CHAPTER 2
The Representation of Knowledge

2.1 INTRODUCTION

This chapter and the next are about **logic**. Most people think of logic as meaning the same as reasonable. That is, if someone is logical, they are reasonable and don't jump to conclusions. However we need to define these terms more closely in order to work with AI and expert systems based on logic. As anyone who's used computers knows, the advantage of computers is that they do exactly as told. And the disadvantage of computers is that they do exactly as told. Since expert systems now determine your credit rating, whether you will be audited by the IRS, the operation of nuclear power plants, and other events of similar importance, it's a good idea to make them very, very logical and not subject to ambiguities. In technical terms, logic is the study of making valid inferences. This means that given a set of true facts, the conclusion will always be true. An invalid inference means that a false conclusion is reached with true facts. (So if you're ever stopped for speeding, simply show this book to the police officer.)

We also want to make clear the distinction between **formal logic** and **informal logic**. Informal logic is the kind of logic people and especially lawyers use when trying to win an argument, i.e., a case. An argument is not necessarily a heated exchange that erupts in gunfire, but a legal argument in a courtroom where emotionally charged words are used to persuade the jury which side has the best lawyer and so win the case. A complex logical argument is a chain of inferences in which one conclusion leads to another and so on. In court this may lead to one of several conclusions until the verdict of Guilty, or Not Guilty, or Not Guilty by Reason of Insanity, or a mistrial is declared, and then the case is appealed.

In **formal logic**, also called **symbolic logic**, the inferences and other factors which enter into proving the final conclusion true or false in a valid way are of utmost importance. If you've ever encountered a bug in a computer program, that is a perfect example of the program making an invalid symbolic inference. (Fortunately, if you have spent hundreds of dollars to upgrade your operating system to the latest version, you can now always get immediate satisfaction by sending an error report to the manufacturer.) Logic also needs **semantics** to give meaning to symbols. In formal logic we choose semantics that are not emotionally charged words like "Do you prefer Pepsi or Coke?" Instead the semantics of formal logic are like choosing meaningful names for variables in ordinary programming.

In addition to introducing logic, this chapter is also an introduction to the commonly used representations of knowledge. Knowledge representation (KR) has long been considered central to AI because it is as significant a factor in determining the success of a system as the software that uses the knowledge. This importance also carries over to computer science in the field of database design. While databases are generally considered as repositories of current data, such as a store's product inventory, accounts payable, accounts due, etc., rather than knowledge, many companies are now actively engaged in **data mining** to extract knowledge.

Data mining is meant to use **archival data** stored in **data warehouses** in order to predict future trends. For example, a company may look at its last five years of sales reports for the month of December to predict what kind and how much of a certain inventory to keep in stock, For example, they may discover through data mining that Christmas cards sell well in December but Valentine's Day cards do not. A slightly more realistic case would be the discovery that red and green clothes sell better in Winter than Spring since red and green are associated with Christmas, while brown, orange and yellow clothes sell better in the Fall. While human managers may be aware of this, data mining can serve to provide quantitative estimates of how much clothes to buy and when to put them on sale once the season is over. Of course the real value of data mining is in discovering patterns that are not obvious to a human but may be discovered by analyzing huge amounts of historical data that is stored in a company's data archive. In addition to classical statistical methods, AI techniques such as ANS, Genetic algorithms, Evolutionary algorithms, and expert systems may all be used with data mining, either singly or in hybrid combinations (Werbos 94).

KR is of major importance in expert systems for two reasons. First, expert systems are designed for a certain type of knowledge representation based on **rules of logic** called **inferences**. Normally we think of **reasoning** as drawing conclusions from facts. Unfortunately people are not very good at this because we tend to mix up the semantics with the reasoning process itself, and so do not always reach a valid conclusion. Good examples of this occur in ads for political races. The method of reasoning that uses no facts, unreliable facts, or the same facts to reach completely opposite conclusions is the basis of political logic.

Making inferences is the formal term for a special type of reasoning that does not rely on semantics; that is the meaning of words. While semantics are indispensable in the real world, an expert system is designed to reason on the basis of logic and not get involved in the emotional entanglements that semantics may bring into reasoning. The object of inferences is to reach a valid conclusion based on the facts and form of the argument. Here again, in Logic, the term *argument* means the formal way that facts and rules of inference are made to justify a valid conclusion. If you've ever watched news shows in which the newscaster engages in a long discussion with someone who is a former general, former ambassador, former juror, or whatever, the purpose is not to reach a valid conclusion but to keep the TV viewer entertained.

Making valid inferences is logical reasoning. While the terms *commonsense* and *probabilistic reasoning* are invaluable in the real world, they involve uncertainty because things in the real world are never 100 percent certain. If someone says, "The sky is blue," a little while later it may be grey or even worse, green! Reasoning under uncertainty is a very important subject that we will discuss in more detail in Chapters 4 and 5.

The second reason KR is important is it affects the development, efficiency, speed, and maintenance of the system. This is just like the situation in ordinary programming where the choice of data structure is of fundamental importance to the program. A good program design must choose from simple named variables, arrays, linked lists, queues, trees, graphs, networks or even stand-alone external databases such as Microsoft Access, SQL Server, or Oracle. In CLIPS, the KR can be rules, deftemplates, objects, and facts.

In the next chapter we will discuss how inferences are made on knowledge to produce valid conclusions, and common fallacies you should watch out for that may appear logical but are not. This is especially important in knowledge acquisition discussed in Chapter 6 when you have to interview a human expert for your expert system. You will have to identify true knowledge from the semantics that can lead to invalid conclusions. On the other hand, if you argue too much with your knowledge expert and don't come up with the conclusions they want, you may get fired!

Many computer programs have been written using AI for reasoning, theorem proving, and even learning logic as shown in Appendix G.

2.2 THE MEANING OF KNOWLEDGE

Knowledge, like love, is one of those words that everyone knows the meaning of, yet finds hard to define or even feel the same way about. In fact, the only true love that all people feel the same about is for their cat or dog (with apologies for those who love snakes and tarantulas.) Like love, knowledge has many meanings depending on the mind of the beholder. Other words such as data, facts, and information are often used interchangeably with knowledge (Russell 03).

People may seek solutions to problems through reasoning and experience. The general term for using experience to solve a problem is **heuristics**. Long, specific examples of heuristic experience are called **case-based reasoning**. This is one of the main types of reasoning used in law, medicine, and repairs where lawyers, doctors, and auto mechanics try to repair a problem using previous similar cases, called **precedents** (Leake 96). While you may feel a case-based solution is expensive since someone's already had your problem, see how much it costs to establish a new medical, legal, or auto repair precedent!

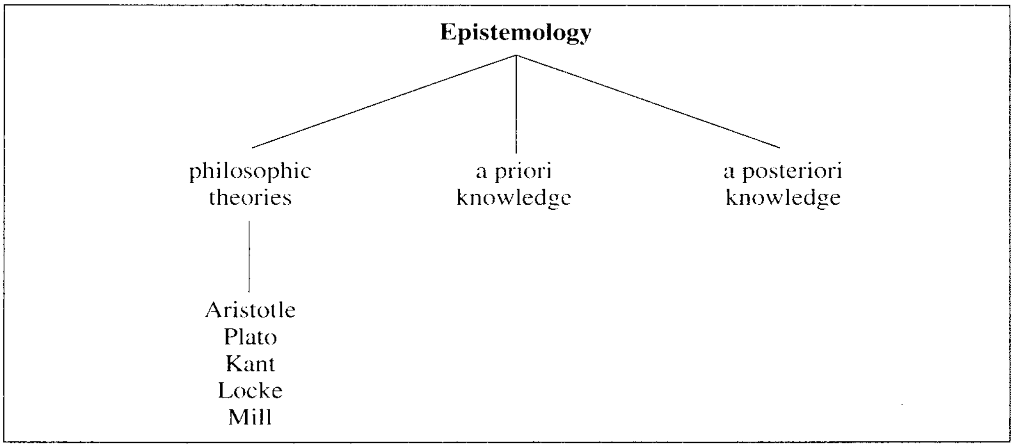
An expert system may have hundreds or thousands of little cases to refer to. Each rule may be thought of as a micro-precedent which may be able to solve the problem or be used in a chain of inference that will hopefully provide a solution.

The study of knowledge is **epistemology**. It is concerned with the nature, structure, and origins of knowledge. Figure 2.1 illustrates some of the categories of epistemology. Besides the philosophical kinds of knowledge expressed by Aristotle, Plato, Descartes, Hume, Kant, and others, there are two special types called **a priori** and **a posteriori**. The Latin term *a priori* means *that which precedes*. A priori knowledge comes before and is independent of knowledge from the senses. For example, the statements "everything has a cause" and "all triangles have 180 degrees in the Euclidean plane" are examples of a priori knowledge. A priori knowledge is considered to be universally true and cannot be denied without contradiction. Logic statements, mathematical laws, and the knowledge possessed by teenagers are examples of a priori knowledge.

The opposite of a priori knowledge is knowledge derived from the senses: *a posteriori* knowledge. The truth or falsity of a posteriori knowledge can be verified using sensory experience, as in the statement "the light is green." However, since sensory experience may not always be reliable, a posteriori knowledge can be denied on the basis of new knowledge without the necessity of contradictions. For example, if you see someone with brown eyes, you would believe that their eyes were brown. However, if you later saw that person removing brown contact lenses to reveal blue eyes, your knowledge would have to be revised.

Knowledge can be further classified into **procedural knowledge**, **declarative knowledge**, and **tacit knowledge**. The procedural and declarative knowledge types correspond to the procedural and declarative paradigms discussed in Chapter 1 and in more depth in (Brewka 97).

Figure 2.1 Some Categories of Epistemology



Procedural knowledge is often referred to as knowing how to do something. An example of procedural knowledge is knowing how to boil a pot of water. Thinking you know how to boil water at all elevations can lead to a failure of commonsense reasoning; a true oxymoron since there's no reasoning in commonsense but real world experience (see the problem at the end of the chapter on boiling water). Declarative knowledge refers to knowing that something is true or false. It is concerned with knowledge expressed in the form of declarative statements such as "Don't put your fingers in a pot of boiling water." Many different ways of representing knowledge using logic have been developed, see (Sowa 00)

Tacit knowledge is sometimes called **unconscious knowledge** because it cannot be expressed by language. An example is knowing how to move your hand. On a gross scale, you might say that you move your hand by tightening or relaxing certain muscles and tendons. But at the next lower level, how do you know how to tighten or relax the muscles and tendons? Other examples are walking or riding a bicycle. In computer systems ANS is related to tacit knowledge because normally the neural net cannot directly explain its knowledge, but may be able to if given an appropriate program (see Section 1.14).

Knowledge is of primary importance in expert systems. In fact, an analogy to Nicholas Wirth's classic expression used in his seminal book on Pascal:

```
Algorithms + Data Structures = Programs
```

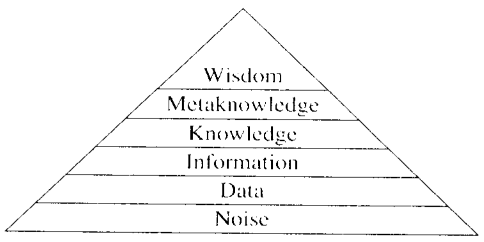
is the following for expert systems:

```
Knowledge + Inference = Expert Systems
```

As used in this book and illustrated in Figure 2.2, knowledge is part of a hierarchy. At the bottom is noise, consisting of items that are of no interest and may obscure data. The next higher level is raw data, which are items of potential interest. Next higher is information, which is processed data of definite interest. Next up is knowledge, which represents such important information that it should be preserved and acted upon. In Chapter 1, knowledge in rule-based expert systems is defined as the rules that were activated by facts or other rules to produce new facts or conclusions. A conclusion is the end product of a chain of reasoning, called inferencing when done according to formal rules. This process of making inferences is an essential part of an expert system. The term **inferencing** is generally used for mechanical systems such as expert systems. Reasoning is the term used for human thinking.

An ANS does not make inferences. Rather it seeks to find an underlying pattern to data that is not obvious to a human. Basically a ANS is a pattern classifier. For example, your ability to read is based on pattern recognition of a neural net in your brain that has been trained to recognize the pattern of letters. Another part of your brain translates those patterns into the words that people hear in their minds when reading because this is how they were taught to read as children: by sounding out the words in a phonetic manner. As an example, turn this book upside down and try to read it. Most people cannot read upside down at first but it is possible to retrain your reading neural net to recognize letters. In

Figure 2.2 The Pyramid of Knowledge



Wisdom: Using knowledge in a beneficial way
Metaknowledge: Rules about knowledge
Knowledge: Rules about using information
Information: Potentially useful for knowledge
Data: Potentially useful information
Noise: No apparent information.

classical psychological experiments, people wore glasses that inverted the image of the world. After a few days their brains adapted and things looked right-side-up again. In fact the lens of our eyes project an image upside down and our brain inverts this to make it seem normal.

