 \supset \tau_2)$$
$$(\tau_1 \leq \tau_2) \qquad (\tau_1 \subseteq \tau_2)$$
$$(\tau_1 \geq \tau_2) \qquad (\tau_1 \supseteq \tau_2)$$

Atomic sentences involving these relations are sometimes given special names. For example, the sentence $(\tau_1 = \tau_2)$ is called an *equation*.

Function constants can also be used as relation constants, provided an expression designating the value of the function is included as the final argument. For example, the following two expressions are legal, and the facts they express are identical.

```
(Age(Confucius)=100)
Age(Confucius,100)
```

We also want to be able to express facts that cannot conveniently be expressed by atomic sentences. One often needs to express negations, disjunctions, contingencies, and so on. In predicate calculus, atomic sentences such as these can be combined with logical operators to form *logical sentences*.

A *negation* is formed using the $\neg$ operator. A sentence of the following form is true if and only if the embedded sentence is not true (regardless of the interpretation of the embedded sentence).

$$(\neg\phi)$$

A *conjunction* is a set of sentences connected by the $\wedge$ operator. Each of the component sentences is called a *conjunct*. A conjunction is true if and only if each of the conjuncts is true.

$$(\phi_1 \wedge \ldots \wedge \phi_n)$$

A *disjunction* is a set of sentences connected by the $\vee$ operator. Each of the component sentences is called a *disjunct*. A disjunction is true if and only if at least one of the disjuncts is true. Note that more than one of the disjuncts may be true.

$$(\phi_1 \vee \ldots \vee \phi_n)$$

An *implication* is formed using the $\Rightarrow$ operator. The sentence to the left of the operator is called the *antecedent*, and the sentence to the right is called the *consequent*. An implication is a statement that the consequent is true whenever the antecedent is true. By convention, whenever the antecedent is false, the implication is assumed to be true, whether or not the consequent is true.

$$(\phi \Rightarrow \psi)$$

A *reverse implication*, formed using the $\Leftarrow$ operator, is just an implication with its arguments reversed. The antecedent is to the right, and the consequent is to the left.

$$(\psi \Leftarrow \phi)$$

A *bidirectional implication* or *equivalence*, formed using the $\Leftrightarrow$ operator, is a statement that the component sentences are either both true or both false.

$$(\phi \Leftrightarrow \psi)$$

The following are all logical sentences. The intended meaning of the first is that the age of Confucius is not 100. The second states that elephants are either herbivores or carnivores. The third states that George is sick if he is at home.

```
(¬Age(Confucius,100))

((Elephants ⊂ Carnivores) ∨ (Elephants ⊂ Herbivores))

(Location(George,Home) ⇒ Sick(George))
```

With the syntax given so far, we can designate objects only by name (using an object symbol) or by description (using a functional expression). *Quantified sentences* provide a more flexible way of talking about all objects in our universe of discourse or asserting a property of an individual object without identifying that object.

A *universally quantified sentence* is formed by combining the *universal quantifier* $\forall$, a variable ν, and any simpler sentence ϕ. The intended meaning is that the sentence ϕ is true, no matter what object the variable ν represents.

$$(\forall \nu \ \phi)$$

The following two sentences are examples. The first states that all apples are red. The second states that every object in the universe of discourse is a red apple.

$$(\forall x \ (\text{Apple}(x) \Rightarrow \text{Red}(x)))$$

$$(\forall x \ (\text{Apple}(x) \wedge \text{Red}(x)))$$

An *existentially quantified sentence* is formed by combining the *existential quantifier* $\exists$, a variable ν, and any simpler sentence ϕ. The intended meaning is that the sentence ϕ is true, for at least one object in the universe of discourse.

$$(\exists \nu \ \phi)$$

Of the two sentences that follow, the first states that there is a red apple in the universe of discourse; the second states that there is an object that is either an apple or a pear.

$$(\exists x \ (\text{Apple}(x) \wedge \text{Red}(x)))$$

$$(\exists x \ (\text{Apple}(x) \vee \text{Pear}(x)))$$

A *quantified sentence* is either a universally quantified sentence or an existentially quantified sentence. The *scope* of the quantifier in a quantified sentence is the sentence embedded within the quantified sentence.

Like atomic and logical sentences, quantified sentences can be combined to form more complex sentences, as in the following examples.

$$((\forall x \ \text{Apple}(x)) \vee (\exists x \ \text{Pear}(x)))$$

$$(\forall x (\forall y \ \text{Loves}(x,y)))$$

When a quantified sentence of one type is nested within a quantified sentence of the other type, the order of nesting is extremely important.

```
(∀x(∃y Loves(x,y)))
(∃y(∀x Loves(x,y)))
```

The first sentence states that every person has some person he loves and makes no statement about whether the object of one person's love is the same as the object of another person's love. The second sentence states that there is one single person whom everybody loves, a quite different statement.

A variable can also occur as a term in a sentence without an enclosing quantifier. When used in this way, a variable is said to be *free*, whereas a variable that occurs in a sentence and in an enclosing quantifier is said to be *bound*. For example, in the following sentences, the variable `x` is free in the first, bound in the second, and occurs both free and bound in the third:

```
(Apple(x) ⇒ Red(x))
(∀x (Apple(x) ⇒ Red(x)))
(Apple(x) ∨ (∃x Pear(x)))
```

If a sentence has no free variables, it is called a *closed* sentence. If it has neither free nor bound variables, it is called a *ground* sentence.

Note that the variables in quantified sentences refer to objects of a universe of discourse, not to functions or relations. Consequently, they cannot be used in functional or relational positions. We say that a language with this property is *first order*. A *second-order* language is one with function and relation variables as well. We have chosen to restrict our attention here to a first-order language both because this language allows us to prove some strong results that simply do not hold of second-order languages and because it is adequate for most purposes in AI.

Note that the parentheses around expressions involving functional, relational, and logical operators are essential to prevent ambiguities. If they were dropped indiscriminately, some terms could be interpreted in more than one way. For example, `A*B+C` could be the sum of a product and a constant or the product of a constant and a sum. Fortunately, such ambiguities often can be eliminated by imposing a precedence ordering on operators.

A table of precedences is given in Table 2.1. The symbol ↑ has higher precedence than * and /. The symbols * and / have higher precedence than + and −. Whenever an expression is flanked by operators of different precedence, it is associated with the operator of higher precedence; e.g., the

Table 2.1 Precedence relations ordered from high to low

↑
* / ∩
+ - ∪
= < > ≤ ≥ ∈ ⊂ ⊃ ⊆ ⊇
¬
∧
∨
⇒ ⇐ ⇔
∀ ∃

expression `A*B+C` is the sum of a product and a constant. Whenever an expression is flanked by operators of equal precedence, it is associated with the operator on the left; e.g., the expression `A*B/C` is a quotient of a product and a constant. These rules of precedence are assumed throughout this book, and parentheses usually are dropped when there is no chance of ambiguity.

In addition to dropping parentheses, it is common in mathematical notation to drop the parentheses following a 0-ary function constant or relation constant. So, in the interests of simplicity, we permit this abbreviation in our language. Thus, the term `F()` can be written as `F`, and the atomic sentence `R()` can be written as `R`.

Another concession to standard notation is the abbreviation of negated atomic sentences involving mathematical operators. Rather than writing the negation operator in prefix position as indicated, the fact that the atomic sentence is negated is denoted by drawing a bar through the operator. Thus, we usually write the sentence $\phi \not\leq \psi$ in place of $\neg(\phi \leq \psi)$.

The preceding sections completely characterize the syntax of predicate calculus. Any sentence allowed by these rules and conventions is syntactically correct, and any sentence not specifically allowed is syntactically illegal. In later chapters, we modify the syntax slightly to allow additional types of sentences.

2.3 Semantics

In the previous section, we provided a precise definition for the syntax of predicate calculus. The treatment of semantics, however, was quite informal. In this section, we provide a precise definition of meaning called *declarative semantics.*

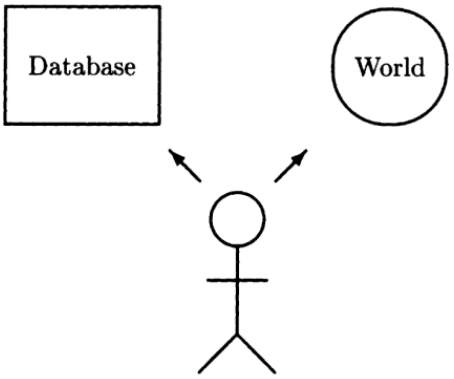


Figure 2.2 Declarative semantics.

