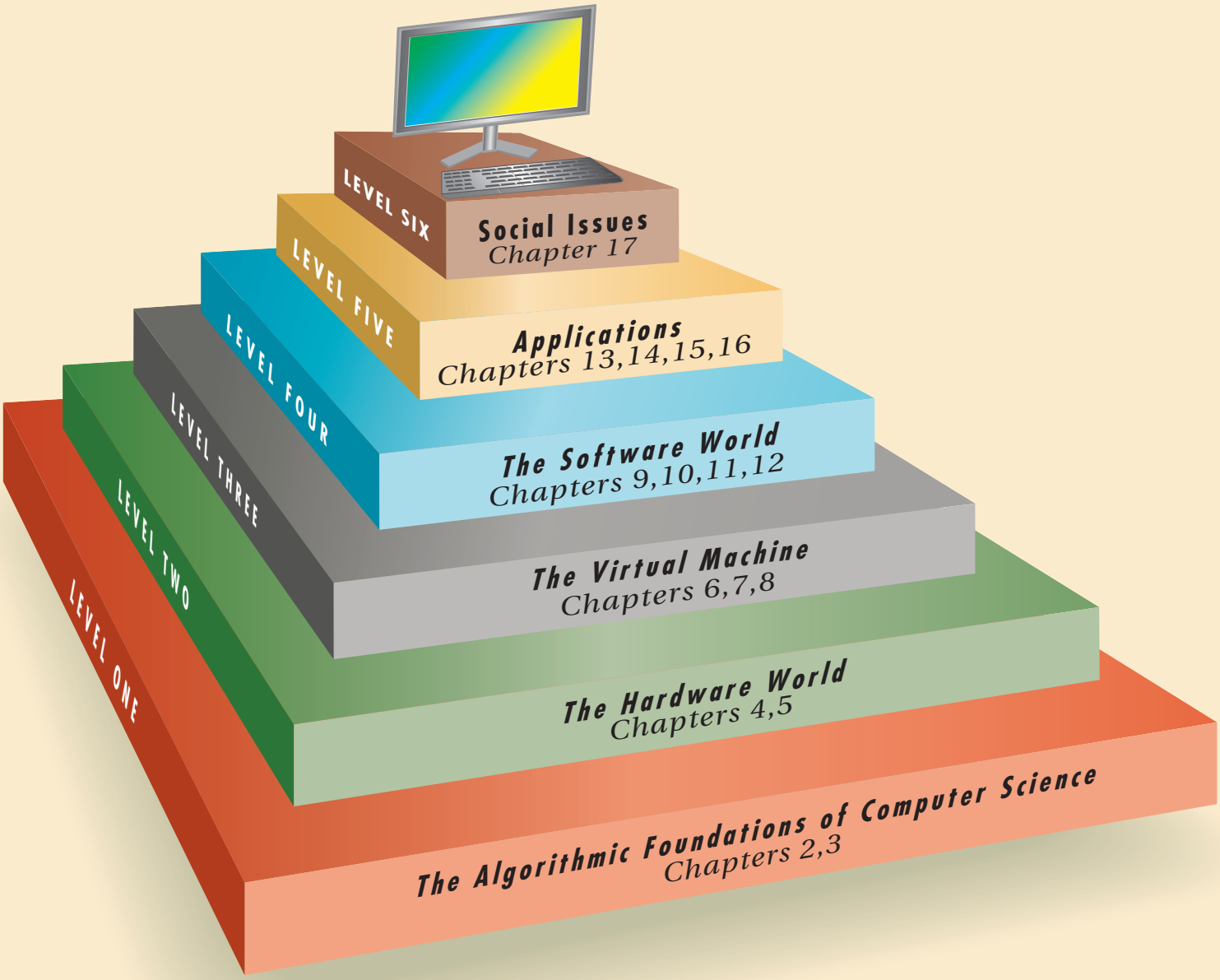

FIFTH EDITION
INVITATION TO
COMPUTER SCIENCE
G. Michael Schneider • Judith L. Gersting