Another example of the versatility of neural nets is to rotate the book by an angle, say 30° (Saratchandran 96). You will still be able to read although a bit more slowly. As you rotate the book towards 180°, you will have a harder time reading. With practice, some people can read as well upside down which shows the amazing adaptability of our neural nets. In the same way, a ANS that is trained on letters at different angles will be able to interpolate and read text that is not shown normal. The more rotations that the net is trained on, the faster will be its accuracy. Likewise, training a net on different cursive styles will enable a ANS to read more styles of handwriting just as a person can read different writers' handwriting.

The term **facts** mean information that is considered reliable. Expert systems draw inferences using facts. Facts that are later shown to be false may be retracted using the Truth Maintenance facility of CLIPS and all conclusions, rules, and other facts generated by the false fact are automatically retracted. Expert systems may also (1) separate data from noise, (2) transform data into information, or (3) transform information into knowledge. It is extremely dangerous to use raw data in an expert system that expects facts since the reliability of the resulting conclusions may be highly unreliable. This is, of course, another way of saying the adage, "Garbage in, garbage out." Unless of course someone wants garbage to support a particular agenda.

As an example of these concepts, consider the following sequence of 24 numbers:

```
137178766832525156430015
```

Without knowledge, this entire sequence may appear to be noise. However, if it is suspected or known that this sequence is meaningful, then the sequence is data. Determining what is data and what is noise is like the old saying about gardening: "a weed is anything that grows which isn't what you want."

Certain knowledge may exist to transform data into information. For example, the following algorithm processes the data to yield information:

```
Group the numbers by twos.
Ignore any two-digit numbers less than 32.
Substitute the ASCII characters for the two-digit
numbers.
```

Application of this algorithm to the previous 24 numbers yields the information:

```
GOLD 438+
```

Now knowledge can be applied to this information. For example, there may be a rule:

```
IF gold is less than 500
   and the price is rising (+)
THEN
   buy gold
```

Although not explicitly shown in the figure, **expertise** is a specialized type of knowledge and skill that experts have. It is in the knowledge, metaknowledge, and wisdom sections of Figure 2.2. Although very specialized knowledge may be found in public sources of information such as books and papers, just reading the book does not an expert make. For example, detailed information about surgical procedures may be found in a medical textbook. Yet would you be willing to undergo brain surgery if someone came to your door claiming they had taken an online course and was willing to do it at a cut-rate price? Even if you get a free set of Ginsu knifes (slightly used from your brain surgery?)

Expertise is the implicit knowledge and skills of the expert that must be extracted and made explicit so that it can be encoded in an expert system. The reason the knowledge is implicit is that a true expert knows the knowledge so well that it is second-nature and does not require thinking. As an example, after graduating from medical school, interns serve a year typically working 80 or more hours per week until they can practice medicine without even thinking. While this practice has been criticized, it does embed the knowledge so deeply it becomes second nature. (Of course if you're a patient, it's also a good idea to ask the intern how much sleep they've gotten this week.) Near the top of the hierarchy, above knowledge, is **metaknowledge.** The prefix **meta** means "above."

Metaknowledge is knowledge about knowledge and expertise. Although an expert system may be designed with knowledge about several different domains, this is generally undesirable because the system becomes less well-defined. Experience has shown that the most successful expert systems are restricted to as small a domain as possible. For example, if an expert system was designed to identify bacterial diseases, it would not be good to have it also diagnose car problems. As a real world example, consider that doctors specialize in only one small area, not all of medicine. Even a family physician (who used to be called a general practitioner) refers patients to the appropriate specialist when needed.

In expert systems, an ontology is the metaknowledge that describes everything known about the problem domain. Ideally an ontology should be described in a formal manner so that inconsistencies and inadequacies can be easily identified. A number of free and commercial tools are available to construct ontologies. Constructing an ontology should be done before the expert system is implemented or else the rules may have to be revised as more information is known about the domain, thus increasing the cost, development time and bugs.

For example, an expert system could have knowledge bases about repairing GM cars, GM SUVs, and GM diesel trucks. Depending on which type of vehicle needed repair, the appropriate knowledge base would be used. It would be inefficient in terms of memory and speed for all the knowledge bases to reside in memory at once since the Rete network continuously modifies the network of all rules in memory. In addition, there could be conflicts if the antecedents of a rule for truck and car were the same pattern, yet the conclusion was different. For example, if the fuel gauge read empty, the car expert system might say "Fill tank with gas," while the truck expert system might say, "Fill tank with diesel." It would not be good to fill the car's gas tank with diesel or the truck's with gas. The expert system also slows down as the number of rules in the system increases since the Rete network gets larger. Metaknowledge can be used to decide which knowledge domain should be loaded into memory and also as a general guide to designing and maintaining the expert system and ontology.

In a philosophical sense, **wisdom** is the peak of all knowledge. Wisdom is the metaknowledge of determining the best goals of life and how to obtain them. A rule of wisdom might be expressed as:

```
IF I have enough money to keep my spouse happy
THEN I will retire and enjoy life
```

The subject of AI-based wisdom engineering has been in constant growth. However, while there is a great abundance of wisdom in the world there are few listeners. In this book we shall restrict ourselves to knowledge-based systems and leave wisdom-based systems to politicians and other experts.

2.3 PRODUCTIONS

A number of different knowledge-representation techniques have been devised. These include rules, semantic nets, frames, scripts, logic, conceptual graphs, and others. In particular, many knowledge representation languages have been proposed such as the classic knowledge language one, KL-ONE, and its frame-based descendant, CLASSIC (Brachman 91). Many others have been proposed including visual-based languages.

As described in Chapter 1, rules are commonly used as the knowledge base in expert systems since their advantages greatly outweigh their disadvantages.

One formal notation for defining productions is the Backus Naur-form (BNF). This notation is a **metalanguage** for defining the **syntax** of a language. Syntax defines the form, while **semantics** refers to meaning. A metalanguage is a language for describing languages. Since the prefix meta means above, a metalanguage is above normal language.

There are many types of languages such as natural languages, logic languages, mathematics, and computer languages. The BNF notation for a simple English language rule that a sentence consists of a noun and a verb followed by ending punctuation is given as the following production rule:

```
<sentence>::= <subject> <verb> <end-mark>
```

where the **angle brackets,** <>, and ::= are symbols of the metalanguage and not of the language being specified. The symbol ::= means "is defined as" and is the BNF equivalent of the arrow, →, used in Chapter 1 with production rules.

The terms within angle brackets are called **nonterminal symbols** or simply *nonterminals*. A nonterminal is a variable that represents another term. The other term may be a nonterminal or a **terminal**. A terminal cannot be replaced by anything else and so is a constant.

The nonterminal <sentence> is special because it is the one **start symbol** from which the other symbols are defined. In the definition of programming languages, the start symbol is usually named <program>. The production rule:

```
<sentence> → <subject> <verb> <end-mark>
```

states that a sentence is composed of a subject, followed by a verb, followed by an end-mark. The following rules complete the nonterminals by specifying their possible terminals. The **bar** means "or" in the metalanguage:

```
<subject> → I | You | We
<verb> → left | came
<end-mark> → . | ? | !
```

All possible sentences in the language, i.e., the productions, can be produced by successively replacing each nonterminal by its right-hand-side nonterminals or terminals until all nonterminals are eliminated. The following are some productions:

```
I left.
I left?
I left!
```

```
You left.
You left?
You left!
We left.
We left?
We left!
```

A set of terminals is called a **string** of the language. If the string can be derived from the start symbol by replacing nonterminals with their definition rules, then the string is called a valid **sentence**. For example, "We," "WeWe," and "leftcamecame" are all valid strings of the language, but are not valid sentences.

A **grammar** is a complete set of production rules that defines a language unambiguously. While the previous rules do define a grammar, it is a very restricted one because there are so few possible productions. For example, a more elaborate grammar could also include direct objects, as in the following productions:

```
<sentence> → <subject> <verb> <object> <end-mark>
<object> → home | work | school
```

Although this is a valid grammar, it is too simple for practical use. A more practical grammar is the following where the end-mark has been left out for simplicity:

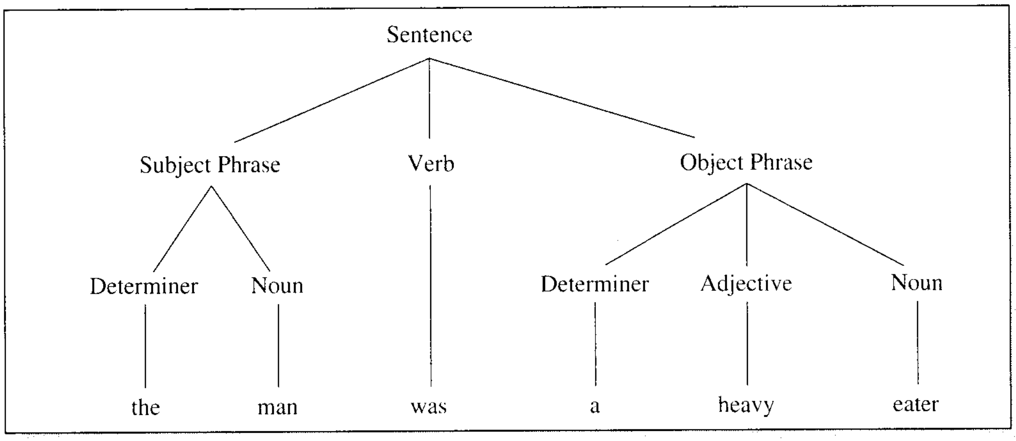
```
<sentence> → <subject phrase> <verb> <object
  phrase>
<subject phrase> → <determiner> <noun>
<object phrase> → <determiner> <adjective> <noun>
<determiner> → a | an | the | this | these | those
<noun> → man | eater
<verb> → is | was
<adjective> → dessert | heavy
```

The <determiner> is used to indicate a specific item. Using this grammar, we can generate sentences such as:

```
the man was a dessert eater
an eater was the heavy man
```

A **parse tree** or **derivation tree** is a graphic representation of a sentence decomposed into al! the terminals and nonterminals used to derive the sentence. Figure 2.3 shows the parse tree for the sentence "the man was a heavy eater." However, the string "man was a heavy eater" is not a valid sentence because it lacks the determiner in the subject phrase. A compiler creates a parse tree when it tries to determine if statements in a program conform to the valid syntax of a language.

Figure 2.3 Parse Tree of a Sentence



The tree in Figure 2.3 shows that the sentence "the man was a heavy eater" can be derived from the start symbol by applying appropriate productions. The steps in this process are shown following; where the **double arrow, =>**, means apply the productions shown:

```
<sentence> => <subject phrase> <verb>
              <object phrase>
<subject phrase> => <determiner> <noun>
<determiner> => the
<noun> => man
<verb> => was
<object phrase> => <determiner> <adjective> <noun>
<determiner> => a
<adjective> => heavy
<noun> => eater
```

An alternative way of using productions is to generate valid sentences by substituting all the appropriate terminals for nonterminals, as discussed previously. Of course, not all productions, such as "the man was the dessert," make sense (unless you're a cannibal).

Finite state machines (FSMs) are well suited to recognizing sentence structure. For example, compilers that translate source code of computer languages use FSM to **parse** source code into the smallest units of meaning, called **tokens.** FSM and parsing are essential to applications such as compilers to translate source code into assembly language, and accurate speech recogination. The term **compiler** has now been broadened with Java to mean translating source code using the javac compiler into a platform independent bytecode, as was done with the introduction of Pascal in the 1970s, whose bytecode could run on any microprocessor. Pascal's conversion into bytecode was correctly called an interpreter rather than a compiler since the bytecode was not specific machine language instructions like assembly language. However interpreters are slower than compilers and so it makes better advertising to call something a compiler rather than an interpreter.

An online demo of a finite state machine from Xerox is at (http://www.xrce.xerox.com/competencies/content-analysis/fsCompiler/fsinput.html). Links to examples of FSM such as the soft-drink machine is at (http://www.xrce.xerox.com/competencies/content-analysis/fsCompiler/fsexamples.html). A very comprehensive reference to FSM is (http://odur.let.rug.nl/alfa/fsa_stuff/).

Although FSMs are good with a restricted set of symbols, e.g., the numerals 0-9, or the letters of the alphabet, a problem occurs in areas such as speech recognition where ambiguities may occur. For example, in the two sentences:

```
(1) No one has let us read
  or
(2) No one has lettuce red
```

In sentence (1), someone is complaining they have not been able to read. In (2), no one has red lettuce. A good way to disambiguate the words "lettuce," "let us," "read," and "red" uses a **hidden Markov Machine** (HMM) that assigns probabilities to actions in a finite state machine. By looking at the whole sentence structure or additional sentences, the HMM figures out the correct context (either reading a book or searching for a vegetable on a grocery list). While expert systems are not the software of choice for doing HMM, they can be used as a front-end for speech recognition that an expert system like CLIPS can use to trigger an appropriate rule.

In fact, CLIPS has the capability to easily interface C or C++ code so that an HMM written in C or C++ can be called just like any other CLIPS keyword. One of the strengths of CLIPS is that it is an extensible language where the user can easily add keywords at compile time for greatest efficiency. Also with the object-oriented features of COOL, objects can be used to extend CLIPS using all the power of multiple inheritance. Other software such as ANS, genetic algorithms, and other software written in C or C++ can be added either on the left-hand-side to trigger rules or on the right-hand-side of rules for output. For example, a speech synthesizer, robotic effectors, and actuator can be called from CLIPS by defining the appropriate keyword functions. The availability of the source code for CLIPS means that the new keyword calls can be compiled in CLIPS with no loss of speed compared to other expert tools in which the source code is not provided.