In making our definition, we assume the perspective of the observer in Figure 2.2. We have a set of sentences and a conceptualization of the world, and we associate the symbols used in the sentences with the objects, functions, and relations of our conceptualization. We evaluate the truth of the sentences in accordance with this association, saying that a sentence is true if and only if it accurately describes the world according to our conceptualization.

Note that, in this definition of semantics, there is no dependence on the way in which the sentences of the language are expected to be used. In this respect, the approach differs radically from that taken in much of computer science, where abstract data structures are defined in terms of the operations we can perform on them.

An *interpretation* I is a mapping between elements of the language and elements of a conceptualization. We represent the mapping by the function $I(\sigma)$, where σ is an element of the language. We usually abbreviate $I(\sigma)$ to σ^I, and we represent the universe of discourse by $|I|$. For I to be an interpretation, it must satisfy the following properties.

(1) If σ is an object constant, then $\sigma^I \epsilon |I|$.

(2) If π is an n-ary function constant, then $\pi^I : |I|^n \rightarrow |I|$.

(3) If ρ is an n-ary relation constant, then $\rho^I \subseteq |I|^n$.

Note that, in describing the semantics of predicate calculus, we use symbols, such as I and σ, that are not part of the language we are describing. I, σ, and other symbols we introduce later are part of our *metalanguage* for talking *about* predicate calculus. Practice keeping firmly in mind which symbols and expressions are part of the predicate-calculus language and which are part of the metalanguage.

As an example of an interpretation, once again consider the Blocks World scene in Figure 2.1. Assume the predicate-calculus language has the five object constants `A`, `B`, `C`, `D`, and `E`, the function constant `Hat`, and the relation constants `On`, `Above`, `Table`, and `Clear`. The following mapping corresponds to our usual interpretation for these symbols.

$$\begin{aligned}
\texttt{A}^I &= a \\
\texttt{B}^I &= b \\
\texttt{C}^I &= c \\
\texttt{D}^I &= d \\
\texttt{E}^I &= e \\
\texttt{Hat}^I &= \{\langle b, a\rangle, \langle c, b\rangle, \langle e, d\rangle\} \\
\texttt{On}^I &= \{\langle a, b\rangle, \langle b, c\rangle, \langle d, e\rangle\} \\
\texttt{Above}^I &= \{\langle a, b\rangle, \langle b, c\rangle, \langle a, c\rangle, \langle d, e\rangle\} \\
\texttt{Table}^I &= \{c, e\} \\
\texttt{Clear}^I &= \{a, d\}
\end{aligned}$$

This is the *intended interpretation*, the one suggested by the names of the constants. However, these constants can equally well be interpreted in other ways, as in the interpretation J that follows. J agrees with I on the object constants and the function constant but disagrees on the relation constants. Under this interpretation, `On` means *under*, `Above` means *below*, `Table` means *clear*, and `Clear` means *table*.

$$\begin{aligned}
\texttt{A}^J &= a \\
\texttt{B}^J &= b \\
\texttt{C}^J &= c \\
\texttt{D}^J &= d \\
\texttt{E}^J &= e \\
\texttt{Hat}^J &= \{\langle b, a\rangle, \langle c, b\rangle, \langle e, d\rangle\} \\
\texttt{On}^J &= \{\langle b, a\rangle, \langle c, b\rangle, \langle e, d\rangle\} \\
\texttt{Above}^J &= \{\langle b, a\rangle, \langle c, b\rangle, \langle c, a\rangle, \langle e, d\rangle\} \\
\texttt{Table}^J &= \{a, d\} \\
\texttt{Clear}^J &= \{c, e\}
\end{aligned}$$

For reasons that will become clear, it is useful to interpret variables in sentences separately from other symbols. A *variable assignment* U is a function from the variables of a language to objects in the universe of discourse.

The following partial assignment is an example. (We use the abbreviation σ^U for $U(\sigma)$.) The variable x is assigned block a; variable y also is assigned block a; and variable z is assigned block b.

$$\mathtt{x}^U = a$$
$$\mathtt{y}^U = a$$
$$\mathtt{z}^U = b$$

An interpretation I and a variable assignment U can be combined into a joint assignment T_{IU} that applies to terms in general. In particular, the assignment of each nonvariable symbol corresponds to the interpretation I; the assignment of each variable corresponds to the variable assignment U; and the assignment of an expression is the result of applying the function corresponding to the function constant to the objects designated by the terms.

Given an interpretation I and a variable assignment U, the *term assignment* T_{IU} corresponding to I and U is a mapping from terms to objects, defined as follows.

(1) If τ is an object constant, then $T_{IU}(\tau) = I(\tau)$.

(2) If τ is a variable, then $T_{IU}(\tau) = U(\tau)$.

(3) If τ is a term of the form $\pi(\tau_1, \ldots, \tau_n)$ and $I(\pi) = g$ and $T_{IU}(\tau_i) = x_i$, then $T_{IU}(\tau) = g(x_1, \ldots, x_n)$.

As an example, consider the term assignment corresponding to the interpretation I and the variable assignment U defined previously. Under this assignment, the term `Hat(C)` designates the block b. I maps C to block c and the tuple $\langle c, b \rangle$ is a member of the function designated by `Hat`. The term `Hat(z)` designates block a, since U maps z to b, and the tuple $\langle b, a \rangle$ is in the set of tuples designated by `Hat`.

The notions of interpretation and variable assignment are important because they allow us to define a relative notion of truth called *satisfaction.* The definition differs from one type of sentence to another and is presented case by case in the following paragraphs. By convention, the fact that a sentence ϕ is satisfied by an interpretation I and a variable assignment U is written $\models_I \phi$[U]. In this case, we say that the sentence ϕ is *true* relative to the interpretation I and the assignment U.

An interpretation and variable assignment satisfy an equation if and only if the corresponding term assignment maps the equated terms into the same object. When this is the case, the two terms are said to be *coreferential.*

(1) $\models_I$ $(\sigma{=}\tau)$[U] if and only if $T_{IU}(\sigma) = T_{IU}(\tau)$.

An interpretation I and a variable assignment U satisfy an atomic sentence other than an equation if and only if the tuple formed from the objects designated by the terms in the sentence is an element of the the relation designated by the relation constant.

(2) $\models_I \rho(\tau_1, \ldots, \tau_n)$[U] if and only if $\langle T_{IU}(\tau_1), \ldots, T_{IU}(\tau_n)\rangle \epsilon I(\rho)$.

For example, consider interpretation I as defined in the previous section. Since the object constant `A` designates block a and `B` designates b, and the tuple $\langle a, b\rangle$ is a member of the set designated by the relation constant `On`, it is the case that $\models_I$ `On(A,B)`[U]. Thus, we can say that `On(A,B)` is true under this interpretation.

If the mapping of the relation symbol `On` were changed to the value in interpretation J (where `On` designates the *under* relation), then the sentence `On(A,B)` would not be satisfied. Tuple $\langle a, b\rangle$ is not a member of that relation, and thus `On(A,B)` would be false under that interpretation.

These examples illustrate the dependence of satisfiability on interpretation. Under some interpretations, a sentence can be true; under other interpretations, it can be false.

The satisfiability of logical sentences depends on the logical operator involved. The negation of a sentence is satisfied if and only if the sentence itself is not satisfied. A conjunction is satisfied if and only if all the conjuncts are satisfied. A disjunction is satisfied if and only if at least one of the disjuncts is satisfied. Note the inclusive sense of disjunction being assumed here. A unidirectional implication is satisfied if and only if the antecedent is false or the consequent is true. A bidirectional implication is satisfied if and only if the two component implications are satisfied.

(3) $\models_I (\neg\phi)$[U] if and only if $\not\models_I \phi$[U].

(4) $\models_I (\phi_1 \wedge \ldots \wedge \phi_n)$[U] if and only if $\models_I \phi_i$[U] for all $i = 1, \ldots, n$.

(5) $\models_I (\phi_1 \vee \ldots \vee \phi_n)$[U] if and only if $\models_I \phi_i$[U] for some i, $1 \leq i \leq n$.

(6) $\models_I (\phi \Rightarrow \psi)$[U] if and only if $\not\models_I \phi$[U] or $\models_I \psi$[U].

(7) $\models_I (\phi \Leftarrow \psi)$[U] if and only if $\models_I \phi$[U] or $\not\models_I \psi$[U].

(8) $\models_I (\phi \Leftrightarrow \psi)$[U] if and only if $\models_I (\phi \Rightarrow \psi)$[U] and $\models_I (\phi \Leftarrow \psi)$[U].

A universally quantified sentence is satisfied if and only if the enclosed sentence is satisfied for all assignments of the quantified variable. An existentially quantified sentence is satisfied if and only if the enclosed sentence is satisfied for some assignment of the quantified variable.