5TH EDITION

Invitation to Computer Science


LEVEL SIX
Social Issues
Chapter 17
LEVEL FIVE
Applications
Chapters 13,14,15,16
LEVEL FOUR
The Software World
Chapters 9,10,11,12
LEVEL THREE
The Virtual Machine
Chapters 6,7,8
LEVEL TWO
The Hardware World
Chapters 4,5
LEVEL ONE
The Algorithmic Foundations of Computer Science
Chapters 2,3

5TH EDITION

Invitation to Computer Science

G. Michael Schneider
Macalester College

Judith L. Gersting
University of Hawaii, Hilo

Contributing author:
Keith Miller
University of Illinois, Springfield

COURSE TECHNOLOGY
CENGAGE Learning

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

Invitation to Computer Science, Fifth Edition
G. Michael Schneider and Judith L. Gersting

Executive Editor: Marie Lee

Acquisitions Editor: Amy Jollymore

Senior Product Manager: Alyssa Pratt

Development Editor: Deb Kaufmann

Editorial Assistant: Julia Leroux-Lindsey

Marketing Manager: Bryant Chrzan

Content Project Manager: Jennifer K. Feltri

Art Director: Faith Brosnan

Cover Designer: RHDG/Tim Heraldo

Cover Artwork: Fotolia.com (Royalty Free), Image # 375162

Compositor: Integra

ISBN-13: 978-0-324-78859-4
ISBN-10: 0-324-78859-2

Course Technology
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at:
international.cengage.com/region

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

For your lifelong learning solutions, visit **course.cengage.com**
Visit our corporate website at **cengage.com**.

Printed in Canada
1 2 3 4 5 6 7 13 12 11 10 09

BRIEF CONTENTS

CHAPTER 15

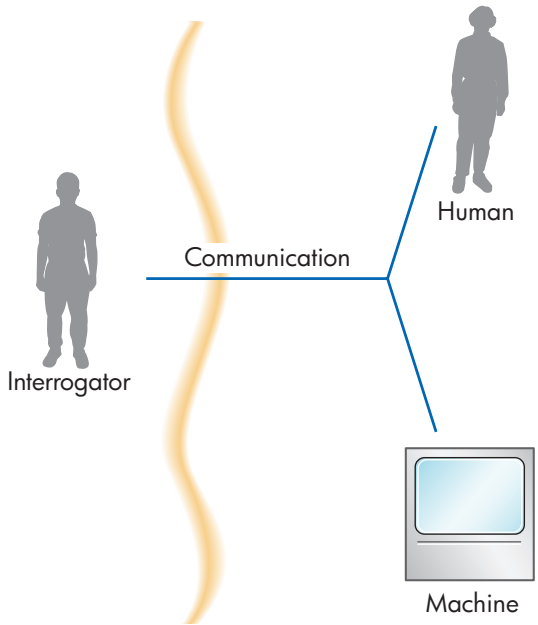
Artificial Intelligence

15.1 Introduction

Artificial intelligence (AI) is the branch of computer science that explores techniques for incorporating aspects of intelligence into computer systems. This definition, however, raises more questions than it answers. What really is "intelligence"? Is it a uniquely human attribute? If a computer system exhibits behavior that we might characterize as intelligent, does that make it truly intelligent? What sorts of behaviors demonstrate intelligence?

Alan Turing, whose investigations into the fundamental nature of computation led to the Turing machine (Chapter 12), was also interested in artificial intelligence. In 1950, before the term *artificial intelligence* was coined, he proposed a test for intelligent behavior of machines. The **Turing test** allows a human to interrogate two entities, both hidden from the interrogator (Figure 15.1). One entity is a human and the other a machine (a computer). The interrogator can ask the entities questions and receive their responses. The communication is carried on in some form that does not alone reveal which entity is the computer; for example, the interrogator's questions could be typed on a keyboard and the responses printed out. If, as a result of this questioning, the interrogator is unable to determine which entity is the human and which the computer,

FIGURE 15.1
The Turing Test



To Whom Am I Speaking?