2.4 SEMANTIC NETS

A **semantic network**, or net, is a classic AI representation technique used for propositional information. (http://www.pcai.com/web/T6110H2/R6.o1.h8/pcai.htm) A semantic net is sometimes called a **propositional net**. As discussed before, a proposition is a statement that is either true or false, such as "all dogs are mammals" and "a triangle has three sides." Propositions are a form of declarative knowledge because they state facts. In mathematical terms, a semantic net is a labeled, directed graph. A proposition is always true or false and is called **atomic** because its truth value cannot be further divided. Here the term *atomic* is used in the classical Greek sense of an indivisible object. In contrast, the fuzzy propositions discussed in Chapter 5 need not be exactly true or false.

Semantic nets were first developed for AI as a way of representing human memory and language understanding by Quillian in 1968. He used semantic nets to analyze the meanings of words in sentences. Note that the meaning of a sentence is not the same as parsing it into its tokens and lexical structure as the FSM and HMM mentioned in the previous section can do. Since then semantic nets have been applied to many problems involving knowledge representation. Such understanding of meaning is necessary to move beyond simple expert systems or AI software in order to avoid ambiguity. In the next chapter on inference, we will see how expert systems draw conclusions from facts that may be used by other rules in a chain of inference until a valid conclusion is reached. Without semantics, expert systems might fail in the same way that a human who is confused by ambiguity may also fail.

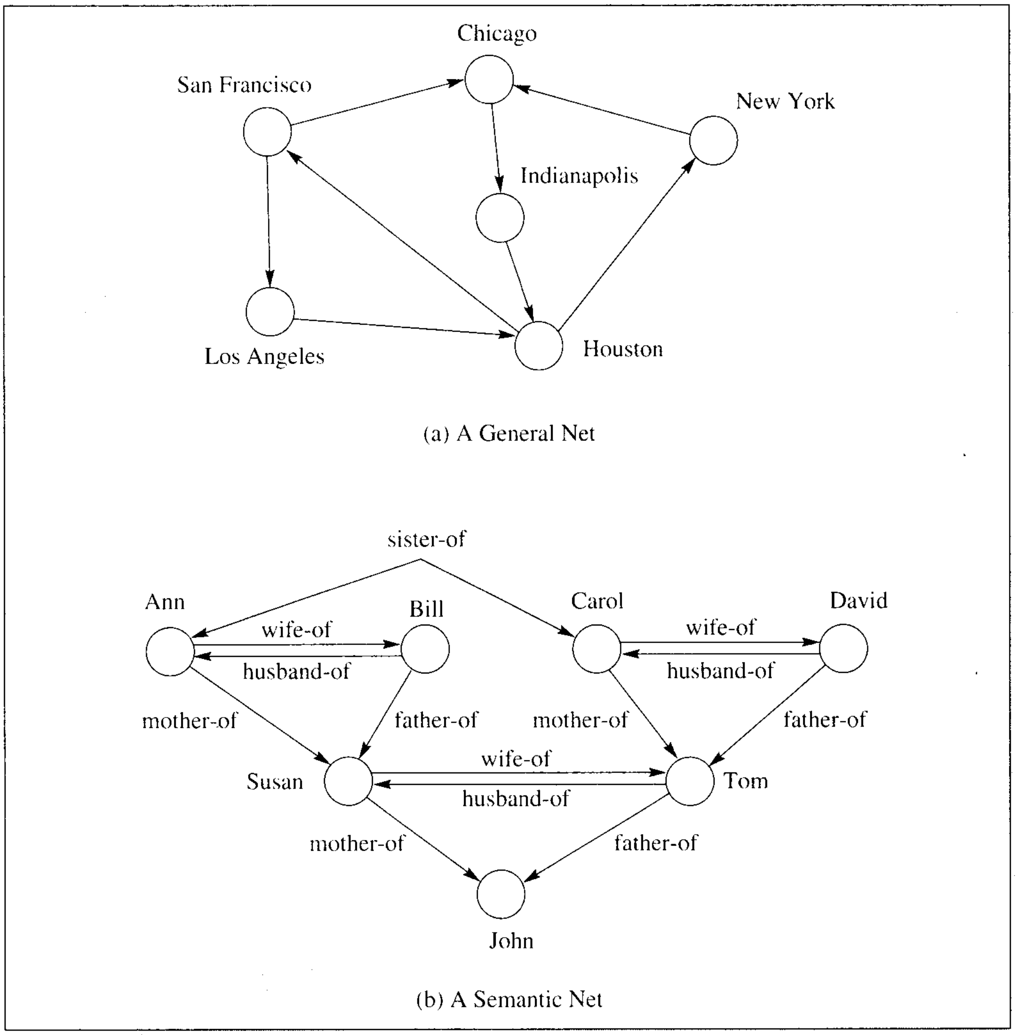
The structure of a semantic net is shown graphically in terms of **nodes** and the **arcs** connecting them. Nodes are sometimes referred to as **objects** and the arcs as **links** or **edges**.

The links of a semantic net are used to express relationships. Nodes are generally used to represent physical objects, concepts, or situations. Figure 2.4a shows an ordinary net, actually a directed graph, in which the links indicate airline routes between cities. Nodes are circles and links are the lines connecting nodes. The arrows show the directions in which planes can fly, hence the term *directed graph*. In Figure 2.4b the links show the relationship between members of a family. Relationships are of primary importance in semantic nets because they provide the basic structure for organizing knowledge. Without relationships, knowledge is simply a collection of unrelated facts. With relationships, knowledge is a cohesive structure about which other knowledge can be inferred. For example, in Figure 2.4b, it can be inferred that Ann and Bill are the grandparents of John even though there is no explicit link labeled "grandfather-of."

Semantic nets are sometimes referred to as **associative nets** because nodes are associated or related to others. In fact, Quillian's original work modeled human memory as an associative net in which concepts were the nodes and links formed the connections between concepts. According to this model, as one concept node is stimulated by reading words in a sentence, its links to other nodes are activated in a spreading pattern. If another node receives sufficient activation, the concept is elevated to the conscious mind. For example, although you know thousands of words, you are thinking of only the specific words in this sentence as you read it.

Certain types of relationships have proven particularly useful in a wide variety of knowledge representations. Rather than defining new relationships for different problems, it is customary to use these types. The use of common types makes it easier for different people to understand an unfamiliar net.

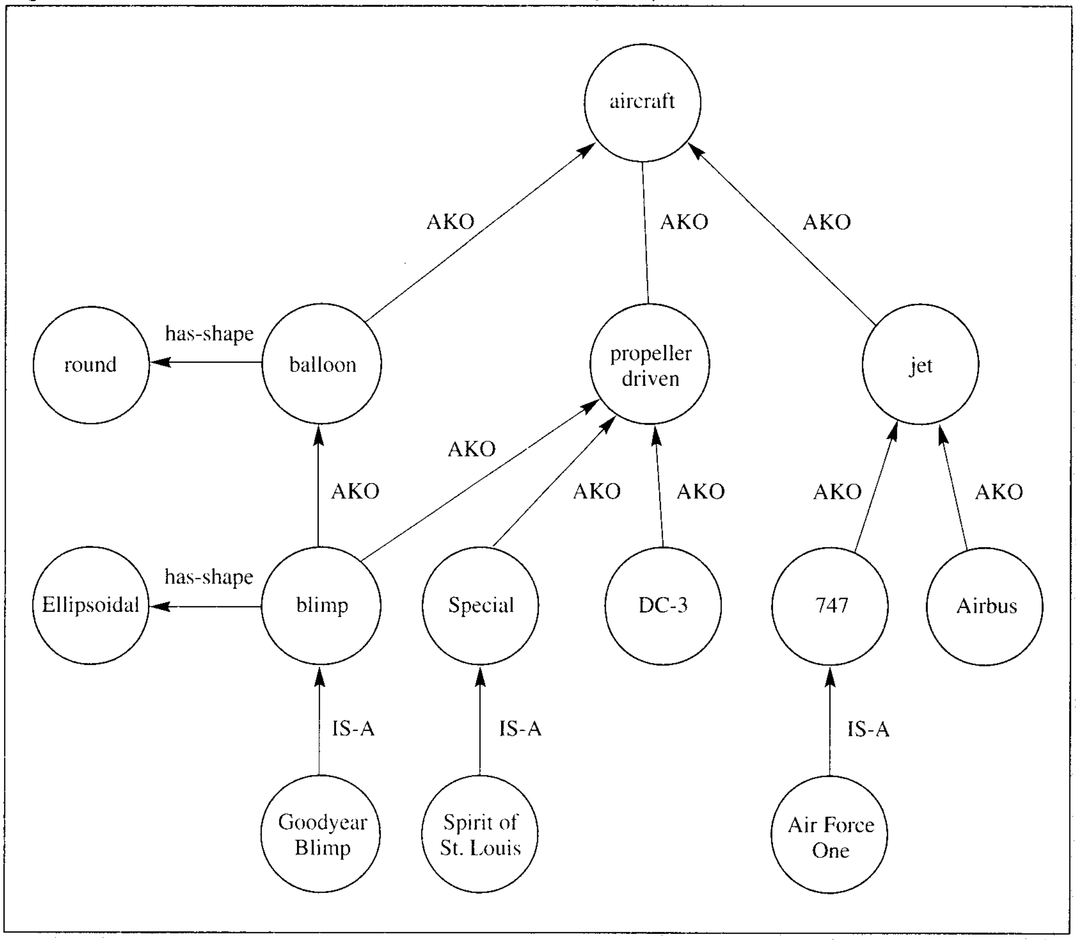
Figure 2.4 Two Types of Nets



Two types of commonly used links are **IS-A** and **A-KIND-OF**, which are sometimes written as **IS-A** and **AKO**. Figure 2.5 illustrates a semantic net using these links. In this figure the IS-A means "is an instance of" and refers to a specific member of a class. A **class** is related to the mathematical concept of a set in that it refers to a group of objects. While a set can have elements of any type, the objects in a class have some relation to one another. For example, it is possible to define a set consisting of the following elements:

```
{ 3, eggs, blue, tires, art }
```

Figure 2.5 A Semantic Net with IS-A and A-Kind-Of (AKO) Links



However, members of this set have no common relationship. In contrast, the class consisting of planes, trains, and automobiles are related because they are all types of transportation.

The link AKO is used here to relate one class to another. The AKO is not used to relate a specific individual because that is the function of IS-A. The AKO relates an individual class to a parent class of classes of which the individual is a child class.

From another viewpoint, the AKO relates **generic** nodes to generic nodes while the IS-A relates an instance or **individual** to a generic class. In Figure 2.5 notice that the more general classes are at the top and the more specific classes at the bottom. The more general class that an AKO arrow points to is called a **superclass**. If a superclass has an AKO pointing to another node, then it is also a class of the superclass the AKO points to. Another way of expressing this is that an AKO points from a **subclass** to a class. The link **ARE** is sometimes used for AKO where ARE is read as the ordinary word "are."

The objects in a class have one or more **attributes** in common. Each attribute has a **value**. The combination of attribute and value is a **property**. For example, a blimp has attributes of size, weight, shape, and color. The value of the shape attribute is ellipsoidal. In other words, a blimp has the property of an ellipsoidal shape. Other types of links may also be found in semantic nets. The **IS-A** link defines a value. For example, whatever aircraft the President is on is Air Force One. If the President is on a helicopter, then Air Force One IS -A helicopter. The **CAUSE** link expresses causal knowledge. For example, hot air CAUSES a balloon to rise.

Since the Goodyear Blimp is a blimp and blimps have ellipsoidal shape, it follows that the Goodyear Blimp is ellipsoidal. The duplication of one node's characteristics by a descendent is called **inheritance**. Unless there is more specific evidence to the contrary, it is assumed that all members of a class will inherit all the properties of their superclasses. For example, balloons have a round shape. However, since the blimp class has a link pointing to the ellipsoidal shape, this takes precedence. Inheritance is a useful tool in knowledge representation because it eliminates the need to repeat common characteristics. Links and inheritance provide an efficient means of knowledge representation since many complex relationships can be shown with a few nodes and links. Later on we will discuss the object-oriented capabilities of CLIPS using its embedded language called COOL. You can write expert systems in rules, objects, or a combination of rules and objects with true multiple inheritance.

2.5 OBJECT-ATTRIBUTE-VALUE TRIPLES

One problem with using semantic nets is that there is no standard definition of link names. For example, some books use IS-A for both generic and individual relations. The IS-A is then used in the sense of the ordinary words "is a" and is also used for AKO.

Another common link is HAS-A, which relates a class to a subclass. The HAS-A points opposite to the AKO and is often used to relate an object to a part of the object. For example,

```
car HAS-A engine
car HAS-A tires
car IS-A ford
```

More specifically, the IS-A relates a value to an attribute while a HAS-A relates an object to an attribute.