(9) $\models_I (\forall\nu\phi)$[U] if and only if for all $d \in |I|$ it is the case that $\models_I \phi$[V] where $V(\nu) = d$ and $V(\mu) = U(\mu)$ for $\mu \neq \nu$.

(10) $\models_I (\exists\nu\phi)$[U] if and only if for some $d \in |I|$ it is the case that $\models_I \phi$[V] where $V(\nu) = d$ and $V(\mu) = U(\mu)$ for $\mu \neq \nu$.

If an interpretation I satisfies a sentence ϕ for all variable assignments, then I is said to be a *model* of ϕ, written $\models_I \phi$. For example, interpretation I from our Blocks World example is a model of the sentence `On(x,y) ⇒ Above(x,y)`. Consider the variable assignment U that maps `x` into block a and `y` into block b. Under this variable assignment and interpretation I, the sentence `On(x,y)` and the sentence `Above(x,y)` are both satisfied. Therefore, by the definition of satisfaction, they satisfy the implication as well. As an alternative, consider the variable assignment V that maps both `x` and `y` into block a. Under this variable assignment, `Above(x,y)` is not satisfied—but neither is `On(x,y)`. Therefore, once again the implication is satisfied.

Obviously, a variable assignment has no effect on the satisfaction of a sentence that contains no free variables (e.g., a ground sentence or a closed sentence). Consequently, any interpretation that satisfies a ground sentence for one variable assignment is a model of that sentence.

A sentence is said to be *satisfiable* if and only if there is *some* interpretation and variable assignment that satisfy it. Otherwise, it is *unsatisfiable*. A sentence is *valid* if and only if it is satisfied by *every* interpretation and variable assignment. Valid sentences are those that are true by virtue of their logical form and thus provide no information about the domain being described. The sentence `P(A) ∨ ¬P(A)` is valid, because any interpretation satisfies either `P(A)` or `¬P(A)`.

We can easily extend the definitions in this section to sets of sentences as well as individual sentences. A set Γ of sentences is satisfied by an interpretation I and a variable assignment U (written $\models_I \Gamma[U]$) if and only if every member of Γ is satisfied by I and U. An interpretation I is a model of a set Γ of sentences (written $\models_I \Gamma$) if and only if it is a model of every member of the set. A set of sentences is satisfiable if and only if there is an interpretation and variable assignment that satisfies every element. Otherwise, it is unsatisfiable or *inconsistent*. A set of sentences is valid if and only if every element is valid.

Unfortunately, our definition of satisfaction is somewhat disturbing in that it relativizes the notion of truth to one's interpretation. As a result, different individuals with different interpretations may disagree on the truth of a sentence.

It is generally true that, as one writes more sentences, the number of possible models decreases. This raises the question of whether it is possible for an individual to define his symbols so thoroughly that no interpretation is possible except the one he intended. As it turns out, there is no way in general of ensuring a unique interpretation, no matter how many sentences we write down.

The concept of elementary equivalence is important in this regard. It means that two interpretations cannot be distinguished by sentences in predicate calculus. More precisely, two interpretations I and J are

elementarily equivalent ($I \equiv J$) if and only if $\models_I \phi$ implies and is implied by $\models_J \phi$ for any sentence ϕ.

Consider the two interpretations I and J defined as follows. I's universe of discourse consists of the real numbers, and I maps the relation symbol `R` into the *greater than* relation on reals. J's universe of discourse consists of the rational numbers, and J maps `R` into the *greater than* relation on rationals. As it turns out, I and J are elementarily equivalent. Despite the fact that the two universes are of different cardinality, there is no sentence that is satisfied by one interpretation that is not also satisfied by the other.

Along with the problem of ambiguity in the definition of symbols, there is the parallel issue of the definability of the elements in a conceptualization (i.e., the objects, functions, and relations). An element x of a conceptualization is *definable* in terms of elements $x_1, \ldots, x_n$ if and only if there is a first-order sentence ϕ with nonlogical symbols $\sigma_1, \ldots, \sigma_n$ and σ for which every model over the conceptualization that maps σ_i into x_i also maps σ into x.

For example, it is possible to define the *clear* relation in terms of the *on* relation. Given an interpretation I that maps the symbol `On` into the *on* relation, we can define the *clear* relation using the single sentence `¬∃x On(x,y)`. An object is clear if and only if no object is on top of it.

Unfortunately, not every relation on a universe of discourse is definable with every interpretation. For any interpretation with an infinite universe of discourse, there are uncountably many relations, but the language of predicate calculus has only a countable number of finite sentences. As a result, some relations must necessarily be left out.

For example, the relation *on* cannot be defined in terms of *clear*. With a fixed interpretation for `Clear`, the sentence `¬∃x On(x,y)` constrains the set of possible interpretations for `On` but does not make it unique.

Before we examine specific examples, it is worth pausing briefly to consider the relevance of these ideas to representing knowledge in machines. As we already mentioned, the first step in encoding declarative knowledge is conceptualizing the application area. We then select a vocabulary of object constants, function constants, and relation constants. We associate these constants with the objects, functions, and relations in our conceptualization. We can then begin to write sentences that constitute the machine's declarative knowledge.

Of course, in designing a useful machine, we try to write sentences we believe to be true; i.e., ones that are satisfied by our intended interpretation. The intended interpretation is then a model of the sentences we write. Notice that, if our beliefs are incorrect, the sentences we write may not be true in reality.

Note also that, in describing an application area, we seldom start with a complete conceptualization. For example, we rarely have lists of tuples for every function and relation. Rather, we start with an idea of a

conceptualization and attempt to make it precise by writing more and more sentences.

2.4 Blocks World Example

As an example of expressing knowledge in predicate calculus, once again consider the Blocks World scene in Figure 2.1. We assume a conceptualization of the scene as five objects and the relations *on*, *clear*, *table*, and *above*. For our predicate-calculus vocabulary, we use the five object constants `A`, `B`, `C`, `D`, and `E` and the relation constants `On`, `Clear`, `Table`, and `Above`. In using these symbols to encode facts about our conceptualization, we assume the standard interpretation, I.

The sentences that follow encode the essential information about this scene. Block a is on block b; block b is on block c; and block d is on block e. Block a is above b and c; b is above c; and d is above e. Finally, block a is clear, and so is block d. Block c is on the table, and so is block e.

`On(A,B)`	`Above(A,B)`	`Clear(A)`
`On(B,C)`	`Above(B,C)`	`Clear(D)`
`On(D,E)`	`Above(A,C)`	`Table(C)`
	`Above(D,E)`	`Table(E)`

Note that all these sentences are true under the intended interpretation. Since `A` designates block a and `B` designates block b, and block a is on block b, the first sentence in the first column is true. Since `D` designates block d and `E` designates block e, and the pair $\langle d, e \rangle$ is a member of the relation designated by the symbol `Above`, the last sentence in the second column is true. For similar reasons, the other sentences are true as well.

In addition to encoding simple sentences, we can encode more general facts. In the Blocks World, if one block is on another, then that block is above the other. Furthermore, the *above* relation is transitive; if one block is above a second and the second is above a third, then the first also is above the third.

```
∀x∀y (On(x,y) ⇒ Above(x,y))
∀x∀y∀z (Above(x,y) ∧ Above(y,z) ⇒ Above(x,z))
```

One advantage to writing such general sentences is economy. If we record *on* information for every object and encode the relationship between the *on* relation and the *above* relation, there is no need to record any *above* information explicitly.

Another advantage is that these general sentences apply to Blocks World scenes other than the one pictured here. It is possible to create a Blocks World scene in which none of the specific sentences we have listed is true, but the general sentences are still correct.

Of course, there are other true sentences one can write about the Blocks World. Many of these sentences are redundant in that they are entailed by the preceding sentences. This notion of logical entailment is defined more precisely in the next chapter.

2.5 Circuits Example

Figure 2.3 is a schematic diagram for a digital circuit called a full adder. Let us consider how we might conceptualize a circuit of this sort and describe its structure as a set of sentences in predicate calculus.

We can think of the circuit f_1 as a composite component, consisting of subcomponents called "gates." There are two *xor* gates x_1 and x_2, two *and* gates a_1 and a_2, and an *or* gate o_1. Each device has a number of ports through which data flow. The input ports are on the left side of the box designating the device; the output ports are on the right side. Thus, the universe of discourse consists of 26 objects: 6 components and 20 ports.

We can use functions to associate ports with components. The binary function *input* maps an integer and a component into the corresponding input port. The binary function *output* maps an integer and a component into the corresponding output port. Thus, we can think about the first input of the adder, or about its second output, and so forth.