The Loebner Prize Competition has been held annually since 1991. This competition is essentially a Turing test. Dr. Hugh Loebner, a sociologist and inventor, has posted a $100,000 grand prize and a gold medal for the first computer program whose responses are indistinguishable from a human's. Though the grand prize remains unclaimed, a prize of $2,000 and a bronze medal is awarded each year to the computer program judged to be "most human" in that year's competition.

One of the most famous early AI experiments occurred in the laboratory of Professor Joseph Weizenbaum at MIT in 1966. A program named ELIZA conversed with people in the fashion of a psychotherapist, picking up on key words and echoing them back in a sentence. This was quite mechanical and there was no understanding involved. For example, an ELIZA conversation might go something like this:

Human: I am not feeling well today.
Computer: I AM SORRY TO HEAR THAT YOU ARE NOT FEELING WELL.
Human: I had a fight with my wife.
Computer: TELL ME ABOUT YOUR WIFE.
Human: The house ate my zebra.
Computer: TELL ME ABOUT YOUR ZEBRA.

Weizenbaum was horrified at the ease with which people who interacted with ELIZA became convinced that they were talking with a real person. Today, there are many "chatbot" programs available on the Web. The 2007 winning entry in the Loebner Prize Competition was Ultra Hal Assistant, a chatbot program that can not only hold a conversation but remember and remind you of appointments, tell you the local weather report, give you the definition of a word, or do some Web searching for you. Ultra Hal also appears to learn; consider this sample conversation (*http://zabaware.com/loebner.html*):

User: If it rains hard, then the roads get slick.
Hal: I will remember that one follows the other. Caught in a rainstorm, it's time to seek shelter.
User: If the roads get slick, then drivers need to slow down.
Hal: I see the relationship. It would be nice to get rain when we need it.
User: If drivers need to slow down, then traffic moves more slowly.
Hal: Yes, I get that clearly.
User: Ok, so what happens when it rains hard?
Hal: Traffic moves more slowly because drivers need to slow down.

While this conversation seems much more intelligent than the one with ELIZA, we can see the same type of if-then reasoning that the Prolog inference engine uses (Chapter 10).

then the computer has exhibited sufficiently human intelligence to pass the Turing test. This test does not explore the nature of human intelligence in a deep philosophical way; it merely says that if a machine exhibits behavior indistinguishable from that of a human, then how can we tell the difference—a question that has been explored more than once in science fiction.

In Chapter 13 we discussed models of physical phenomena (objects falling under the force of gravity, customers in a store). Artificial intelligence is the construction of computer models of human intelligence. Just as we learned in Chapter 13 that a model cannot capture all aspects of the system it represents, so artificial intelligence cannot capture all aspects of "intelligence." Although science fiction conjures scenarios of thinking computers controlling the world, the advances in artificial intelligence have been more modest. Nonetheless, artificial intelligence has produced practical results in natural language understanding, robotics, game playing, problem-solving, machine learning, expert systems, pattern recognition, and other areas. And conversely, attempts to model intelligence within a computer have in turn made contributions to **cognitive science**, the study of how we as humans think and learn.

15.2 A Division of Labor

To understand better what artificial intelligence is all about, let's consider a division of task types. Humans can perform a great variety of tasks, but we'll divide them into three categories, representative but by no means exhaustive:

- *Computational tasks*
  - Adding a column of numbers
  - Sorting a list of numbers into numerical order
  - Searching for a given name in a list of names
  - Managing a payroll
  - Calculating trajectory adjustments for the space shuttle
- *Recognition tasks*
  - Recognizing your best friend
  - Understanding the spoken word
  - Finding the tennis ball in the grass in your backyard
- *Reasoning tasks*
  - Planning what to wear today
  - Deciding on the strategic direction a company should follow for the next 5 years
  - Running the triage center in a hospital emergency room after an earthquake

Algorithmic solutions exist for computational tasks (we devised algorithms for sorting and searching in the early chapters of this book). As humans, we can, in principle at least, follow these step-by-step instructions. Computational tasks are also tasks for which accurate answers must be found—sometimes very quickly—and that's where we as humans fall down. We make mistakes, we get bored, and we aren't very speedy. Computers are better (faster and more accurate) at performing computational tasks, provided they are given programs that correctly embody the algorithms. Throughout most of this book, with its emphasis on algorithms, we've been talking about procedures to solve computational tasks, how to write those procedures, how to get the computer to execute them, and so on.