The three items of object, attribute, and value occur so frequently that it is possible to build a simplified semantic net using just them. An **object-attribute-value triple (OAV)**, or **triplet**, can be used to characterize all the knowledge in a semantic net and was used in the expert system MYCIN for diagnosing infectious diseases. The OAV triple representation is convenient for listing knowledge in the form of a table and thus translating the table into computer code by rule induction. An example of an OAV triple table is shown in Table 2.1. Induction is a powerful tool of logic that is often misused. The classic example is asking someone why they haven't backed up their hard drive. The usual faulty inductive answer is that their hard drive has never crashed and so by induction it never will. (Stockbrokers love this kind of "reasoning.") For more details on inductive logic programming, see (Bergadano 96).

OAV triples are especially useful for representing facts and the patterns to match the facts in the antecedent of a rule. The semantic net for such a system consists of nodes for objects, attributes, and values connected by HAS-A and IS-A links. If only a single object is to be represented and inheritance is not required, then an even simpler representation called **attribute-value (AV) pairs** may suffice.

Table 2.1 An OAV Table

Object	Attribute	Value
apple	color	red
apple	type	mcintosh
apple	quantity	100
grapes	color	red
grapes	type	seedless
grapes	quantity	500

2.6 PROLOG AND SEMANTIC NETS

Semantic nets are easy to translate into PROLOG. For example, the following PROLOG statements express some of the relationships in the semantic net of Figure 2.5:

```
is_a(goodyear_blimp,blimp).
is_a(spirit_of_st_louis,special).
has_shape(blimp,ellipsoidal).
has_shape(balloon,round).
```

Notice that a period marks the end of a PROLOG statement.

Essentials of PROLOG

Each of the statements above is a PROLOG **predicate expression**, or simply a predicate, because it is based on predicate logic. However, PROLOG is not a true predicate logic language because it is a computer language with executable statements. In PROLOG a predicate expression consists of the predicate name, such as is_a, followed by zero or more arguments enclosed in parentheses and separated by commas. The following are some examples of PROLOG predicates. Comments are preceded by semicolons and ignored by the PROLOG engine:

```
color(red).                 ; red is a color is a
                              fact
mother(pat,ann).            ; pat is the mother of
                              ann
parents(jim,ann,tom)        ; jim and ann are
                              parents of tom
surrogatemother(pat,tom). ; pat is surrogatemother
                              of tom
```

Predicates with two arguments are more easily understood if you consider the predicate name following the first argument. The meanings of predicates with more than two arguments must be explicitly stated, as the parents predicate illustrates. Another difficulty is that semantic nets are primarily useful for representing binary relationships since a drawn line has only two ends. It's not possible to draw the parents predicate as a single directed edge since there are three arguments. The idea of drawing Tom and Susan together in one parent's node and John at the other leads to a new complication. It is then impossible to use the parent's node for other binary relationships, such as mother-of, since it would also involve Tom.

Predicates can also be expressed with relations such as the IS-A and HAS-A:

```
is_a(red,color).
has_a(john,father).
has_a(john,mother).
has_a(john,parents).
```

Notice that the has_a predicates do not express the same meaning as previously because John's father, mother, and parents are not explicitly named. In order to name them, some additional predicates must be added:

```
is_a(tom,father).
is_a(susan,mother).
is_a(tom,parent).
is_a(susan,parent).
```

Even these additional predicates do not express the same meaning as the original predicates. For example, we know that John has a father and that Tom is a father, but this does not provide the information that Tom is the father of John.

All the preceding statements actually describe facts in PROLOG. Programs in PROLOG consist of facts and rules in the general form of **goals**:

```
p:- p1,p2,...pN.
```

where p is called the rule's head and the p_k are the **subgoals**. Normally this expression is a Horn clause which states that the head goal, p, is satisfied if and only if all of the subgoals are satisfied. The exception is when the special predicate for failure is used. A failure predicate is convenient because it's not easy to prove negation in classical logic, and PROLOG is based on classical logic. Negation is viewed as the failure to find a proof and this can be a very long and involved process if there are many potential matches. The **cut** and special failure predicate makes the negation process more efficient by reducing the searching for a proof.

The commas separating the subgoals represent a logical AND. The symbol, **:-** , is interpreted as an IF. If only the head exists and there is no right-hand side, as in:

```
p.
```

then the head is considered true. This is why predicates such as the following are considered facts and thus must be true:

```
color(red).
has_a(john,father).
```

Another way of thinking about facts is that a fact is an unconditional conclusion that does not depend on anything else and so the IF,:- , is not necessary. In contrast, PROLOG rules require the IF because they are conditional conclusions whose truth depends on one or more conditions. As an example, the parent rules are as follows:

```
parent(X,Y):- father(X,Y).
parent(X,Y):- mother(X,Y).
```

which means that X is the parent of Y if X is the father of Y or if X is the mother of Y. Likewise, a grandparent can be defined as:

```
grandparent(X,Y):- parent(X,Z),parent(Z,Y).
```

and an ancestor can be defined as:

```
(1) ancestor(X,Y):- parent(X,Y).
(2) ancestor(X,Y):- ancestor(X,Z),ancestor(Z,Y).
```

where (1) and (2) are used for identification purposes in this book.

Searching in PROLOG

A system for executing PROLOG statements is generally an interpreter, although some systems can generate compiled code. The general form of a PROLOG system is shown in Figure 2.6. The user interacts with PROLOG by entering predicate queries and receiving answers. The **predicate database** contains the rule and fact predicates that have been entered and thus forms the knowledge base. The interpreter tries to determine whether a query predicate that the user enters is in the database. The answer returned is yes if it is in the database and no if it is not. If the

Figure 2.6 General Organization of a PROLOG System

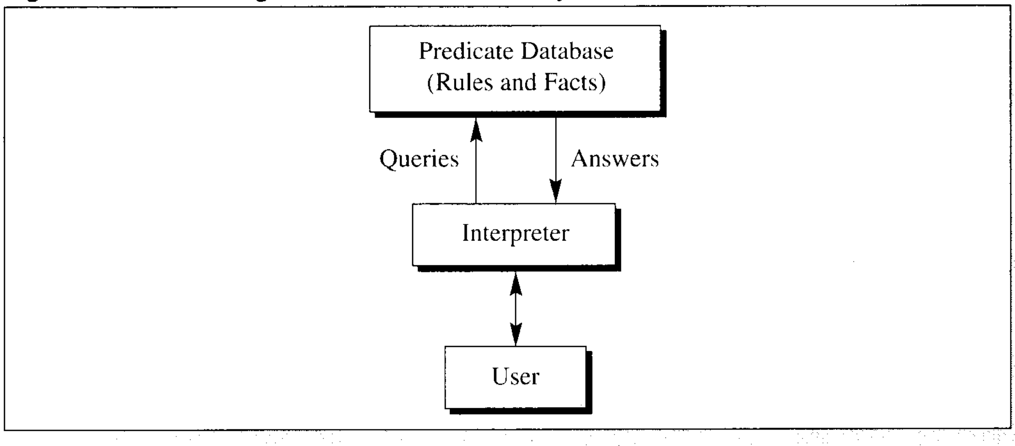
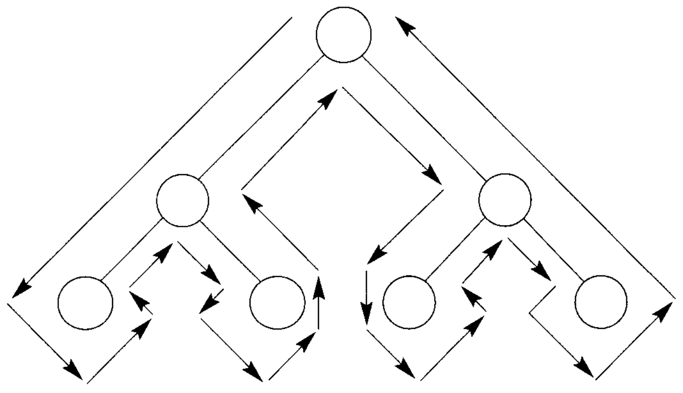
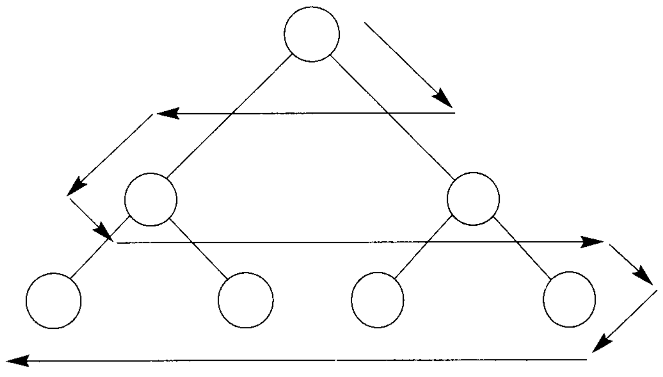


Figure 2.7 Depth-First and Breadth-First Searches for an Arbitrary Tree


(a) Depth-first Search


(b) Breadth-first Search

query is a rule, the interpreter will try to satisfy the subgoals by conducting a **depth-first search**, as shown in Figure 2.7. For contrast, a **breadth-first search** is also shown, although this is not the normal mode of PROLOG.

In a depth-first search, the search goes down as far as possible from the root and then back up again. In PROLOG the search also goes from left to right. A breadth-first search proceeds one level at a time before descending to the next lower level.

As an example of goal searching in PROLOG, consider the ancestor rules defined as (1) and (2). Assume the following facts are now input:

```
(3) parent(ann,mary).
(4) parent(ann,susan).
(5) parent(mary,bob).
(6) parent(susan,john).
```

Now suppose PROLOG is queried to determine if Ann is the ancestor of Susan:

```
:- ancestor(ann,susan).
```

where the absence of a head indicates a **query**, which is a condition to be proved by PROLOG. Facts, rules, and queries are the three types of Horn clauses of PROLOG. A condition can be proved if it is the conclusion of an instance of a clause. Of course, the clause itself must be provable, and this is done by proving the conditions of the clause.

On receipt of this input PROLOG will start searching for a statement whose head matches the input pattern of `ancestor(ann,susan)`. This is called **pattern matching**, just like the pattern matching to facts in the antecedent of a production rule. The search starts from the first statement that was entered (1), the **top**, and goes to the last statement (6), the **bottom**. There is a possible match with the first ancestor rule (1). The variable `X` matches `ann` and the variable `Y` matches `susan`. Since the head matches, PROLOG will now try to match the body of (1), yielding the subgoal `parent(ann, susan)`. PROLOG now tries to match this subgoal against clauses and eventually matches with the fact (4) `parent(ann,susan)`. There are no more goals to match and PROLOG answers "yes" because the original query is true.

As another example, suppose the query is the following:

```
:- ancestor(ann,john).
```

The first ancestor rule (1) matches and `X` is set to `ann` while `Y` is set to `john`. PROLOG now tries to match the body of (1), `parent(ann,john)` with every parent statement. None match and so the body of (1) cannot be true. Since the body of (1) is not true, the head cannot be true.

Since (1) cannot be true, PROLOG then tries the second ancestor statement (2). `X` is set to `ann` and `Y` is set to `john`. PROLOG tries to prove that the head of (2) is true by proving that the subgoals of (2) are both true. That is, ancestor(ann,Z) and ancestor(Z,john) must be proven true. PROLOG tries to match the subgoals in a left-to-right manner starting with ancestor(ann,Z). Starting from the top, this first match's statement (1) and so PROLOG tries to prove the body of (1), parent(ann,Z). Starting to search from the top again, this body first matches statement (3) and so PROLOG sets Z to mary. Now PROLOG tries to match the second subgoal of (2) as ancestor(mary,john). This matches (1) and so PROLOG tries to satisfy its body, parent(mary,john). However, there is no parent(mary,john) in the statements (3) through (6) and so this search fails.

PROLOG then reconsiders its guess that Z should be mary. Since this choice did not work, it tries to find another value that will work. Another possibility arises from (4) by setting Z to susan. This technique of going back to try a different search path when one path fails is called **backtracking**, and is often used for problem solving.

With the choice of Z as susan, PROLOG attempts to prove that ancestor(susan,john) is true. By (1), the body parent(susan,john) must be proven true. Indeed, the fact (3) does match and so the query:

```
:- ancestor(ann,john)
```

is proven true since its conditions have been proven true.

Notice that the control structure of PROLOG is of the Markov algorithm type, where searching for pattern matching is normally determined by the order that the Horn clauses are entered. This contrasts with rule-based expert systems, which normally follow the Post paradigm, in which the order that rules are entered does not affect the search.

PROLOG has many other features and capabilities not mentioned here. From an expert systems point of view, the backtracking and pattern matching are especially relevant. The declarative nature of PROLOG is also useful because a program specification is an executable program.