The solid lines connecting ports to other ports depict wires, which transmit data between components. We could conceptualize these wires as objects like gates, with inputs and outputs of their own; but this would not answer the question of how to encode the relationship between the inputs

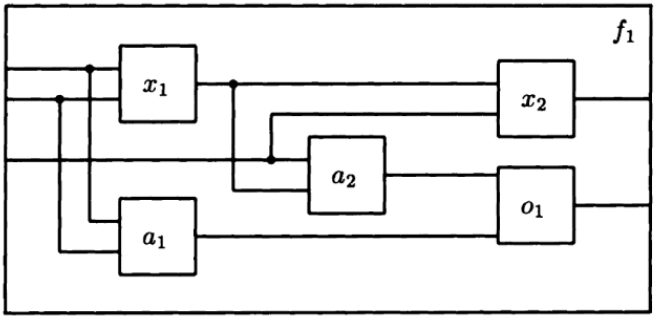


Figure 2.3 A full adder.

and outputs of those wires and the ports to which they are connected. Instead, let us ignore the presence of wires and think about the connectivity of a circuit using a single binary relation that associates ports with the ports to which they are connected. For example, the third input of f_1 is connected to the first input of a_2. We assume that connectivity is unidirectional, from left to right.

To describe the structure of f_1 in predicate calculus, we need to choose symbols that designate the elements of our conceptualization. The following vocabulary suffices.

- `F1, X1, X2, A1, A2, O1` designate the six components.
- `Adder(x)` means that `x` is an adder.
- `Xorg(x)` means that `x` is an *xor* gate.
- `Andg(x)` means that `x` is an *and* gate.
- `Org(x)` means that `x` is an *or* gate.
- `I(i,x)` designates the `i`th input port of device `x`.
- `O(i,x)` designates the `i`th output port of device `x`.
- `Conn(x,y)` means that port `x` is connected to port `y`.

Given this vocabulary, we can describe our conceptualization of the circuit with the following predicate-calculus sentences. The first six sentences indicate the types of the components. The remaining sentences capture their connectivity.

```
Adder(F1)
Xorg(X1)
Xorg(X2)
Andg(A1)
Andg(A2)
Org(A3)

Conn(I(1,F1),I(1,X1))
Conn(I(2,F1),I(2,X1))
Conn(I(1,F1),I(1,A1))
Conn(I(2,F1),I(2,A1))
Conn(I(3,F1),I(2,X2))
Conn(I(3,F1),I(1,A2))
```

```
Conn(O(1,X1),I(1,X2))
Conn(O(1,X1),I(2,A2))
Conn(O(1,A2),I(1,O1))
Conn(O(1,A1),I(2,O1))
Conn(O(1,X2),O(1,F1))
Conn(O(1,O1),O(2,F1))
```

We can describe the state of a circuit such as f_1 by enlarging our conceptualization to include high and low values (i.e., bits) and a relation that associates a port with the value on that port. These additional conceptual elements suggest the following vocabulary.

- `V(x,z)` means that the value on port `x` is `z`.
- `1` and `0` designate high and low signals, respectively.

Using this vocabulary, we can assert specific values for the ports in the circuit. For example, the following sentences assert that the inputs to the circuit are high, low, and high, respectively, and that the outputs are low and high.

```
V(I(1,F1),1)
V(I(2,F1),0)
V(I(3,F1),1)

V(O(1,F1),0)
V(O(1,F1),1)
```

We also can use this vocabulary to describe the general behavior of the components in the circuit. The first two sentences that follow capture the behavior of an *and* gate; the second pair of sentences describes the behavior of an *or* gate; and the third pair describes an *xor* gate. The final sentence describes the behavior of an ideal connection.

```
∀x (Andg(x) ∧ V(I(1,x),1) ∧ V(I(2,x),1) ⇒ V(O(1,x),1))
∀x∀n (Andg(x) ∧ V(I(n,x),0) ⇒ V(O(1,x),0))

∀x∀n (Org(x) ∧ V(I(n,x),1) ⇒ V(O(1,x),1))
∀x (Org(x) ∧ V(I(1,x),0) ∧ V(I(2,x),0) ⇒ V(O(1,x),0))

∀x∀z (Xorg(x) ∧ V(I(1,x),z) ∧ V(I(2,x),z) ⇒ V(O(1,x),0))
```

```
∀x∀y∀z (Xorg(x) ∧ V(I(1,x),y) ∧ V(I(2,x),z) ∧ y≠z
        ⇒ V(O(1,x),1))

∀x∀y∀z (Conn(x,y) ∧ V(x,z) ⇒ V(y,z))
```

Notice that these sentences completely characterize the digital structure and behavior of f_1. To describe additional properties, we would have to expand or modify our conceptualization and vocabulary. For example, we might want to express the fact that a_1 is malfunctioning. To do this, we would have to add an additional relation and phrase an appropriate sentence. Asserting that a connection is malfunctioning is a little more complicated, because connections are not objects. To express such information, we would need to reify connections. For the circuit in Figure 2.3, this would lead to 12 new objects; to relate these new connection objects to the ports they connect, we would have to extend the binary connectivity relation into a ternary relation that associates a port, the port to which it is connected, and the corresponding connection. In formalizing knowledge, it is always important to recognize the need for a new conceptualization and a new vocabulary when appropriate.

2.6 Algebraic Examples

Using predicate calculus, we can express the definitions and properties of common mathematical functions and relations, as illustrated by the examples in this section.

The following sentences express the associativity, commutativity, and identity properties of the + function. The first sentence states that the number obtained by adding x to the result of adding y to z is the same as the number obtained by adding to z the result of adding x and y. The second sentence states that the order of addition is unimportant. The third sentence states that 0 is an identity for +.

```
∀x∀y∀z x+(y+z)=(x+y)+z

∀x∀y x+y=y+x

∀x x+0=x
```

In its usual interpretation, the ≤ symbol is used to denote a partial order; i.e., one that is reflexive, antisymmetric, and transitive. The first sentence that follows asserts that the relation holds of any object and itself. The second sentence asserts that, if the relation holds between object x and object y and between y and x, then x and y must be equal. The third

sentence states that the relation holds between object x and object z if it holds between object x and object y and also between object y and object z.

$$\forall x\ x \leq x$$

$$\forall x \forall y\ x \leq y \wedge y \leq x \Rightarrow x = y$$

$$\forall x \forall y \forall z\ x \leq y \wedge y \leq z \Rightarrow x \leq z$$

We can characterize functions and relations on sets in a similar manner. For example, given the membership relation $\in$, we can define the intersection function $\cap$, as follows. An object is a member of the intersection of two sets if and only if it is a member of both sets.

$$\forall s \forall t \forall x\ (x \in s \wedge x \in t) \Leftrightarrow x \in s \cap t$$

The following sentences express the associativity, commutativity, and idempotence of the intersection function. All three properties can be proved from the preceding definition.

$$\forall r \forall s \forall t\ r \cap (s \cap t) = (r \cap s) \cap t$$

$$\forall s \forall t\ s \cap t = t \cap s$$

$$\forall s\ s \cap s = s$$

If the sentences in this section look familiar, that is as it should be. Predicate calculus was originally developed to express mathematical facts, and it is in common use today.

2.7 List Examples

If $\tau_1, \ldots, \tau_n$ are legal terms in our language, then a *list* is a term of the following form, where n is any integer greater than or equal to zero.

$$[\tau_1, \ldots, \tau_n]$$

The primary use of lists is the representation of sequences of objects. For example, if we use numerals to designate numbers, we can use the following list to designate the sequence consisting of the first three integers in ascending order.

$$[1,2,3]$$

Because lists are themselves terms, we can nest lists within lists. For example, the following term is a list of all permutations of the first three positive integers.

```
[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

To talk about lists of arbitrary length, we use the binary functional operator "." in infix form. In particular, a term of the form $\tau_1 . \tau_2$ designates a sequence in which τ_1 is the first element and τ_2 is the rest of the list. With this operator, we can rewrite the list `[1,2,3]` as follows.

```
(1.(2.(3.[])))
```

The advantage of this representation is that it allows us to describe functions and relations on lists without regard to length.

As an example, consider the definition of the binary relation **Member**, which holds of an object and a list if the object is a top-level member of the sequence denoted by the list. Using the "." operator, we can describe the **Member** relation as follows. Obviously, an object is a member of a sequence if it is the first element; however, it is also a member if it is member of the rest of the list.

```
∀x∀l Member(x,x.l)

∀x∀y∀l Member(x,l) ⇒ Member(x,y.l)
```

We also can define functions to manipulate lists in different ways. For example, the following axioms define a function called **Append**. The value of **Append** is a sequence consisting of the elements in the sequence supplied as its first argument followed by the elements in the sequence supplied as its second argument. Thus, `Append([1,2],[3,4])` denotes the same sequence as the list `[1,2,3,4]`.