Humans are better at recognition tasks. We should perhaps expand the name of this task type to sensory/recognition/motor-skills tasks, because we receive information through our senses (primarily seeing and hearing), we recognize or "make sense of" the information we receive, and we often respond to the information with some sort of physical response that involves controlled movement. Although we wait until elementary school to learn how to add, an infant a few weeks old, on seeing its mother's face, recognizes that face and smiles; soon that infant understands the spoken word. You spot the tennis ball in the yard even though it is green and nestled in among other green things (grass, dandelions). You register whether the tennis ball is close or far away, and you manipulate your legs and feet to propel you in the right direction.

How do we do these things? Traditional step-by-step procedural algorithms don't seem to apply, or if they do, we don't know what those

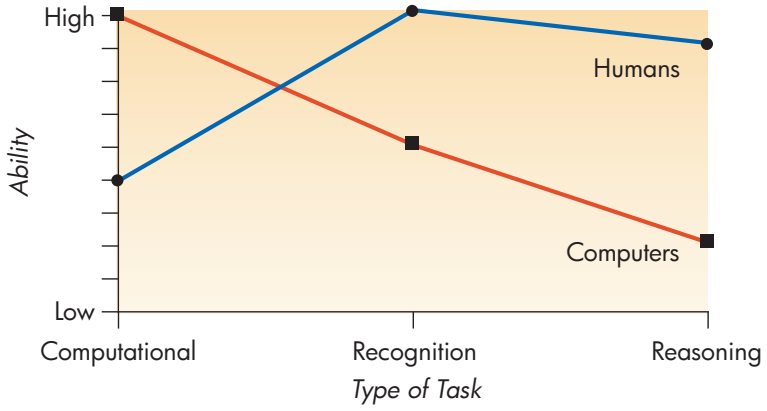
algorithms are. Rather, it seems that we as humans succeed at these tasks by processing a huge amount of data and then matching the results against an even larger storehouse of data based on our past experiences. Consider the task of recognizing your best friend. You have, in effect, been shown a number of "pictures" of your friend's face that seem to be "burned into" your memory, along with pictures of the faces of everyone else you know well. When you see your friend, you sort through your mental picture file until you come to a match. It is a bit more complicated than that, however, because if you encounter your friend's sister, you may know who it is even though you have never met her before. You find not an exact match to one of the images in your mental picture file, but a close approximation. Approximation, unlike the exactitude required in computational tasks, is good enough. These complex recognition tasks that we find so easy are difficult for computers to perform.

When humans perform reasoning tasks, they are also using a large storehouse of experience. This experience involves not just images but also cause and effect situations. You know that you should wear a coat when it's cold because you've experienced discomfort in cold weather when you didn't wear a coat. This could be considered "mere" commonsense reasoning, but getting a computer to mimic common sense, to say nothing of higher-order conceptual, planning, or reasoning tasks, is extremely challenging. There may be no "right" answer to such tasks, and the way humans arrive at their respective answers sometimes seems ambiguous or based at least in part on intuition, which may be just another name for knowledge or reasoning that we don't yet understand.

Figure 15.2 summarizes what we've outlined as the relative capabilities of humans and computers in these three types of tasks. Computers fall below humans where procedural algorithms either don't work or aren't known, and where there seems to be a high level of complexity and perhaps approximation or ambiguity. Artificial intelligence seeks ways to improve the computer's ability to perform recognition and reasoning tasks, and we'll look at artificial intelligence approaches in these two areas in the rest of this chapter. As mentioned earlier, however, both types of tasks seem to require a storehouse of information—images, past experiences, and the like—for which we'll use the general term *knowledge*. Therefore, we'll first look at various approaches to representing knowledge.