2.7 DIFFICULTIES WITH SEMANTIC NETS

Although semantic nets can be very useful in representing knowledge, they have limitations, such as the lack of link name standards discussed previously. This makes it difficult to understand what the net is really designed for and whether it was designed in a consistent manner. A complementary problem to naming links is the naming of nodes. If a node is labeled "chair," for example, does it represent:

a specific chair
the class of all chairs
the concept of a chair
the person who is the chair of a meeting

or something else? For a semantic net to represent **definitive knowledge**, that is, knowledge that can be defined, the link and node names must be rigorously defined. Of course, the same problems may occur in programming languages. Fortunately this problem has now been solved with the introduction of **extensible markup language (XML)** and ontologies. XML has proven invaluable in providing a standard way of encoding formal semantics into all kinds of languages. A rule-based version of XML is also available as are markup languages for math, music, the food industry and all other fields that use computers for information processing. XML and ontologies are turning the World Wide Web (WWW or W3) from just data into a collection of machine readable information of much more use. The web is now being referred to as the **Semantic Web** rather than just the web because of its new sense of meaning.

Another problem is the combinatorial explosion of searching nodes, especially if the response to a query is negative. That is, for a query to produce a negative result, many or all of the links in a net may have to be searched. As shown in the Traveling Salesman problem in Chapter 1, the number of links is the factorial of the number of nodes minus one if they are all connected. Although not all representations will require this degree of connectivity, the possibility of a combinatorial explosion exists.

Semantic nets were originally proposed as a model of human associative memory in which one node has links to others and information retrieval occurs due to a spreading activation of nodes. However, other mechanisms must also be available to the human brain since it does not take a long time for a human to answer the query—Is there a football team on Pluto? There are about 10^{11} neurons in the human brain and about 10^{15} links. If all knowledge were represented by a semantic net, it would take a very, very long time to answer negative queries like the football question because of all the searching involved with 10^{15} links.

Semantic nets are logically inadequate because they cannot define knowledge in the way that logic can. A logic representation can specify a certain chair, some chairs, all chairs, no chairs, and so forth, as will be discussed later in this chapter. Another problem is that semantic nets are heuristically inadequate because there is no way to embed heuristic information in the net on how to efficiently search the net. A **heuristic** is a rule of thumb that may help in finding a solution but is not guaranteed, in the way an algorithm is guaranteed, to find a solution. Heuristics are very important in AI because typical AI problems are so hard that an algorithmic solution does not exist or is too inefficient for practical use. The only standard control strategy built into a net that might help is inheritance but not all problems may have this structure.

A number of approaches have been tried to correct the inherent problems of semantic nets. Logic enhancements have been made and heuristic enhancements have been tried by attaching procedures to nodes. The procedures will be executed when the node becomes activated. However, the resulting systems gained little in capability at the expense of the natural expressiveness of semantic nets. The conclusion of all this effort is that like any tool, semantic nets should be used for those things they do best, showing binary relationships, and not be distorted into a universal tool.

2.8 SCHEMATA

A semantic net is an example of a **shallow knowledge structure**. The shallowness occurs because all the knowledge in the semantic net is contained in the links and nodes. **Knowledge structure** is analogous to a data structure in that it represents an ordered collection of knowledge rather than just data. A **deep knowledge structure** has causal knowledge that explains why something occurs. For example, it is possible to build a medical expert system with shallow knowledge as follows:

```
IF a person has a fever
THEN take an aspirin
```

But these systems do not know the fundamental biochemistry of the body and why aspirin decreases fever. The rule could have been defined:

```
IF a person has a pink monkey
THEN take a refrigerator
```

In other words, the expert system's knowledge is shallow because it is based on *syntax* rather than *semantics*, where any two words could be substituted for X and Y in the following rule:

```
IF a person has a (X)
THEN take a (Y)
```

Note that (X) and (Y) are not variables in this rule, but represent any two words. Doctors have causal knowledge because they have many years of medical school and have experience treating ill people. If a treatment is not working right, doctors can use reasoning powers to find an alternative to their usual case-based method. In other words, an expert knows when to break the rules.

Many types of real-world knowledge cannot be represented by the simple structure of a semantic net. More complex structures are needed to better represent complex knowledge structures. In AI, the term **schema** (plural schemas or schemata) is used to describe a more complex knowledge structure than the semantic net. The term schema comes from psychology, where it refers to the continual organizing of knowledge or responses by a creature in response to stimuli. That is, as creatures learn the causal relationship between a cause and its outcome, they will try to repeat the cause if pleasurable and avoid the cause if painful.

For example, the acts of eating and drinking are pleasurable **sensorimotor schemata** that involve coordinating information from the senses with the required motor (muscle) movements to eat and drink. A person does not have to think about this tacit knowledge to know to perform these acts, and it is very difficult to explain exactly how it is done down to the level of controlling muscles. An even more difficult schema to explain is how to ride a bicycle. Try explaining a sense of balance!

Another type of schema is the **concept schema** by which we represent concepts. For example, everyone has the concept of animal. If most people were asked to explain what an animal is, they would probably describe it in terms of something that has four legs and fur. Of course, the concept of animal differs depending on whether the person grew up on a farm, in a city, or near a river. However, we all have **stereotypes** in our minds of concepts. While the term stereotype may have a derogatory meaning in casual language, in AI it means a typical example. Thus a stereotype of an animal might be a dog to many people.

A conceptual schema is an abstraction in which specific objects are classified by their general properties. For example, if you see a small red, round object with a green stem under a sign that says Artificial Fruit, you will probably recognize it as an artificial apple. The object has the properties of appleness that you associate with the conceptual schema of apple.

The conceptual schema of a real apple will include general properties of apples such as sizes, colors, tastes, uses, and so forth. The schema will not include details of exactly where the apple was picked, the truck by which it was transported to the supermarket, or the name of the person who put it on the shelf. These details are not important to the properties that comprise your abstract concept of an apple. Also, note that a person who is blind may have a very different concept schema of an apple in which texture is most important.

By focusing on the general properties of an object, it is easier to reason about them without becoming distracted by irrelevant details. Customarily, schemas have internal structure to their nodes while semantic nets do not. The label of a semantic net is all the knowledge about the node. A semantic net is like a data structure in computer science, in which the search key is also the data stored in the node. A schema is like a data structure in which nodes contain records. Each record may contain data, records, or pointers to other nodes.

2.9 FRAMES

One type of schema that has been used in many AI applications is the **frame**. Another type of schema is the **script**, which is essentially a time-ordered sequence of frames. Proposed as a method for understanding vision, natural language, and other areas, frames provide a convenient structure for representing objects that are typical to a given situation such as stereotypes. In particular, frames are useful for simulating commonsense knowledge, which is a very difficult area for computers to master. While semantic nets are basically a two-dimensional representation of knowledge, frames add a third dimension by allowing nodes to have structures. These structures can be simple values or other frames.

The basic characteristic of a frame is that it represents related knowledge about a narrow subject that has much default knowledge. A frame system would be a good choice for describing a mechanical device, for example a car. Components of the car such as the engine, body, brakes, and so forth would be related to give an overall view of their relationships. Further detail about the components could be obtained by examining the structure of the frames. Although individual brands of cars will vary, most cars have common characteristics such as wheels, engine, body, and transmission. The frame contrasts with the semantic net, which is generally used for broad knowledge representation. Just as with semantic nets, there are no standards for defining frame-based systems. A number of special-purpose languages have been designed for frames, such as FRL, SRL, KRL, KEE, HP-RL, and frame enhancement features to LISP such as LOOPS and FLAVORS.

A frame is analogous to a record structure in a high-level language such as C or an atom with its property list in LISP. Corresponding to the fields and values of a record are the **slots** and slot **fillers** of a frame. A frame is basically a group of slots and fillers that defines a stereotypical object. A frame for a car is shown in Figure 2.8. In OAV terms, the car is the object, the slot name is the attribute, and the filler is the value.

Figure 2.8 A Car Frame

Slots	Fillers
manufacturer	General Motors
model	Chevrolet Caprice
year	1979
transmission	automatic
engine	gasoline
tires	4
color	blue

Most frames are not as simple as of the one shown in Figure 2.8. The utility of frames lies in hierarchical frame systems and inheritance. By using frames in the filler slots and inheritance, very powerful knowledge representation systems can be built. In particular, frame-based expert systems are very useful for representing causal knowledge because their information is organized by cause and effect. By contrast, rule-based expert systems generally rely on unorganized knowledge that is not causal.

Some frame-based tools such as KEE allow a wide range of items to be stored in slots. Frame slots may hold rules, graphics, comments, debugging information, questions for users, hypotheses concerning a situation, or other frames.

Frames are generally designed to represent either generic or specific knowledge. Figure 2.9 illustrates a generic frame for the concept of human property.

The fillers may be values, such as property in the name slot, or a range of values, as in the types slot. The slots may also contain procedures attached to the slots, called **procedural attachments**. These are generally of three types. The **if-needed** type is executed when a filler value is needed but none are initially present or the **default** value is not suitable. Defaults are of primary importance in frames because they model some aspects of the brain. Defaults

Figure 2.9 A Generic Frame for Property

Slots	Fillers
name	property
specialization_of	a_kind_of object
types	(car, boat, house) if-added: Procedure ADD_PROPERTY
owner	default: government if-needed: Procedure FIND_OWNER
location	(home, work, mobile)
status	(missing, poor, good)
under_warranty	(yes, no)

correspond to the expectations of a situation that we build up based on experience. When we encounter a new situation, the closest frame is modified to help us adjust to the situation. People do not start from scratch in every new situation. Instead, the defaults or other fillers are modified. Defaults are often used to represent commonsense knowledge. **Commonsense knowledge** can be considered to be the knowledge that is generally known. We use commonsense when no more situation-specific knowledge is available.

The **if-added** type is run for procedures to be executed when a value is to be added to a slot. In the types slot, the if-added procedure executes a procedure called ADD-PROPERTY to add a new type of property, if necessary. For example, this procedure would be run for jewelry, TV, stereo, and so forth since the types slot does not contain these values.

An **if-removal** type is run whenever a value is to be removed from a slot. This type of procedure would be run if a value were obsolete.

Slot fillers may also contain relations, as in the specialization of slots. The a-kind-of and is-a relations are used in Figures 2.9, 2.10, and 2.11 to show how these frames are hierarchically related. The frames of Figures 2.9 and 2.10 are generic frames while that of Figure 2.11 is a specific frame because it is an instance of the car frame. We have adopted the convention here that a-kind-of relations are generic and is-a relations are specific.

Frame systems are designed so that more generic frames are at the top of the hierarchy. It is assumed that frames can be customized for specific cases by modifying the default cases and creating more specific frames. Frames attempt to model real-world objects by using generic knowledge for the majority of an object's attributes and specific knowledge for special cases. For example, we commonly think of birds as creatures that can fly. Yet certain types of birds such as penguins and ostriches cannot fly. These types represent more specific classes

Figure 2.10 Car Frame—A Generic Subframe of Property

Slots	Fillers
name	car
specialization_of	a-kind-of property
types	(sedan, sports, convertible)
manufacturer	(GM, Ford, Toyota)
location	mobile
wheels	4
transmission	(manual, automatic)
engine	(gasoline, hybrid gas/electric)

Figure 2.11 An Instance of a Car Frame

Slots	Fillers
name	John's car
specialization_of	is_a car
manufacturer	GM
owner	John Doe
transmission	automatic
engine	gasoline
status	good
under_warranty	yes

of birds and their characteristics would be found lower in a frame hierarchy than birds like canaries or robins. In other words, the top of the bird frame hierarchy specifies things that are more true of all birds while the lower levels reflect the fuzzy boundaries between real-world objects. The object that has all of the typical characteristics is called a **prototype**, which literally means the *first type*.

Frames may also be classified by their applications. A **situational frame** contains knowledge about what to expect in a given situation, such as a birthday party. An **action frame** contains slots that specify the actions to be performed in a given situation. That is, the fillers are procedures to perform some actions, such as removing a defective part from a conveyer belt. An action frame represents procedural knowledge. The combination of situational and action frames can be used to describe cause-and-effect relationships in the form of **causal knowledge frames**.

Very sophisticated frame systems have been built for a variety of tasks. One of the most impressive systems that demonstrated the power of frames to creatively discover mathematical concepts was the Automated Mathematician (AM) of Doug Lenat. The classic AM system of Lenat made conjectures of interesting new concepts and explored them. It came up with some completely new mathematical proofs of certain theorems. Many other examples of theorem provers and discovery systems are shown in Appendix G for this chapter.

2.10 DIFFICULTIES WITH FRAMES

Frames were originally conceived as a paradigm for representing stereotyped knowledge. The important characteristic of a stereotype is that it has well-defined features so that many of its slots have default values. Mathematical concepts are good examples of stereotypes well suited to frames. The frame paradigm has an intuitive appeal because its organized representation of knowledge is generally easier to understand than either logic or production systems with many rules (Jackson 99).

However, major problems have appeared in frame systems that allow unrestrained alteration or cancellation of slots. The classic example of this problem is illustrated with a frame describing elephants, as shown in Figure 2.12.

Figure 2.12 Elephant Frame

name	elephant
specialization_of	a-kind-of mammal
color	grey
legs	4
trunk	a cylinder

At first sight, the elephant frame seems like a reasonable generic description of elephants. However, suppose there exists a specific, three-legged elephant called Clyde. Clyde may have lost a leg due to a hunting accident or simply lied about losing a leg to get his name in this book. The significant thing is that the elephant frame claims that an elephant has four legs, not three. Thus we cannot consider the elephant frame to be the valid definition of an elephant.