```
∀m Append([],m)=m

∀x∀l∀m Append(x.l,m)=(x.Append(l,m))
```

Of course, we can also define relations that depend on the structure of the elements of a sequence. For example, the **Among** relation is true of an object and a sequence if the object is a member of the sequence, if it is a member of a sequence that is itself a member of the sequence, and so on.

```
∀x Among(x,x)

∀x∀y∀z (Among(x,y) ∨ Among(x,z)) ⇒ Among(x,y.z)
```

Lists are an extremely versatile representational device, and the reader is encouraged to become as familiar as possible with the techniques of writing definitions for functions and relations on lists. As is true of many tasks, practice is the best approach to gaining skill.

2.8 Natural-Language Examples

As a final example of using predicate calculus, consider the task of formalizing the information in the following English sentences. We assume that the conceptualization underlying all the sentences is the same. The universe of discourse is the set of all plants. There is a unary relation for being a mushroom, another for being purple, and a third for being poisonous. We designate these relations with the unary relation symbols `Mushroom`, `Purple`, and `Poisonous`. Each English sentence is followed by one or more translations into predicate calculus. Where more than one translation is given, the alternatives are logically equivalent.

All purple mushrooms are poisonous.

$\forall x\ \texttt{Purple(x)} \wedge \texttt{Mushroom(x)} \Rightarrow \texttt{Poisonous(x)}$

$\forall x\ \texttt{Purple(x)} \Rightarrow (\texttt{Mushroom(x)} \Rightarrow \texttt{Poisonous(x)})$

$\forall x\ \texttt{Mushroom(x)} \Rightarrow (\texttt{Purple(x)} \Rightarrow \texttt{Poisonous(x)})$

The use of the word *all* in this sentence is a clear indication that it is universally quantified. The equivalence of the three sentences should be obvious. The first states that, if an object is a mushroom and purple, then it is poisonous. The second states that, if an object is purple, then, if it is also a mushroom, it is poisonous. The third sentence states that, if an object is a mushroom, then, if it is also purple, it is poisonous. All three statements assert the poisonous quality of any purple mushroom.

A mushroom is poisonous only if it is purple.

$\forall x\ \texttt{Mushroom(x)} \wedge \texttt{Poisonous(x)} \Rightarrow \texttt{Purple(x)}$

$\forall x\ \texttt{Mushroom(x)} \Rightarrow (\texttt{Poisonous(x)} \Rightarrow \texttt{Purple(x)})$

Here we have the converse of the relationship in the preceding sentence. The argument for equivalence is the same as for the preceding sentence.

(Caution: A conceptualization of the world in which this is a true sentence may be hazardous to your health!)

No purple mushroom is poisonous.

$\forall x \neg(\text{Purple}(x) \wedge \text{Mushroom}(x) \wedge \text{Poisonous}(x))$

$\neg(\exists x\ \text{Purple}(x) \wedge \text{Mushroom}(x) \wedge \text{Poisonous}(x))$

The use of the word *no* here indicates that something is not true. The fact that, for all objects, something is not true (as suggested by the first rendition) is equivalent to the lack of existence of an object for which it is true (as suggested by the second).

There is exactly one mushroom.

$\exists x\ \text{Mushroom}(x) \wedge (\forall z\ z \neq x \Rightarrow \neg\text{Mushroom}(z))$

The easiest way to encode information about the number of objects having a property is to state the cardinality of the set of all objects having that property. Although the specified conceptualization includes neither this set nor the cardinality function, it is possible to express that there is only one mushroom using the equality relation. Note that the fact can be stated even though the identity of the individual mushroom is unknown.

2.9 Specialized Languages

One of the disadvantages of predicate calculus as a knowledge-representation language is that, like English, it is sometimes cumbersome. For this reason, AI researchers often prefer specialized languages, many of them graphical. In this section, we present a few examples and address their strengths and weaknesses for encoding declarative knowledge.

A *binary table*, is an example of a sentence in a graphical language. As our alphabet, we take the set of all uppercase and lowercase letters, the digits, and both horizontal and vertical lines. The symbols are the

π	τ_1	$\cdots$	τ_n
σ_1	α_{11}	$\cdots$	α_{1n}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
σ_m	α_{m1}	$\cdots$	α_{mn}

Figure 2.4 The form of a binary table.

same as those in predicate calculus, except that we divide all symbols into object constants and binary function constants only. A legal sentence in the table language is a two-dimensional configuration of symbols in the form shown in Figure 2.4, where π is a binary function constant and the symbols $\sigma_1, \ldots, \sigma_m$ and $\tau_1, \ldots, \tau_n$ and $\alpha_{11}, \ldots, \alpha_{mn}$ are all object constants.

An interpretation I satisfies a sentence in the table language if and only if each entry in the table designates the value of the function designated by the function constant in the upper-left corner applied to the objects designated by the corresponding row and column labels.

$$\pi^I(\sigma_i^I, \tau_j^I) = \alpha_{ij}^I$$

Figure 2.5 presents a legal binary table, assuming that the symbol `Score` is a binary function constant and the other symbols in the table are all object constants.

Suppose that I is an interpretation that maps the symbols `Gauss`, `Herbrand`, and `Laurent` into the students with those names. It maps the symbols `Quiz1`, `Quiz2`, `Quiz3`, and `Final` into four tests taken by those students. It maps sequences of digits into the corresponding base-10 integers. It maps the function constant `Score` into a function that maps a student and a test into the student's score on that test. Then I satisfies this table if and only if under this mapping the indicated scores are correct; e.g., Gauss got a 94 on the second quiz.

The language of binary tables is excellent for expressing information about binary functions; it does not work at all for other types of information. Of course, there are many other types of tables, some with multiple labels on rows, some with multiple labels on columns, some with more complex entries.

Another specialized language, explored in the early days of AI, is the semantic net. A *semantic net* is a directed graph with labeled nodes and arcs. The alphabet consists of all uppercase and lowercase letters, the digits, nodes, and directed arcs of arbitrary length and direction. The symbols of the language are the same as in predicate calculus, and they are divided into object constants and binary relation constants. A two-dimensional configuration of elements from this alphabet is a legal labeled directed graph if and only if each node has an associated object constant

Score	Quiz1	Quiz2	Quiz3	Final
Gauss	92	94	89	100
Herbrand	86	79	92	85
Laurent	52	70	45	68

Figure 2.5 Knowledge encoded in a binary table.

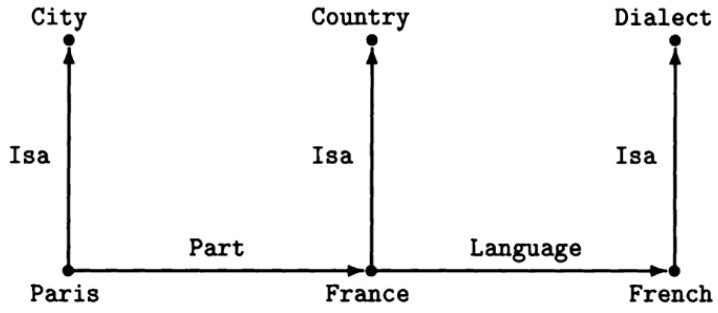


Figure 2.6 A semantic net.

written near it, each arc has an associated binary relation constant written near it, and each arc begins at a node and ends at a node. Figure 2.6 is an example of a semantic net, provided that the symbols `Isa`, `Part`, and `Language` are all binary relation constants and that the other symbols are all object constants.

An interpretation satisfies a semantic net if and only if the relation designated by the label on each arc holds of the objects designated by the labels on the nodes the arc connects. This particular semantic net is satisfied by the standard interpretation, since Paris is a city in France, France is a country, and the language in France is the French dialect.

A semantic net is particularly good for representing binary relations and, consequently, unary functions. Nonbinary relations can be handled by using arcs with more than two endpoints.

The language of frames is another language that has received considerable attention in the AI community, as much for its semantic richness (discussed later) as for its syntax. There are numerous frame languages and considerable variation in detail from one language to another. However, the following definition is consistent with the majority of these languages.

The alphabet of our frames language consists of uppercase and lowercase letters, digits, the colon character, and both horizontal and vertical lines. The symbols of the frames language are the same as those in predicate calculus and are divided into object constants, unary function constants, and binary relation constants. Each sentence is a structured object in the form of a frame (see Figure 2.7), where the symbol in the upper-left corner is an object constant, the symbols before the colons are function or relation constants, and the symbols after the colons are object constants. The sentences in the language are called *frames*; the symbol in the upper-left corner is the frame's *name*; the symbols before the colons are commonly termed *slots*; and the symbols after the colons are called *values*.

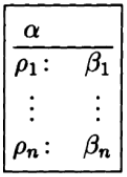


Figure 2.7 The general form of a frame.

An interpretation satisfies a sentence in the frames language if and only if the object designated by the value of each slot is the same as the object obtained by applying the function designated by the slot to the object designated by the frame name.

$$\langle \alpha^I, \beta_i^I \rangle \in \rho_i^I$$

Figure 2.8 shows two examples of knowledge encoded in frames. Jones is a freshman advised by Tversky and is in the psychology department. Tversky is a faculty member in the psychology department and advises Jones and Thorndyke.

One problem common to specialized languages such as tables, semantic nets, and frames is an inability to handle partial information. For example, there is no way in the table language to state the fact that either Herbrand or Laurent got a 90 on the first quiz without saying which of them did. There is no way in a semantic net to say that Paris is in some country without saying which one. There is no way in the frame language to say that Tversky is *not* Jones's advisor without saying who is.

In fairness to the language of semantic nets, it should be pointed out that various extensions have been proposed that allow one to express logical combinations of facts or quantified facts. However, these extensions compromise the simplicity of the language to a large extent.

In fairness to the frames language, it should be pointed out that the original idea of frames included provision for the association of procedural

Jones	
Isa:	Freshman
Dept:	Psychology
Advisor:	Tversky

Tversky	
Isa:	Faculty
Dept:	Psychology
Advisees:	{Jones,Thorndyke}

Figure 2.8 Knowledge encoded in frames.

knowledge with the declarative knowledge stored as slot values. This allows us to express knowledge beyond that we discussed. Unfortunately, it does not allow us to express this knowledge in declarative form.

In fairness to all these specialized languages, it should be noted that partial information always can be captured by defining new relations. For example, we can change the *score* function illustrated in Figure 2.5 to be a binary function from students and quizzes to *sets* of scores, with the idea that the actual score is a member of the set so designated. This would allow one to state that Herbrand got either an 80 or a 90 by recording the set {80,90} as his score. Representing other partial information is more difficult but still is possible. This approach has the disadvantage that the new conceptualizations are quite cumbersome; and, as a result, the specialized languages lose much of their perspicuity.

The language of predicate calculus addresses the problem of partial information head on by providing logical operators and quantifiers that allow us to express partial information. As a result, there is no need (in principle) either to encode declarative knowledge in procedural form or to change one's conceptualization of the world.

The primary disadvantage of predicate calculus is that it is less succinct than are specialized languages for many kinds of knowledge. On the other hand, no single specialized language is ideal for expressing all facts. For some kinds of information, tables are best. For other information, semantic nets or frames are best. For yet other information, bar graphs or pie charts or color or animation may be best.