FIGURE 15.2

Human and Computer Capabilities



15.3 Knowledge Representation

We can consider **knowledge** about some topic as a body of facts or truths. For the computer to make use of that knowledge, there must be some representational form in which the knowledge is stored within the computer. (At the lowest level, of course, only 0s and 1s are stored within the computer, but strings of 0s and 1s are organized and interpreted at a higher level of abstraction—as integers or characters, for example.) For computational tasks, the relevant knowledge is often isolated numeric or textual items. This is the data that we've manipulated with procedural programs. What about more complex knowledge?

There are many workable representation schemes; let's consider four possibilities.

1. *Natural language.* A paragraph or a page of text that contains all the knowledge we are trying to capture is written in English, French, Spanish, or some other natural language. Here is an example:

   Spot is a brown dog and, like any dog, has four legs and a tail. Also like any dog, Spot is a mammal, which means Spot is warm-blooded.

   Note that although this representational form is text, it is text in a different sense from the character strings that are used in computational tasks. Here it is not simply the strings of characters that are important but also the meaning that those strings of characters convey. When reading a natural language paragraph, we use our understanding of the richness of the language's vocabulary to extract the meaning. Some researchers believe that the words we read or hear do not actually communicate meaning, but merely act as "triggers" to meanings stored in our brains.

2. *Formal language.* A formal language sacrifices richness of expression for precision of expression. Attributes and cause and effect relationships are more explicitly stated. A formal language version of the foregoing natural language paragraph might look like this:

   Spot is a dog.

   Spot is brown.

   Every dog has four legs.

   Every dog has a tail.

   Every dog is a mammal.

   Every mammal is warm-blooded.

   The term *language* was used in Chapter 11 to mean the set of statements derivable by using the rules of a grammar. But here the term **formal language** means the language of formal logic, usually expressed more symbolically than we have done in our example. In the usual notation of formal logic, we might use *dog*(*x*) to symbolize that the symbolic entity *x* has the attribute of being a dog and *brown*(*x*) to mean that *x* has the attribute of being brown. Similarly *four-legged*(*x*), *tail*(*x*), *mammal*(*x*), and *warm-blooded*(*x*) could symbolize that *x* has these various attributes. The specific entity Spot could be represented

by S. Then $dog(S)$ would mean that Spot has the attribute of being a dog. Cause and effect relationships are translated into "if-then" statements. Thus, "Every dog has four legs" is equivalent to "For every x, if x is a dog, then x has four legs." An arrow symbolizes cause and effect (if-then); "If x is a dog, then x has four legs" would be written symbolically as

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

To show that every x that has the dog property also has the four-legged property, we would use a *universal quantifier*, $(\forall x)$, which means "for every x." Therefore,

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

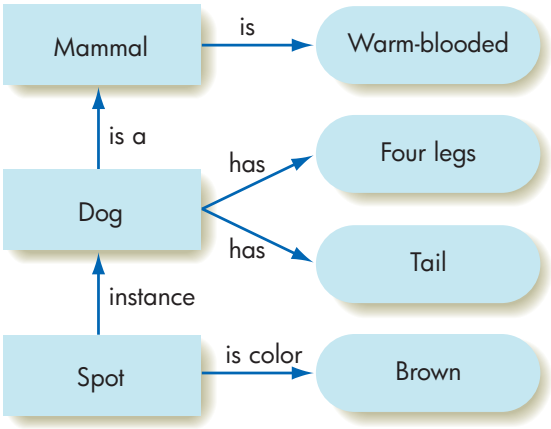
means "For every x, if x is a dog, then x has four legs" or "Every dog has four legs." Symbolically, the preceding six formal language statements become

Natural Language Statement	*Symbolic Representation*
Spot is a dog.	$dog(S)$
Spot is brown.	$brown(S)$
Every dog has four legs.	$(\forall x)(dog(x) \rightarrow four\text{-}legged(x))$
Every dog has a tail.	$(\forall x)(dog(x) \rightarrow tail(x))$
Every dog is a mammal.	$(\forall x)(dog(x) \rightarrow mammal(x))$
Every mammal is warm-blooded.	$(\forall x)(mammal(x) \rightarrow warm\text{-}blooded(x))$

The use of formal languages represents one of the major approaches to building artificial intelligence systems. Intelligent behavior is achieved by using symbols to represent knowledge and by manipulating these symbols according to well-defined rules. We'll see an example of this when we discuss expert systems later in this chapter.