Of course, the frame could be modified to allow three-legged, two-legged, one-legged, or even legless elephants as exceptions. However, this does not provide a very good definition. Additional problems may arise with the other slots. Suppose Clyde gets a bad case of jaundice and his skin turns yellow. Is he then not an elephant?

An alternative to viewing a frame as a definition is to consider it as describing a typical elephant. However, this leads to other problems because of inheritance. Notice that the elephant frame says that an elephant is a-kind-of mammal. Since we are interpreting frames as typical, then our frame system says that a typical elephant is a typical mammal. Although an elephant is a mammal, it is probably not a typical mammal. On the basis of quantity, people, cows, rats, and sheep are probably more representative of typical mammals.

Most frame systems do not provide a way of defining unalterable slots. Since any slot can be changed, the properties that a frame inherits can be altered or cancelled anywhere in the hierarchy. This means that every frame is really a primitive frame because there is no guarantee that properties are common. Each frame makes up its own rules and so each frame is a primitive. Nothing is really certain in such an unrestrained system, and so it is impossible to make universal statements such as those made in defining an elephant. Also, from simpler definitions such as that for an elephant, it is impossible to reliably build composite objects such as elephant-with-three-legs. The same types of problems apply to semantic nets with inheritance. If the properties of any node can be changed, then nothing is certain.

However there is another way to view frames that is very useful. If the concept of frame is extended to include the properties of objects, then an object can be considered a frame. CLIPS has a complete object-oriented language called COOL embedded within it that can be considered as an extension of a frame-based language. All the advantages of objects are now available in CLIPS. Object-oriented expert systems can be constructed in CLIPS that have both the advantages of rules as small chunks of knowledge while providing the capability to organize larger chunks of knowledge into objects for easier maintenance and development. Rules can operate on facts or objects and thus CLIPS has the advantages of both a rule-based and object-based expert system tool. The use of objects in CLIPS makes it easier to organize, execute, and maintain large knowledge bases rather than trying to keep all the knowledge in thousands of separate rules and facts.

2.11 LOGIC AND SETS

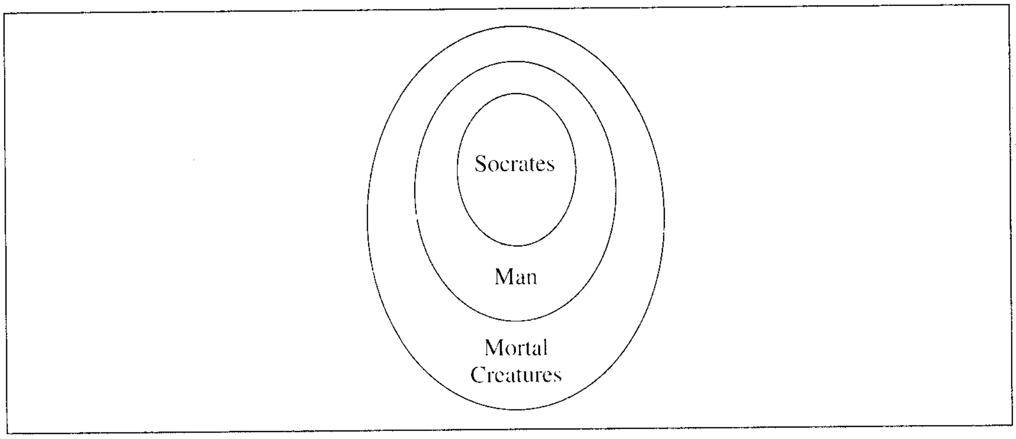
In addition to rules, frames, and semantic nets, knowledge can also be represented by the symbols of **logic**, which is the study of the rules of exact reasoning. An important part of reasoning is inferring conclusions from premises. The application of computers to perform reasoning has resulted in **logic programming** and the development of logic-based languages such as PROLOG. Logic is also of primary importance in expert systems in which the inference engine reasons from facts to conclusions. In fact, a descriptive term for logic programming and expert systems is **automated reasoning systems**.

The earliest formal logic was developed by the Greek philosopher Aristotle in the fourth century B.C. Aristotelian logic is based on the **syllogism**, of which he invented fourteen types. Five more were invented in medieval times. Syllogisms have two **premises** and one **conclusion**, which is inferred from the premises. The following is a classic example of a syllogism:

```
Premise:    All men are mortal
Premise:    Socrates is a man
Conclusion: Socrates is mortal
```

In a syllogism the **premises** provide the evidence from which the conclusion must necessarily follow. A syllogism is a way of representing knowledge. Another way is a **Venn diagram**, as shown in Figure 2.13.

Figure 2.13 Venn Diagram



The outer circle represents all mortal creatures. The inner circle representing men is drawn entirely within the mortal circle to indicate that all men are mortal. Since Socrates is a man, the circle representing him is drawn entirely within the human circle. Strictly speaking, the circle representing Socrates should be a point, since a circle implies a class. For readability, we'll use circles for all. The conclusion that Socrates is mortal is a consequence of the fact that his circle is within that of mortal creatures and so he must be mortal.

In mathematical terms, a circle of the Venn diagram represents a **set**, which is a collection of objects. The objects within a set are called its **elements**. Some examples of sets are the following:

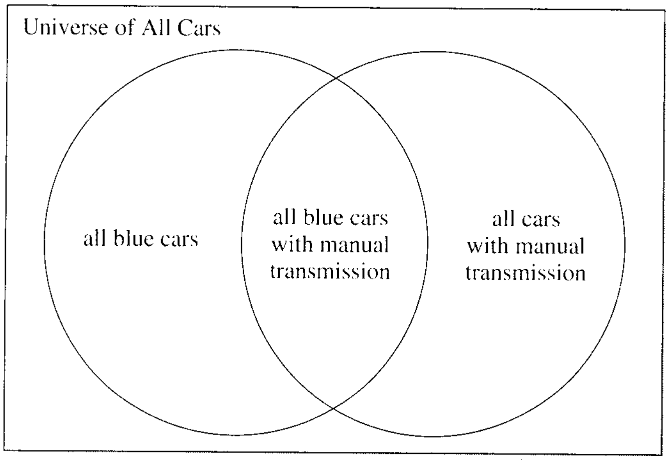
```
A = {1,3,5}
B = {1,2,3,4,5}
C = {0,2,4,...}
D = {...,-4,-2,0,2,4,...}
E = {airplanes,balloons,blimps,jets}
F = {airplanes,balloons}
```

where the three dots, ..., called **ellipses**, indicate that the terms continue on indefinitely.

The Greek symbol epsilon, $\in$, indicates an element is a member of a set. For example, $1 \in A$, means that the number 1 is an element of the set A, previously defined. If an element is not a member of a set, the symbol $\notin$ is used, as in $2 \notin A$.

If two arbitrary sets, such as X and Y, are defined such that every element in X is an element of Y, then X is a **subset** of Y and is written in the mathematical form $X \subset Y$ or $Y \supset X$. From the definition of a subset it follows that every set is a subset of itself. A subset that is not the set itself is called a **proper subset**. For example, the set X defined previously is a proper subset of Y. In discussing sets it is useful to consider the sets under discussions as subsets of a **universal set**. The universal set (universe) changes as the topic of discussion change. Figure 2.14 illustrates a subset formed as the **intersection** of two sets in the universe of all cars. In discussing numbers the universe is all numbers. The universe is drawn as a rectangle surrounding its subsets. Universal sets are not used just for convenience. The indiscriminate use of conditions to define sets can result in logical paradoxes.

Figure 2.14 Intersecting Sets


Universe of All Cars
all blue cars
all blue cars with manual transmission
all cars with manual transmission

If we let A be the set of all blue cars, B the set of all cars with manual transmissions, and C the set of all blue cars with manual transmissions, we can write:

$$C = A \cap B$$

where the symbol $\cap$ represents the intersection of sets. Another way of writing this is in terms of elements x as follows:

$$C = \{x \in U \mid (x \in A) \wedge (x \in B)\}$$

where:

U represents the universe set.

| is read as "such that." A colon, :, is sometimes used instead of |.

$\wedge$ is the logical AND.

The expression for C is read that C consists of those elements x in the universe such that x is an element of A and x is an element of B. The logical AND comes from Boolean algebra. An expression consisting of two operands connected by the logical AND operator is true if and only if both operands are true. If A and B have no elements in common, then $A \cap B = \emptyset$ where the Greek letter phi, $\emptyset$, represents the **empty set** or **null set** { }, which contains no elements. Sometimes the Greek letter lambda, Λ, is used for the null set. Other notations for the universe set are sometimes the numeral 1 and the numeral 0 for the null set. Although the null set has no elements, it is still a set. An analogy is a restaurant with a set of customers. If nobody comes, the set of customers is empty, but the restaurant still exists.

Another set operation is **union**, which is the set of all elements in either A or B:

$$A \cup B = \{x \in U \mid (x \in A) \vee (x \in B)\}$$

where:

$\cup$ is the set operator for **union.**
$\vee$ is the **logical OR** operator.

The **complement** of set A is the set of all elements not in A:

$$A' = \{x \in U \mid \sim(x \in A)\}$$

where:

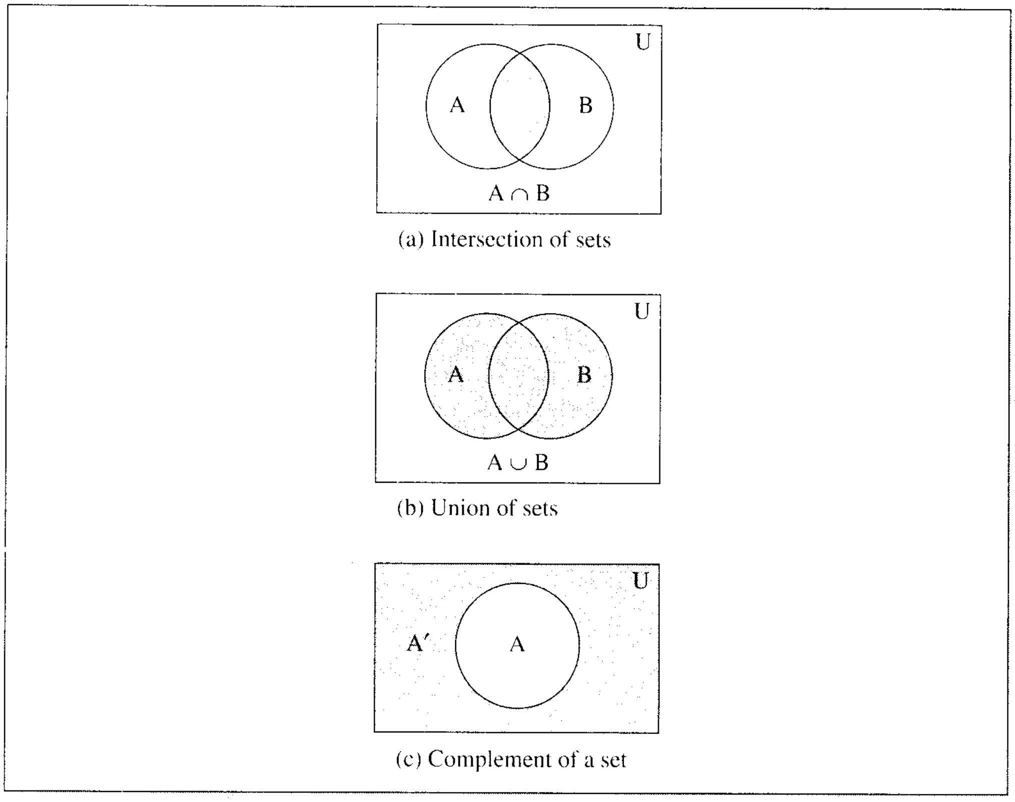
Prime, ′, means complement of a set.
Tilde, ~, is the **logical NOT** operator.

The Venn diagrams for these basic operations are shown in Figure 2.15 on next page.

2.12 PROPOSITIONAL LOGIC

The oldest and one of the simplest types of **formal logic** is the syllogism. The term *formal* means that the logic is concerned with the form of logical statements rather than their meaning. In other words, formal logic is concerned with the syntax of statements, not their semantics. This is extremely important in building expert systems because you have to separate knowledge from reasoning. What may appear as reasoning could be knowledge. For example, the

Figure 2.15 Venn Diagrams of Basic Set Operations



(a) Intersection of sets

(b) Union of sets

(c) Complement of a set

statement, "The President is always right because he is never wrong," has the appearance of reasoning due to the "because" that connects the two parts of the sentence. In fact, an assertion of this type is called a **tautology** because unlike a fact which may or may not be true in the real world, a tautology is always true in a purely logical sense because it refers to itself for proof. It states, "X is X." However if you are not of the same political party as the President, you may disagree based on the semantics rather than the form of the statement, and prefer the tautology, "The President is always wrong because he is never right."