Of course, we can easily define specialized languages such as tables, semantic nets, and frames in terms of predicate calculus. Having done so, we can use those languages where they are appropriate; where they fail, we can fall back on the expressive power of the full predicate calculus.

For these reasons, we have chosen to use predicate calculus in this text. The approach also has the pedagogical benefit of allowing us to compare and analyze different languages within a single common framework. Also, it is possible to describe inference procedures for just one language and have them apply automatically to these other languages as well.

2.10 Bibliographical and Historical Remarks

Although we stress in this book languages and reasoning methods for declarative representations of knowledge, the difficult problem for AI is the conceptualization of a domain. Every AI application begins with a particular conceptualization, and the reader should become familiar with several examples of these in order to gain an appreciation for this aspect of AI.

Conceptualizations used by expert systems usually are sharply limited to a small set of objects, functions, and relations. Typical examples are

those used by MYCIN [Shortliffe 1976], PROSPECTOR [Duda 1984], and DART [Genesereth 1984]. Inventing conceptualizations for broader domains that include common, everyday phenomena has proved quite difficult. Among the attempts to formalize commonsense knowledge are those of Hayes [Hayes 1985a] and those reported in [Hobbs 1985a, Hobbs 1985b]. The problem of grain size in conceptualizations has been studied by Hobbs [Hobbs 1985c]. Probably the most ambitious attempt to capture a large body of general knowledge in a representation that is independent of its many possible later uses is the CYC project of Lenat and colleagues [Lenat 1986].

Our treatment of the predicate calculus in this book is based on that of Enderton [Enderton 1972]. Other good textbooks on logic are those of Smullyan [Smullyan 1968] and Mendelson [Mendelson 1964]. A book by Pospesel [Pospesel 1976] is a good elementary introduction with many examples of English sentences represented as predicate-calculus sentences.

Semantic networks have a long tradition in AI and in cognitive psychology. In psychology, they have been used as models of memory organization [Quillian 1968, Anderson 1973]. In AI, they have been used more or less as a predicate-calculus–like declarative language [Simmons 1973, Hendrix 1979, Schubert 1976, Findler 1979, Duda 1978].

Closely related to semantic networks are languages that use frames. Following a key paper on frames by Minsky [Minsky 1975], several frame-based languages were developed, including KRL [Bobrow 1977, 1979, Lehnert 1979], FRL [Goldstein 1979], the UNITS [Stefik 1979], and KL-ONE [Brachman 1985c].

Comparisons between frames and semantic networks on the one hand and ordinary predicate calculus on the other have been discussed by Woods [Woods 1975], Brachman [Brachman 1979, 1983c], Hayes [Hayes 1979a], and Nilsson [Nilsson 1980, Chapter 9]. Although many versions of semantic networks do not have the full expressive power of first-order predicate calculus, they do carry extra indexing information that makes many types of inferences more efficient. (However, see [Stickel 1982, 1986, Walther 1985] for examples of how similar indexing can be achieved in implementations of systems that perform inferences on predicate-calculus expressions.) There also are relations between semantic network representations and so-called *object-oriented programming* methods [Stefik 1986]. Some representation systems have used semantic-network–style representations to express taxonomic information, and ordinary predicate calculus to express other information [Brachman 1983a, 1983b, 1985a].

For the same reasons that logical languages are important for representing information in AI programs, they also are attractive target languages into which to attempt translations of natural-language sentences in systems that perform natural-language processing. A volume edited by Grosz et al. contains several important papers on this topic [Grosz 1986].

Exercises

1. *Grain size.* Consider a conceptualization of the circuit in Figure 2.3 in which there are just six objects: the full adder and its five subcomponents. Devise a relational basis set that allows you to define the connectivity of the circuit.

2. *Reification.* Devise a conceptualization of the circuit in Figure 2.3 that allows you to express properties of connections, such as *broken* and *intermittent*.

3. *Syntax.* For each of the following examples, indicate whether or not it is a syntactically legal sentence in predicate calculus.

   a. `32456 > 32654`

   b. `32456 > France`

   c. `p ∨ q`

   d. `Likes(Arthur,France ∧ Switzerland)`

   e. `∀x Neighbor(France,Switzerland) ⇒ Prime(x)`

   f. `∀country Neighbor(France,country)`

   g. `∀x∃x Neighbor(x,x)`

   h. `(∀x P(x)) ⇒ (∃x P(x))`

   i. `(∀p p(A)) ⇒ (∃p p(A))`

   j. `(P(0) ∧ (∀x P(x) ⇒ P(x+1))) ⇒ (∀x P(x))`

4. *Groups.* Recall that a group is a set with a binary function and a distinguished element such that (a) the set is closed under the function, (b) the function is associative, (c) the distinguished element is an identity for the function, and (d) each element has an inverse. Express these properties as sentences in predicate calculus.

5. *Lists.* Write the axioms defining the function `Reverse`, whose value is the reverse of the list supplied as its argument.

6. *Translation.* Use the following vocabulary to express the assertions in the following sentences.

   - `Male(x)` means that the object denoted by `x` is male.
   - `Female(x)` means that `x` is female.
   - `Vegetarian(x)` means that `x` is a vegetarian.
   - `Butcher(x)` means that `x` is a butcher.

a. No man is both a butcher and a vegetarian.

b. All men except butchers like vegetarians.

c. The only vegetarian butchers are women.

d. No man likes a woman who is a vegetarian.

e. No woman likes a man who does not like all vegetarians.

7. *Reverse translation.* Translate the following predicate-calculus sentences into colloquial English. You can assume that all constants have their obvious meanings.

a. $\forall x\ \text{Hesitates}(x) \Rightarrow \text{Lost}(x)$

b. $\neg\exists x\ \text{Business}(x) \wedge \text{Like}(x, \text{Showbusiness})$

c. $\neg\forall x\ \text{Glitters}(x) \Rightarrow \text{Gold}(x)$

d. $\exists x \forall t\ \text{Person}(x) \wedge \text{Time}(t) \wedge \text{Canfool}(x,t)$

8. *Interpretation and satisfaction.* For each of the following sentences, give interpretations to the symbols such that the sentence makes sense and represents the world accurately (i.e., you believe it to be true).

a. $2 > 3$

b. $\neg P \Rightarrow \neg Q$

c. $\forall x \forall y \forall z\ R(x,y,z) \Rightarrow R(y,z,x)$

9. *Interpretation and satisfaction.* For each of the following three sentences, give an interpretation that makes that sentence false but makes the other two sentences true.

a. $P(x,y) \wedge P(y,z) \Rightarrow P(x,z)$

b. $P(x,y) \wedge P(y,x) \Rightarrow x=y$

c. $P(A,y) \Rightarrow P(x,B)$

10. *Satisfiability.* Say whether each of the following sentences is unsatisfiable, satisfiable, or valid.

a. $P \Rightarrow P$

b. $P \Rightarrow \neg P$

c. $\neg P \Rightarrow P$

d. $P \Leftrightarrow \neg P$

e. $P \Rightarrow (Q \Rightarrow P)$

11. *Definability.* Define the *above* relation in terms of the *on* relation, and define *on* in terms of *above*.

12. *Tables.* The tables language described in this chapter is ideal for expressing information about binary *functions.* Invent a table language appropriate for expressing binary *relations* and use it to encode the following information. Be sure that you can encode this information without changing the underlying conceptualization.

    a. The facts in Figure 2.6.

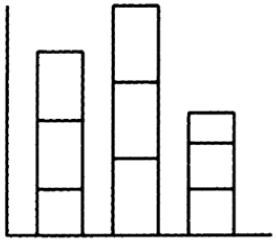
    b. The facts in Figure 2.8.

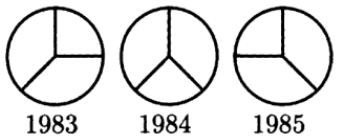
13. *Frames.* Consider the frames language defined in the text.

    a. Explain why you cannot express the facts in Figure 2.5 in this language without changing the underlying conceptualization.

    b. Express the facts in Figure 2.6 in the frames language.

14. *Pie charts and layered bar graphs.* The following illustrations are examples of the same knowledge encoded in two different languages. Both are good for expressing the relative proportions of a total quantity in a set of subcategories.





    a. What information expressed by the layered bar graph is not captured in the pie chart?

    b. Devise a graphical extension of the pie-chart language that allows us to express this additional information.

CHAPTER 3
Inference

INFERENCE IS THE PROCESS of deriving conclusions from premises. For example, from the premise that Art is either at home or at work and the premise that Art is not at home, we can conclude that he must be at work. The ability to perform inferences of this sort is an essential part of intelligence.

We begin this chapter with a discussion of inference and inference procedures in general. We then narrow our discussion by defining the criteria of soundness and completeness. Finally, we present a procedure that satisfies these criteria.

3.1 Derivability

Inference is typically a multistep process. In some cases, we can derive a conclusion from a set of premises in a single step. In other cases, we first need to derive intermediate conclusions.

Each step in this process must be sanctioned by an acceptable *rule of inference.* A rule of inference consists of (1) a set of sentence patterns called *conditions*, and (2) another set of sentence patterns called *conclusions.* Whenever we have sentences that *match* the conditions of the rule, then it is acceptable to infer sentences matching the conclusions.

Modus ponens (MP) is an example. The sentence patterns above the line in the following display are the conditions, and the sentence pattern below the line is the sole conclusion. The significance of this rule is that,

whenever sentences of the form $\phi \Rightarrow \psi$ and ϕ have been established, then it is acceptable to infer the sentence ψ as well.

$$\frac{\phi \Rightarrow \psi \quad \phi}{\psi}$$

Suppose, for example, that we believe the sentence `On(A,B)` and we also believe the sentence `On(A,B)` $\Rightarrow$ `Above(A,B)`. Then, modus ponens allows us to infer the sentence `Above(A,B)` in a single step.

Modus tolens (MT) is the reverse of modus ponens. If we believe that ϕ implies ψ and we believe that ψ is false, then we can infer that ϕ must be false as well.

$$\frac{\phi \Rightarrow \psi \quad \neg\psi}{\neg\phi}$$

And elimination (AE) states that, whenever we believe a conjunction of sentences, then we can infer each of the conjuncts. In this case, note that there are multiple conclusions.

$$\frac{\phi \wedge \psi}{\begin{array}{l}\phi \\ \psi\end{array}}$$

And introduction (AI) states that, whenever we believe some sentences, we can infer their conjunction.

$$\frac{\phi \quad \psi}{\phi \wedge \psi}$$

Universal instantiation (UI) allows us to reason from the general to the particular. It states that, whenever we believe a universally quantified sentence, we can infer an instance of that sentence in which the universally quantified variable is replaced by any appropriate term.

$$\frac{\forall \nu\ \phi}{\phi_{\nu/\tau} \quad \text{where } \tau \text{ is free for } \nu \text{ in } \phi}$$

For example, consider the sentence $\forall$`y Hates(Jane,y)`. From this premise, we can infer that Jane hates Jill; i.e., `Hates(Jane,Jill)`. We also can infer that Jane hates herself; i.e., `Hates(Jane,Jane)`. We can even infer than Jane hates her mother; i.e., `Hates(Jane,Mom(Jane))`.

In addition, we can use universal instantiation to create conclusions with free variables. For example, from $\forall$`y Hates(Jane,y)`, we can infer

A set of sentences Γ *logically implies* (synonymously, *logically entails*) a sentence ϕ (written $\Gamma \models \phi$) if and only if every interpretation and variable assignment that satisfies the sentences in Γ also satisfies ϕ. That is, $\Gamma \models \phi$ if and only if $\models_I \Gamma[U]$ implies $\models_I \phi[U]$ for all I and U. A set of closed sentences Γ logically entails a closed sentence ϕ if and only if every interpretation that satisfies the sentences in Γ also satisfies ϕ.

Consider the set of closed sentences that follows. These sentences logically imply the sentence `Above(A,B)`. Any interpretation that satisfies these sentences also satisfies `Above(A,B)`.