3. *Pictorial.* Information can be stored in pictorial form as an image—a grid of pixels that have attributes of shading and color. Using this representation we might have a picture of Spot, showing that he is brown and has four legs and a tail. We might have some additional labeling that says something like, This is Spot, the dog. This visual representation might contain additional knowledge about Spot's appearance that is not embodied in the natural language paragraph or the formal language statements, but it would also fail to capture the knowledge that Spot is a mammal and that mammals are warm-blooded. It also wouldn't tell us that all dogs have four legs and a tail. (After all, a photo of a three-legged dog does not tell us that all dogs have three legs.)
4. *Graphical.* Here we are using the term *graphical* not in the sense of "visual" (we have already talked about pictorial representation) but in the mathematical sense of a graph with nodes and connecting arcs. Figure 15.3 is such a graph, also called a **semantic net**, for our dog example. In the terminology of object orientation that was a feature of the programming language(s) of Chapter 9, the rectangular nodes represent classes or objects, the oval nodes represent properties, and the arcs represent relationships. The "is a" relationship represents a subclass of a class that inherits properties

FIGURE 15.3
A Semantic Net Representation



from the parent class; "dog" is a subclass of "mammal," and any dog object inherits all the properties of mammals in general, such as being warm-blooded. Objects from the dog class may also have properties of their own. The "instance" relationship shows that something is an object of a class; Spot is a particular object from the dog class and may have a unique property not necessarily shared by all dogs.

Any knowledge representation scheme that we select must have the following four characteristics:

1. *Adequacy*. The representation method must be adequate to capture all of the relevant knowledge. Because of its rich expressive powers, a natural language representation will surely capture a lot of knowledge. However, it may be difficult to extract exactly what that knowledge is. One may have to wade through a lot of unnecessary verbiage, and one must also understand the nuances of meaning within the natural language. A formal language representation has the advantage of extracting the essentials.
2. *Efficiency*. We want the representational form to be minimalist, avoiding redundant information wherever possible. This means allowing some knowledge that is not explicitly represented to be inferred from the knowledge that is explicitly represented. In the preceding example, it is easy to infer from the natural language, the formal language, or the semantic net that because Spot is a dog, he has four legs and a tail and also is a mammal and therefore warm-blooded. This knowledge, as we have said, is not captured in the pictorial format. On the other hand, it would take a much longer natural language paragraph to describe all the additional knowledge about Spot that is captured in the picture.
3. *Extendability*. It should be relatively easy to extend the representation to include new knowledge. For example, the semantic net can easily be extended to tack on another "dog" instance. It would also be easy to capture the fact that dogs have two eyes or that mammals do not lay eggs; these properties can simply be plugged in as new ovals connected into the network.

PRACTICE PROBLEM

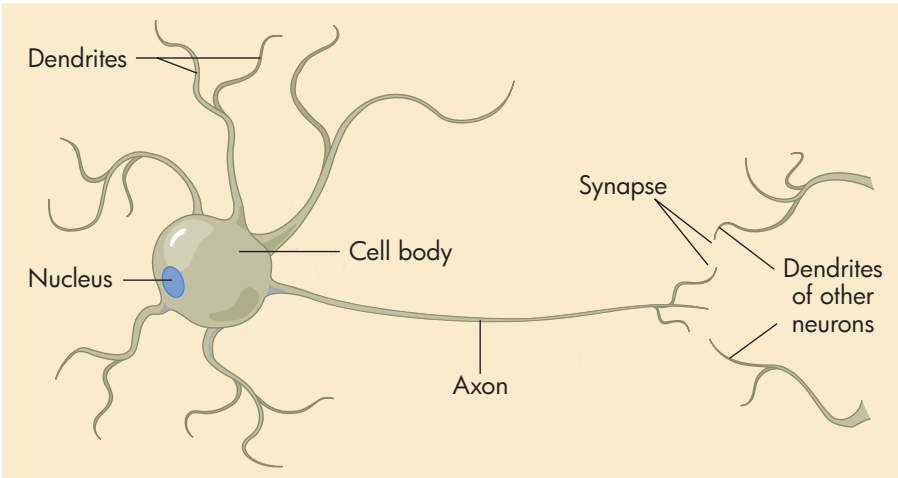
Write a natural language paragraph that describes the concept of a hamburger. Now draw a semantic net that incorporates the same knowledge as your natural language description. Which one is easier for you to produce?