Although the term *formal logic* may sound intimidating, it is no more difficult than algebra. In fact, algebra is really a formal logic of numbers. For example, suppose you were asked to solve the following problem: A school has 25 computers with a total of 60 memory chips. Some of the computers have two memory chips while others have four. How many computers of each type are there? The solution can be written algebraically as follows:

```
25 = X + Y
60 = 2X + 4Y
```

which can easily be solved for X = 20 and Y = 5.

Now consider this problem. There are 25 animals in a barnyard with a total of 60 legs. Some of the animals have two legs, while others have four. (Note: Clyde, our three-legged elephant is in Africa, not the barnyard.) How many animals of each type are there? As you can see, the same algebraic equations apply whether we are talking about computers, animals, or anything else. In the same way that algebraic equations let us concentrate on the mathematical manipulation of symbols without regard to what they represent, formal logic lets us concentrate on the reasoning without becoming confused by what objects we are reasoning about.

As an example of formal logic, consider the syllogism with the nonsense words *squeeg* and *moof*.

```
Premise:    All squeegs are moofs
Premise:    John is a squeeg
Conclusion: John is a moof
```

Although the words squeeg and moof are nonsense and have no meaning, the *form* of this argument is still correct. That is, the argument is **valid** no matter what words are used because the syllogism has a valid form. In fact any syllogism of the form:

```
Premise:    All X are Y
Premise:    Z is a X
Conclusion: Z is a Y
```

is valid no matter what is substituted for X, Y, and Z. This example illustrates that meanings do not matter in formal logic. Only the form or appearance is important. This concept of separating the form from the meaning or semantics is what makes logic such a powerful tool. By separating the form from the semantics, the validity of an argument can be considered objectively, without prejudice caused by the semantics. An analogy to formal logic is algebra, in which the correctness of expressions such as $X + X = 2X$ holds whether X is an integer, apples, or airplanes.

Aristotelian syllogisms were the foundations of logic until 1847, when the English mathematician George Boole published the first book describing **symbolic logic**. Although Leibnitz had developed his own version of symbolic logic in the Seventeenth Century, it never achieved general use. One of the new concepts that Boole proposed was a modification of the Aristotelian view that the subject have existence, called **existential import**. According to the classic Aristotelian view, a proposition such as "all mermaids swim well" could not be used as a premise or conclusion because mermaids don't exist. The Boolean view, now called the modern view, relaxes this restriction. The importance of the modern view is that empty classes can now be reasoned about. For example, a proposition such as "all disks that fail are cheap" could not be used in an Aristotelian syllogism unless there was at least one disk that had failed.

Another contribution by Boole was the definition of a set of **axioms** consisting of symbols to represent objects and classes, and algebraic operations to manipulate the symbols. Axioms are the fundamental definitions from which logical systems such as mathematics and logic itself are built. Using only the axioms, theorems can be constructed. A theorem is a statement that can be proved by showing how it is derived from the axioms. Between 1910 and 1913, Whitehead and Russell published their monumental 2,000-page, three-volume work *Principia Mathematica*, which showed a formal logic as the basis of mathematics. This was hailed as a great milestone because it appeared to put mathematics on a firm foundation—rather than appealing to intuition—by totally eliminating the meaning of arithmetic and instead concentrating on forms and their manipulation. Because of this, mathematics has been described as a collection of meaningless marks on paper.

In 1931 the famous mathematician Gödel proved that formal systems based on axioms could not always be proved internally consistent and free from contradictions. Gödel's proof was a great blow to mathematicians who had hoped that arithmetic could be proved true once and for all. Instead it seems we can't really count on arithmetic (note that any intelligent mathematician reading this pun will think it very funny).

Propositional logic, sometimes called **propositional calculus**, is a symbolic logic for manipulating propositions. In particular, propositional logic deals with the manipulation of **logical variables**, which represent propositions. Although most people think of calculus in terms of the calculus invented by Newton and Leibnitz, the word has a more general meaning. It comes from the Latin word *calculus* and means a little stone used for calculations. The general meaning of calculus is a special system for manipulating symbols. Other terms used for propositional logic are **statement calculus** and **sentential calculus**. Sentences are generally classified as one of four types, as illustrated in Table 2.2.

Table 2.2 Types of Sentences

Type	Example
Imperative	Do what I tell you!
Interrogative	What is that?
Exclamatory	That's great!
Declarative	A square has four equal sides.

Propositional logic is concerned with the subset of declarative sentences that can be classified as either true or false. A sentence such as "A square has four equal sides" has a definite truth value of true, while the sentence "George Washington was the second president" has a truth value of false. A sentence whose truth value can be determined is called a **statement** or **proposition**. A statement is also called a **closed sentence** because its truth value is not open to question.

If I put a preface to this book stating "everything in these pages is a lie" that statement cannot be classified as either true or false. If it is true, then I told the truth in the preface—which I cannot do. If it is untrue, and so every word written must be true, then I lied—which also cannot be true. Statements of this type are known as the **Liar Paradox**. Statements that cannot be answered absolutely are called **open sentences**. The sentence "Spinach tastes wonderful" is also an open sentence because it is true for some people and not true for others. The sentence "He is tall" is called an open sentence because it contains a variable "He." Truth values cannot be assigned to open sentences until we know the specific person or instance referred to by the variable. Another difficulty with this sentence is the meaning of the word "tall." What's tall to some people is not tall to others. While this ambiguity of "tall" cannot be handled in the propositional or predicate calculus, it can be easily dealt with in fuzzy logic, described in Chapter 5.

A **compound statement** is formed by using logical connectives on individual statements. The common logical connectives are shown in Table 2.3.

Table 2.3 Common Logical Connectives

Connective	Meaning
$\wedge$	AND; conjunction
$\vee$	OR; disjunction
~	NOT; negation
$\rightarrow$	if...then; conditional
$\leftrightarrow$	if and only if; biconditional

Strictly speaking, the negation is not a connective because it is a unary operation that applies to the one operand following, so it doesn't really connect anything. The negation has higher precedence than the other operators and so it is not necessary to put parentheses around it. That is, a statement like $\sim p \wedge q$ means the same as $(\sim p) \wedge q$.

The conditional is analogous to the arrow of production rules in that it is expressed as an IF-THEN form. For example:

```
if it is raining then carry an umbrella
```

can be put in the form:

```
p → q
```

where:

```
p = it is raining
q = carry an umbrella
```

Sometimes the $\supset$ is used for $\rightarrow$. Another term for the conditional is **material implication**.

The biconditional, $p \leftrightarrow q$, is equivalent to:

```
(p → q) ∧ (q → p)
```

and is true only when p and q have the same truth values. That is, p↔q is true only when p and q are both true or when they are both false. The biconditional has the following meanings:

```
p if and only if q
q if and only if p
if p then q, and if q then p
```

As mentioned earlier, a tautology is a compound statement that is always true, whether its individual statements are true or false. A **contradiction** is a compound statement that is always false. A **contingent** statement is one that is neither a tautology nor a contradiction. Tautologies and contradictions are called analytically true and analytically false respectively, because their truth values can be determined by their forms alone. For example, the truth table of $p \vee \sim p$ shows it is a tautology, while $p \wedge \sim p$ is a contradiction.

If a conditional is also a tautology, then it is called an **implication** and has the symbol ⇒ in place of →. A biconditional that is also a tautology is called a **logical equivalence** or **material equivalence** and is symbolized by either ⇔ or ≡. Two statements that are logically equivalent always have the same truth values. For example, $p \equiv \sim \sim p$.

Unfortunately, this is not the only possible definition for implication, since there are sixteen possible truth tables for two variables that can take on true and false values. In early expert systems of the 1980s, 11 different definitions of the implication operator were defined.

The conditional does not mean exactly the same as the IF-THEN in a procedural language or a rule-based expert system. In procedural and expert systems, the IF-THEN means to execute the actions following the THEN if the conditions of the IF are true. In logic, the conditional is defined by its truth table. Its meaning can be translated into natural language in a number of ways. For example, if:

```
p → q
```

where p and q are any statements, this can be translated as:

```
p implies q
if p then q
p, only if q
p is sufficient for q
q if p
q is necessary for p
```

For example, let p represent "you are 18 or older" and q represents "you can vote." The conditional p → q can mean any of the following:

```
you are 18 or older implies you can vote
if you are 18 or older then you can vote
you are 18 or older, only if you can vote
you are 18 or older is sufficient for you can vote
you can vote if you are 18 or older
you can vote is necessary for you are 18 or older
```

In some cases, a change of wording is necessary to make these grammatically correct English sentences. The last example says if q does not occur, then neither will p. It is expressed in proper English as "Being able to vote requires you to be 18 or older."

Values for the binary logical connectives are shown in Table 2.4. These are binary connectives because they require two operands. The negation connective, ~, is a unary operator on the one operand that follows it, as shown in Table 2.5.

A set of logical connectives is **adequate** if every truth function can be represented using only the connectives from the adequate set. Examples of adequate sets are { ~, ∧, ∨ }, { ~, ∧ }, {~, ∨ }, and { ~, → }.

A single element set is called a **singleton**. There are two adequate singleton sets. These are the NOT-OR (**NOR**) and the NOT-AND (**NAND**). The NOR set is { ↓ } and the NAND set is { | }. The " | " operator is called a **stroke** or **alternative denial**. It is used to deny that both p and q are true. That is, p | q affirms that at least one of the statements p or q is true. The **joint denial operator**, "↓", denies that either p or q is true. That is, p ↓ q affirms that both p and q are false.

Table 2.4 Truth Table of the Binary Logical Connectives

p	q	$p \wedge q$	$p \vee q$	$p \to q$	$p \leftrightarrow q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Table 2.5 Truth Table of Negation Connectives

p	~p
T	F
F	T

2.13 THE FIRST ORDER PREDICATE LOGIC

Although propositional logic is useful, it does have limitations. The major problem is that propositional logic can deal with only complete statements. That is, it cannot examine the internal structure of a statement. Propositional logic cannot even prove the validity of a syllogism such as:

```
All humans are mortal
All women are humans
Therefore, all women are mortal
```

In order to analyze more general cases, the **predicate logic** was developed. Its simplest form is the **first order** predicate logic and is the basis of logic programming languages such as PROLOG. In this section, we will use the term *predicate logic* to refer to first order predicate logic. Propositional logic is a subset of predicate logic.

Predicate logic is concerned with the internal structure of sentences. In particular, it is concerned with the use of special words called **quantifiers**, such as "all," "some," and "no." These words are very important because they explicitly quantify other words and make sentences more exact. All the quantifiers are concerned with "how many" and thus permit a wider scope of expression than does propositional logic.

2.14 THE UNIVERSAL QUANTIFIER

A universally quantified sentence has the same truth value for all replacements in the same domain. The **universal quantifier** is represented by the symbol ∀ followed by one or more arguments for the **domain variable**. The symbol ∀ is interpreted as "for every" or "for all." For example, in the domain of numbers:

```
(∀x)  (x + x = 2x)
```

states that for every x (where x is a number), the sentence x + x = 2x is true. If we represent this sentence by the symbol p, then it can be expressed even more briefly as:

```
(∀ x)  (p)
```

As another example, let p represent the sentence "all dogs are animals," as in:

```
(∀x)  (p) ≡(∀ x)  (if x is a dog → x is an animal)
```

The opposite statement is "no dogs are animals" and is written as:

```
(∀ x)  (if x is a dog → ~x is an animal)
```

It may also be read as:

```
Every dog is not an animal
All dogs are not animals
```

As another example, "all triangles are polygons" is written as follows:

```
(∀x)  (x is a triangle → x is a polygon)
```

and is read "for all x, if x is a triangle, then x is a polygon." A shorter way of writing logic statements involving predicates is using **predicate functions** to describe the properties of the subject. The above logic statement can also be written as:

```
(∀x)  (triangle(x) → polygon(x))
```

Predicate functions are usually written in a briefer notation using uppercase letters to represent the predicates. For example, let T = triangle and P = polygon. Then the triangle statement can be written even more briefly as:

```
(∀x)  (T(x) → P(x))    or    (∀y)  (T(y) → P(y))
```

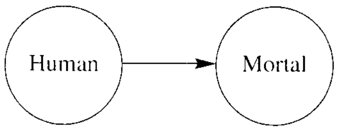
Note that any variables could be used in place of the dummy variables x and y. As another example, let H be the predicate function for human and M be the function for mortal. Then the statement that all humans are mortal can be written as:

```
(∀x)  (H(x) → M(x))
```

and is read that for all x, if x is human, then x is mortal. This predicate logic sentence can also be represented as a semantic net, as shown in Figure 2.16. It can also be expressed in terms of rules:

```
IF x is human
THEN x is mortal
```

Figure 2.16 Semantic Net Representation of a Predicate Logic Statement



The universal quantifier can also be interpreted as a conjunction of predicates about instances. As mentioned earlier, an instance is a particular case. For example, a dog named Sparkler is a particular instance of the class of dogs, which can be expressed as:

```
Dog(Sparkler)
```

where Dog is the predicate function and Sparkler is an instance.

A predicate logic sentence such as:

```
(∀x)  P(x)
```

can be interpreted in terms of instances a_i as:

$$P(a_1) \wedge P(a_2) \wedge P(a_3) \wedge \ldots P(a_N)$$

where the ellipses indicate that the predicates extend to all members of the class. This statement says that the predicate applies to all instances of the class.