```
∀x∀y  On(x,y) ⇒ Above(x,y)

On(A,B)
```

For example, under the intended interpretation for these symbols in our standard Blocks World example (see Figure 2.1), the sentences are clearly satisfied. The first sentence is a general property of the *on* and *above* relations. The second sentence is satisfied in this situation, because block *a* is on block *b*. The interpretation satisfies `Above(A,B)`, because block *a* is also above block *b*.

We could try to construct a counterexample by finding an interpretation that satisfies the premises but does not satisfy the conclusion. For example, we might try an interpretation that maps `On` into the *under* relation and that maps `Above` into the *below* relation. Under this interpretation, `Above(A,B)` clearly is not satisfied, because *a* is not below *b*. The first sentence in the set is satisfied, because *under* implies *below*. Unfortunately, the second sentence in the set is not satisfied, because *a* is in not immediately beneath *b*. Since this interpretation does not satisfy all the sentences in the set, it is not a counterexample.

Given the notion of logical implication, we can define our criteria for evaluating inference procedures. We say that an inference procedure is *sound* if and only if any sentence that can be derived from a database using that procedure is logically implied by that database. We say that an inference procedure is *complete* if and only if any sentence logically implied by a database can be derived using that procedure. Although the procedures presented in the previous section are sound, none are complete. In the next section, we sketch a procedure that is both sound and complete, although somewhat impractical. In the next two chapters, we discuss a more practical procedure that is also sound and complete.

A *theory* is a set of sentences closed under logical implication. Since there are infinitely many conclusions from any set of sentences, a theory is necessarily infinite in extent. A theory $\mathcal{T}$ is *complete* if and only if, for every sentence ϕ, either ϕ or its negation is a member of $\mathcal{T}$.

3.4 Provability

One apparent difficulty with the practical use of logical implication as a criterion for inferential correctness is the infinity lurking in its definition. The definition in the previous section states that a database of sentences Δ logically implies a sentence ϕ if and only if every interpretation that satisfies Δ also satisfies ϕ. The problem is that the number of interpretations of any set of sentences is infinite, so there is no way to check them all in a finite amount of time.

Fortunately, the situation is not hopeless. An important theorem of mathematical logic states that whenever Δ logically implies ϕ, there is a finite "proof" of ϕ from Δ. Therefore, the question of determining logical implication is reduced to the problem of finding such a proof. As it turns out, there is a procedure for enumerating legal proofs; thus, if Δ logically implies ϕ, we can verify it in a finite amount of time.

A *proof* of a sentence ϕ from a database Δ is a finite sequence of sentences in which (1) ϕ is an element of the sequence (usually the last) and (2) every element is a member of Δ, a logical axiom, or the result of applying modus ponens to sentences earlier in the sequence. Note that we allow only one rule of inference in our definition. Thus, a proof is like a derivation, except that we include logical axioms and we use only one rule of inference. As we shall see, if we include enough logical axioms, we can ignore all other rules of inference.

A *logical axiom* is a sentence that is satisfied by all interpretations purely because of its logical form. By adding some basic logical axioms to our set of premises (which latter are called *nonlogical axioms* or *proper axioms*), we can derive conclusions that we cannot derive using modus ponens alone.

Although the number of basic logical axioms is infinite, it is possible to describe the axioms with a finite number of *axiom schemata*. An axiom schema is a sentence pattern with pattern variables (written here as Greek letters) that range over all legal sentences. Each schema denotes the set of all sentences that either match its pattern or are *generalizations* of its pattern, where a generalization of a sentence ϕ is a sentence of the form $\forall \nu \ \phi$.

The *implication introduction* schema (II), together with modus ponens, allows us to infer implications.

$$\phi \Rightarrow (\psi \Rightarrow \phi)$$

The following sentences are all instances of this schema. In the first sentence, ϕ is `P(x)` and ψ is `Q(y)`. In the second sentence, ϕ is the nonatomic sentence `P(x) ⇒ R(x)`. The last three sentences are generalizations of the second sentence.