4. *Appropriateness.* The representation scheme used should be appropriate for the knowledge domain being represented. For example, a pictorial representation scheme would appear to be the most appropriate way to represent the knowledge base for a problem dealing with recognition of visual images. We saw before that a pictorial representation is probably not appropriate for the kind of knowledge about Spot that is difficult to display visually. The level of granularity needed for the intended application might also influence the appropriateness of a particular scheme. Is a given pictorial representation sufficient, or do we need to "zoom in" and expose more detail? The appropriate representational form for knowledge therefore depends on the knowledge to be captured and on the type of task for which the knowledge is to be used.

15.4 Recognition Tasks

If artificial intelligence aims to make computers "think" like humans, then it is natural to investigate and perhaps attempt to mimic the way the human brain functions. It is estimated that the human brain contains about 10^{12} neurons (that's 1 trillion, or 1,000,000,000,000). Each **neuron** is a cell capable of receiving stimuli, in the form of electrochemical signals, from other neurons through its many **dendrites** (Figure 15.4). In turn, it

FIGURE 15.4
A Neuron



can send stimuli to other neurons through its single **axon**. The axon of a neuron does not directly connect with the dendrites of other neurons; rather, it sends signals over small gaps called **synapses**. Some of the synapses appear to send the neuron activating stimuli, whereas others seem to send inhibiting stimuli. A single neuron collects all the stimuli passing through all the synapses around its dendrites. The neuron sums the activating (positive) and inhibiting (negative) stimuli it receives and compares the result with an internal "threshold" value. If the sum equals or exceeds the threshold value, then the neuron "fires," sending its own signal down its axon to affect other neurons.

Each neuron can be thought of as an extremely simple computational device with a single on/off output. The power of the human brain lies in the vast number of neurons, the many interconnections between them, and the activating/inhibiting nature of those connections. To borrow a term from computer science, the human brain uses a **connectionist architecture**, characterized by a large number of simple "processors" with multiple interconnections. This contrasts quite noticeably with the Von Neumann architecture discussed in Chapter 5 and is still the basis for most computers today. In that model there are a small number (maybe only one) of very powerful processors with a limited number of interconnections between them.

In some areas of the brain, an individual neuron may collect signals from as many as 100,000 other neurons and send signals to an equally large number of other neurons. This extensive parallelism is evidently required because of the relatively slow time frame within which a neuron fires. In the human brain, neurons operate on a time scale of milliseconds (thousandths of a second), as opposed to the nanoseconds (billionths of a second) in which computer operations are measured, a difference of 6 orders of magnitude. In a human processing task that takes about 1/10 second (recognition of your friend's face), the number of steps that can be executed by a single neuron would be on the order of 100. To carry out the complexity of a recognition task, then, requires the parallel activities of a large number of neurons executing cooperatively within this short time frame. In addition, massive parallelism supplies redundancy so that information is not stored only in one place but is shared within the network of neurons. Thus, the deterioration of a limited number of single neurons (a process that happens constantly as biological cells wear out) does not cause a failure of the information processing capabilities of the network.

Artificial intelligence systems for recognition tasks have used this connectionist approach. **Artificial neural networks**, usually just called **neural networks**, can be created by simulating individual neurons in hardware and connecting them in a massively parallel network of simple devices that act somewhat like biological neurons. (Recall our discussion in Chapter 5 of parallel processing and non–Von Neumann architectures.) Alternatively, the effect of a neural network may be simulated in software on an ordinary sequential-processing computer. In either case, each neuron has a threshold value, and its incoming lines carry weights that represent stimuli. The neuron fires when the sum of the incoming weights equals or exceeds its threshold value; the input lines are activated via the firing of other neurons.