Multiple quantifiers can be used. For example, the commutative law of addition for numbers requires two quantifiers, as in:

$$(\forall x)\ (\forall y)\ (x + y = y + x)$$

This states that "for every x and for every y, the sum of x and y equals the sum of y and x."

2.15 THE EXISTENTIAL QUANTIFIER

Another type of quantifier is the **existential quantifier**. An existential quantifier describes a statement as being true for at least one member of the domain. This is a restricted form of the universal quantifier, which states that a statement is true for all members of the domain. The existential quantifier is written as $\exists$ followed by one or more arguments. For example:

$$(\exists x)\ (x \cdot x = 1)$$
$$(\exists x)\ (\text{elephant}(x) \land \text{name}(\text{Clyde}))$$

The first sentence above states that there is some x whose product with itself equals 1. The second statement says there is some elephant with the name Clyde.

The existential quantifier may be read in a number of ways, such as:

```
there exists
at least one
for some
there is one
some
```

As another example,

$$(\forall x)\ (\text{elephant}(x) \rightarrow \text{four-legged}(x))$$

says that all elephants are four-legged. However, the statement that some elephants are three-legged is written with the logical AND and existential quantifier as follows:

$$(\exists x)\ (\text{elephant}(x) \land \text{three-legged}(x))$$

Just as the universal quantifier can be expressed as a conjunction, the existential quantifier can be expressed as a disjunction of instances, a_i:

$$P(a_1) \lor P(a_2) \lor P(a_3) \lor \ \ldots P(a_N)$$

Table 2.6 shows quantified statements and their negations for the example in which P represents "elephants are mammals." The numbers in parentheses identify the examples for the following discussion.

Table 2.6 Examples of Negated Quantifiers

Example		Meaning
(1a)	$(\forall x)(P)$	All elephants are mammals.
(1b)	$(\exists x)(\sim P)$	Some elephants are not mammals.
(2a)	$(\exists x)(P)$	Some elephants are mammals.
(2b)	$(\forall x)(\sim P)$	No elephants are mammals.

Examples (1a) and (1b) are negations of each other, as are (2a) and (2b). Notice that the negation of a universally quantified statement of (1a) is an existentially quantified statement of the negation of P as shown by (1b). Likewise, the negation of an existentially quantified statement of (2a) is the universally quantified statement of the negation of P as shown by (2b).

2.16 QUANTIFIERS AND SETS

Quantifiers may be used to define sets over a universe, U, as shown in Table 2.7.

The relation that A is a proper subset of B, written as $A \subset B$, means that while all elements in A are in B, there is at least one element in B that is not in A. Letting E represent elephants and M represent mammals, the set relation:

$$E \subset M$$

states that all elephants are mammals, but not all mammals are elephants. Letting G = grey and F = four-legged, the statement that all grey, four-legged elephants are mammals is written:

$$(E \cap G \cap F) \subset M$$

Table 2.7 Some Set Expressions and Their Logical Equivalents

Set Expression	Logical Equivalent
$A = B$	$\forall x\,(x \in A \leftrightarrow x \in B)$
$A \subseteq B$	$\forall x\,(x \in A \rightarrow x \in B)$
$A \cap B$	$\forall x\,(x \in A \wedge x \in B)$
$A \cup B$	$\forall x\,(x \in A \vee x \in B)$
A'	$\forall x\,(x \in U \mid \sim(x \in A))$
U (Universe)	T (True)
Ø (empty set)	F (False)

Using the following definitions, some examples of quantified sentences are shown:

```
E=elephants
R=reptiles
G=grey
```

```
F=four-legged
D=dogs
M=mammals
```

```
No elephants are reptiles
E ∩ R = Ø
Some elephants are grey
E ∩ G ≠ Ø
No elephants are grey
E ∩ G = Ø
Some elephants are not grey
E ∩ G′ ≠ Ø
All elephants are gray and four-legged
E ⊂ (G ∩ F)
All elephants and dogs are mammals
(E ∪ D) ⊂ M
Some elephants are four-legged and are grey
(E ∩ F ∩ G) ≠ Ø
```

As another analogy of sets and logic forms, de Morgan's laws are shown in Table 2.8. The equivalence symbol, ≡, the (biconditional) means that the statement on the left has the same truth value as the statement on the right. That is, the statements are equivalent.

Table 2.8 Set and Logic Forms of de Morgan's Laws

Set	Logic
$(A \cap B)' \equiv A' \cup B'$	$\sim(p \wedge q) \equiv \sim p \vee \sim q$
$(A \cup B)' \equiv A' \cap B'$	$\sim(p \vee q) \equiv \sim p \wedge \sim q$

2.17 LIMITATIONS OF PREDICATE LOGIC

Although predicate logic is useful in many types of situations, there are some types of statements that cannot be expressed in predicate logic using the universal and existential quantifiers. For example, the following statement cannot be expressed in predicate logic:

```
Most of the class received A's
```

In this statement the quantifier *Most* means more than half.

The *Most* quantifier cannot be expressed in terms of the universal and existential quantifiers. To implement *Most,* a logic must provide some predicates for counting, as in fuzzy logic described in Chapter 5. Another limitation of predicate logic is expressing things that are sometimes, but not always true. This problem can also be solved by fuzzy logic. is However, introducing counting also introduces more complications into the logic system and makes it more like mathematics.

2.18 SUMMARY

In this chapter we have discussed the elements of logic, knowledge representation and some methods for representing knowledge. Knowledge representation is of major importance in expert systems. Fallacies were discussed because it is so important for the knowledge engineer to understand the rules of the domain and not confuse the form of knowledge with semantics. Unless formal rules are specified, an expert system may not yield valid conclusions, which could spell disaster in mission-critical systems on which human life and property depend.

From a logical view, knowledge can be classified in a number of ways such as a priori, a posteriori, procedural, declarative, and tacit. Production rules, objects, semantic nets, schemata, frames, and logic are common methods by which knowledge is represented in expert systems. Each of these paradigms has advantages and disadvantages. Before designing an expert system you should decide which is the best paradigm for the problem to be solved. Rather than trying to use one tool for all problems, pick the best tool. The advantage of CLIPS in being able to represent knowledge by both objects as well as rules was covered. Many references to logic, knowledge, and fallacies are given in Appendix G. Appendices A, B, and C contain collections of useful equivalences, quantifiers, and set properties.

PROBLEMS

2.1 Draw a semantic net for computers using AKO and IS-A links. Consider the classes of microcomputer, mainframe, supercomputer, computing system, dedicated, general purpose, board-level, computer-on-a-chip, single processor, and multiprocessor. Include specific instances.

2.2 Draw a semantic net for computer communications using AKO and IS-A links. Consider the classes of local area net, wide area net, token ring, star, centralized, decentralized, distributed, modems, telecommunications, newsgroups, and electronic mail. Include specific instances.

2.3 Draw a frame system for the building in which you are attending classes. Consider offices, classrooms, laboratories, and so forth. Include instances with filled slots for one type of each frame such as office and classroom.

2.4 Draw an action frame system telling what to do in case of hardware failure for your computer system. Consider disk crash, power supply, CPU, and memory problems.

2.5 Draw the Venn diagram and write the set expression for

(a) Exclusive-OR of two sets A and B consists of all elements that are in one, but not both sets. The exclusive-OR is also called the **symmetric set difference** and is symbolized by the / sign. For example:
$\{1,2\} / \{2,3\} = \{1,3\}$

(b) **Set difference** of two sets, symbolized by "–" consists of all the elements in the first set that are not also in the second. For example:
$\{1,2\} - \{2,3\} = \{1\}$
where $\{1,2\}$ is the first set and $\{2,3\}$ is the second.

2.6 Write the truth tables and determine which of the following are tautologies, contradictions, or **contingent statements**, and which are neither. For (a) and (b), first express the statements with logic symbols and connectives.

(a) If I pass this course and make an A then
I pass this course or I make an A.

(b) If I pass this course then I make an A
and
I pass this course and I do not make an A.

(c) $((A \wedge {\sim}B \rightarrow (C \wedge {\sim}C)) \rightarrow (A \rightarrow B)$

(d) $(A \rightarrow B) \wedge ({\sim}B \vee C) \wedge (A \wedge {\sim}C)$

(e) $A \rightarrow {\sim}B$ (contingent)

2.7 Two sentences are logically equivalent if and only if they have the same truth value. Thus if A and B are any statements, the biconditional statement,

$A \leftrightarrow B$, or the equivalence, $A \equiv B$

will be true in every case, giving a tautology. Determine if the two sentences below are logically equivalent by writing them using logical symbols and determine if the truth table of their biconditional is a tautology.

If you eat a banana split, then you cannot eat a pie.
If you eat a pie, then you cannot eat a banana split.

2.8 Write the logical equivalent corresponding to set difference and symmetric set difference.

2.9 Show that the following are identities for any sets A, B, and C and Ø as the null set:

(a) $(A \cup B) \equiv (B \cup A)$

(b) $(A \cup B) \cup C \equiv A \cup (B \cup C)$

(c) $A \cup \emptyset \equiv A$

(d) $A \cap B \equiv B \cap A$

(e) $A \cap A' \equiv \emptyset$

2.10 Write the following in quantified form:

(a) All dogs are mammals.

(b) No dog is an elephant.

(c) Some programs have bugs.

(d) None of my programs have bugs.
(e) All of your programs have bugs.

2.11 The **power set**, P(S), of a set S is the set of all elements that are subsets of S. P(S) will always have at least the null set, Ø, and S as members.

(a) Find the power set of A = {2, 4, 6}.

Meaning	Definition
either p or q	$(p \vee q) \wedge \sim(p \wedge q)$
neither p nor q	$\sim(p \vee q)$
p unless q	$\sim q \to p$
p because q	$(p \wedge q) \wedge (q \to p)$
no p is q	$p \to \sim q$

(b) For a set with N elements, how many elements does the power set have?

2.12 (a) Write the truth table for the following:
(b) Show that $(p \vee q) \wedge \sim(p \wedge q) \equiv p/q$ where / is the exclusive OR.

2.13 (a) Write the NOR and NAND truth tables.
(b) Prove that $\{\downarrow\}$ and $\{\mid\}$ are adequate sets by expressing ~, ∧, and ∨ in terms of ↓ and then in terms of | by constructing truth tables to show the logical equivalences as follows:

$\sim\sim p \equiv p$

$(p \wedge q) \equiv (p \downarrow p) \downarrow (q \downarrow q)$

$\sim p \equiv p \mid p$

$(p \vee q) \equiv (p \mid p) \mid (q \mid q)$

(c) Since $p \to q \equiv \sim(p \wedge \sim q)$, express $p \to q$ in terms of ↓.
(d) What is the advantage and disadvantage of using adequate singleton sets in terms of (i) notation and (ii) construction of chips for electronic circuits?

2.14 What are the advantages and disadvantages of designing an expert system with knowledge about several domains?

2.15 Explain how you would boil water differently in Denver than in Houston if you were cooking an egg that should be hard-boiled. Is this a problem in logic or physics?

2.16 Given the following PROLOG statements, prove that Tom is his own grandfather:

```
mother(pat,ann).   ; pat is the mother of ann
parents(jim,ann,tom)  ; jim and ann are parents of tom
surrogatemother(pat,tom).  ;    pat is the surrogatemother of tom
```

BIBLIOGRAPHY

(Bergadano 96). *Inductive Logic Programming*, The MIT Press, 1996.

(Brewka 97). Gerhard Brewka, *Principles of Knowledge Representation*, CSLI Publications, 1997.

(Brachman 91). R. Brachman, D. McGuinness, P. Patel-Schneider, A. Borgida, and L. Resnick, Living with CLASSIC: When and How to Use a KL-ONE-Like Language, *Principles of Semantic Networks*, Morgan Kaufman, pp. 401–456, May, 1991.

(Huth 04). Michael Huth and Mark Ryan, Logic in Computer Science, 2nd Edition, Cambridge University Press, 2004. NOTE: Also see software in following list for their book.

(Jackson 99). Peter Jackson, *Introduction to Expert Systems*, Addison-Wesley, Third Edition, 1999.

(Leake 96). Ed. By David Leake et al., *Case-Based Reasoning*, AAAI Press/MIT Press 1996.

(Kahane 03). Howard Kahane & Paul Tidman, *Logic and Philosophy: A Modern Introduction*, 9th edition, Wadsworth, 2003.

(Jacquette 01). Dale Jacquette, *Symbolic Logic*, Wadsworth, 2001.

(Saratchandran 96). P. Saratchandran, et al., *Parallel Implementations of Backpropagation Neural Networks on Transputers: A Study of Training Set Parallelism, Progress in Neural Processing 3*, World Scientific Pub. Co, July 1996

(Sowa 00). J.F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks-Cole, 2000.

(Russell 03). Stuart Russell and Peter Norvig, *Artificial Intelligence*, Second Edition, by Pearson Education, 2003.

(Werbos 94). Paul Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting: Adaptive and Learning Systems for Signal Processing*, Wiley-Interscience, 1994.