```
P(x) ⇒ (Q(y) ⇒ P(x))
(P(x) ⇒ R(x)) ⇒ (Q(y) ⇒ (P(x) ⇒ R(x)))
```

probability values for sentences. It is helpful to begin our discussion, however, on more intuitive grounds.

Consider two ground atoms P and Q. If the probabilities of P and Q are given, what can be said about the probability of $P \wedge Q$? It all depends on the *joint distribution* of P and Q. It will turn out that what might be called a *probabilistic interpretation* for a set of sentences will involve something like an assignment of a joint probability distribution to the ground instances of the atoms in those sentences. An interpretation for the set of sentences $\{P, Q\}$ consists of a joint probability distribution for P and Q. That is, we must specify probabilities for the four combinations of each of P and Q being true and false.

For purposes of discussion, let the four joint probabilities in this example be given by

$$\begin{aligned} p(P \wedge Q) &= p_1 \\ p(P \wedge \neg Q) &= p_2 \\ p(\neg P \wedge Q) &= p_3 \\ p(\neg P \wedge \neg Q) &= p_4 \end{aligned}$$

where $p(\phi)$ denotes the probability of the formula ϕ being true.

The probabilities of P and Q alone are called *marginal probabilities* and are given as sums of the joint probabilities as follows:

$$\begin{aligned} p(P) &= p_1 + p_2 \\ p(Q) &= p_1 + p_3 \end{aligned}$$

We see that merely specifying probabilities (generalized truth values) for P and Q individually does not fully determine the four joint probabilities, and therefore (unlike ordinary logic) it does not allow us to calculate probabilities (generalized truth values) for composite formulas such as $P \wedge Q$.

In ordinary logic, modus ponens allows us to conclude Q, given P and $P \Rightarrow Q$. In probabilistic logic, however, we cannot analogously calculate a probability for Q given probabilities for P and $P \Rightarrow Q$, because these probabilities do not fully determine the four joint probabilities. This lack of analogous inference rules makes reasoning with uncertain beliefs more complex than reasoning with certain beliefs is. Joint probability distributions over n atoms have 2^n component terms—an impractically large number, even for small numbers of atoms. Nevertheless, there are some reasoning procedures for uncertain beliefs that under restricted circumstances produce intuitively satisfactory results, and we shall be discussing some of these in this chapter.

8.2 Using Bayes' Rule in Uncertain Reasoning

In some situations involving uncertain beliefs, we can perform a reasoning step somewhat similar to modus ponens while using probability information that is available. Suppose we want to calculate the probability of Q when we know that P is true and we also know some information about the relationship between P and Q. The probability of Q given that P is true, written $p(Q|P)$ and called the *conditional probability* of Q given P, is just that fraction of the cases in which P is true for which Q also is true. In terms of the joint probabilities defined previously, this fraction is $p_1/(p_1 + p_2)$. Or, $p(Q|P) = p(P,Q)/p(P)$, where $p(P,Q)$ stands for the probability of both P and Q being true (which is the same as $p(P \wedge Q)$.)

Similarly, we could calculate $p(P|Q) = p(P,Q)/p(Q)$. Combining these two expressions allows us to write

$$p(Q|P) = \frac{p(P|Q)p(Q)}{p(P)}$$

which is known as *Bayes' rule*. $p(Q|P)$ is called the *conditional* or *posterior* probability of Q given P, and (in this context) $p(Q)$ and $p(P)$ are called the *prior* or *marginal* probabilities of Q and P, respectively. The importance of Bayes' rule for uncertain reasoning lies in the facts that (1) one often has (or can reasonably assume) prior probabilities for P and Q, and (2) in situations in which some evidence P bears on a hypothesis Q, one's knowledge of the relationship between P and Q often is available in terms of $p(P|Q)$. From these given quantities, the essential *reasoning* step involves using Bayes' rule to calculate $p(Q|P)$.

An example will help to clarify the use of Bayes' rule in uncertain reasoning. Suppose P stands for the sentence, "The automobile's wheels make a squeaking noise," and Q stands for the sentence, "The automobile's brakes need adjustment." We would ordinarily think of P as a symptom and of Q as a hypothesis about the cause of the symptom. The relationship between cause and symptom usually can be expressed in terms of the probability that the symptom occurs given the cause, or $p(\mathtt{P}|\mathtt{Q})$. Let us suppose that poorly adjusted brakes often (but not always) produce wheel squeaks, say $p(\mathtt{P}|\mathtt{Q}) = 0.7$. Suppose, to further specify our example, that $p(\mathtt{P}) = 0.05$ and $p(\mathtt{Q}) = 0.02$. If we are in a situation in which we observe squeaking wheels and want to determine the probability that the brakes need adjustment, then we calculate $p(\mathtt{Q}|\mathtt{P}) = 0.28$ using Bayes' rule. Many instances of reasoning with uncertain information are analogous to this example, in which we have information about "symptoms" and want to infer "causes."

To use Bayes' rule, we must have a value for $p(P)$. In practice, prior probabilities of "symptoms" are often much more difficult to estimate than are prior probabilities of "causes," so it is worth inquiring whether or not Bayes' rule can be expressed in terms of quantities that are easier

to obtain. Fortunately, there is another version of Bayes' rule in which $p(P)$ does not occur. To derive this version, we first observe that, although $p(\neg Q|P) = 1 - p(Q|P)$, the expression also can be written using Bayes' rule as follows:

$$p(\neg Q|P) = \frac{p(P|\neg Q)p(\neg Q)}{p(P)}$$

Dividing the Bayes' rule expression for $p(Q|P)$ by that of $p(\neg Q|P)$ yields

$$\frac{p(Q|P)}{p(\neg Q|P)} = \frac{p(P|Q)p(Q)}{p(P|\neg Q)p(\neg Q)}$$

The probability of an event divided by the probability that the event does not occur usually is called the *odds* of that event. If we denote the odds of E by $O(E)$, we have $O(E) =_{\text{def}} p(E)/p(\neg E) = p(E)/(1 - p(E))$. Using this notation, we rewrite the quotient of the two Bayes' rule expressions as

$$O(Q|P) = \frac{p(P|Q)}{p(P|\neg Q)} O(Q)$$

The remaining fraction in this expression is an important statistical quantity, usually called the *likelihood ratio* of P with respect to Q. We will denote it here by λ. Thus,

$$\lambda =_{\text{def}} \frac{p(P|Q)}{p(P|\neg Q)}$$

The *odds-likelihood* formulation of Bayes' rule is now written as

$$O(Q|P) = \lambda O(Q)$$

This formula has a satisfying intuitive explanation. It tells us how to compute posterior odds on Q (given P) from the prior odds on Q (the odds that applied before we observed that P was true). Upon learning that P is true, to update the strength of our belief in Q (measured in terms of its odds), we simply multiply the previous odds by λ. The ratio λ can be thought of as information about how influential P is in helping to convert indifferent odds on Q to high odds on Q. When λ is equal to one, then knowing that P is true has absolutely no effect on the odds of Q. In that case, Q is independent of P being true. Values of λ less than one depress the odds of Q; values of λ greater than one increase the odds of Q. Note that, even though we have expressed Bayes' rule in terms of odds, we can easily recover the underlying probability by the formula:

$$p(Q) = O(Q)/(O(Q) + 1)$$

Knowledge about how causes and symptoms are related often can be expressed conveniently by estimating values for the appropriate λs. Even when people knowledgeable about these relations might not be

Index