FIGURE 15.5
One Neuron with Three Inputs

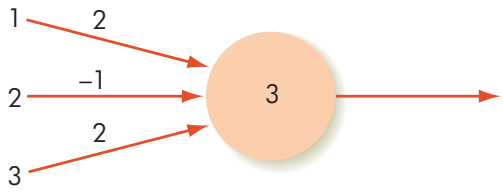


Figure 15.5 represents a neuron with a threshold value of 3 and three input lines with weights of 2, −1, and 2, respectively. If all three input lines are activated, the sum of the incoming signals is $2 + (-1) + 2 = 3$ and the neuron fires. It also fires if only lines 1 and 3 are activated, because the sum of the incoming signals is then $2 + 2 = 4 > 3$. Any other combination of activated input lines cannot carry sufficient stimulation to fire the neuron. (Real, biological neurons fire with intensities that vary through a continuous range but, as usual, our simplified computer representation of such analog values uses a set of discrete values.)

Figure 15.6 depicts a neural net with an input layer and an output layer of neurons. An input value x_i is presented to neuron N_j in the input layer via a line with signal strength $x_i \times w_{ij}$. The values of x_i are usually binary (0 or 1), so that this line carries a signal of either 0 when x_i is 0, or the weight w_{ij} when x_i is 1. The weights to the input neurons, as well as the weights from the input layer to the output layer, can be positive, negative, or zero.

FIGURE 15.6
Neural Network Model

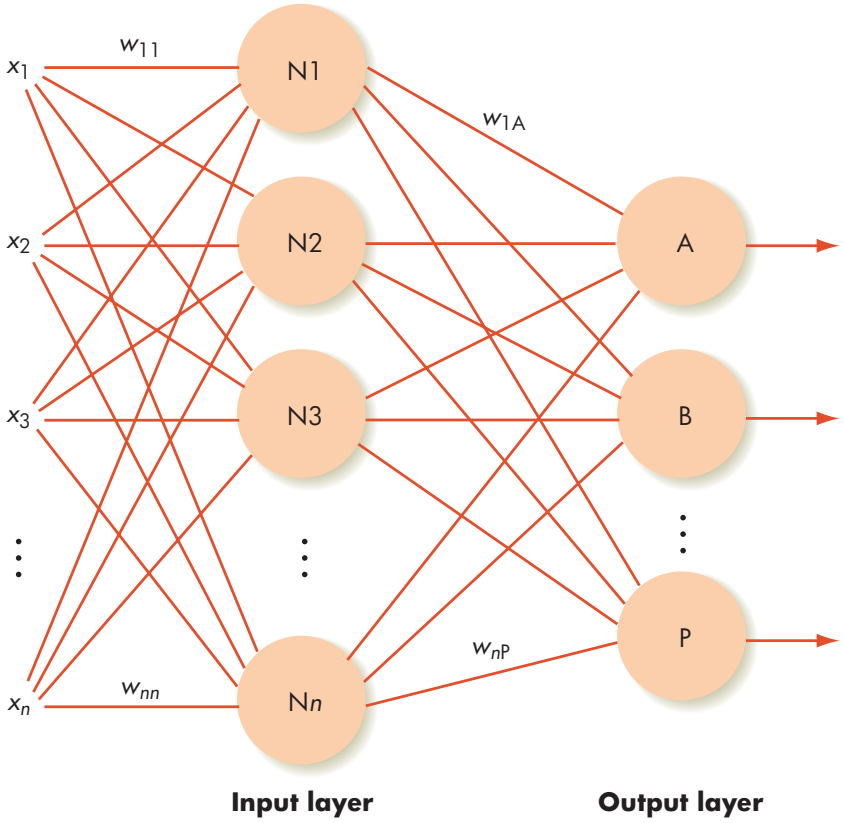
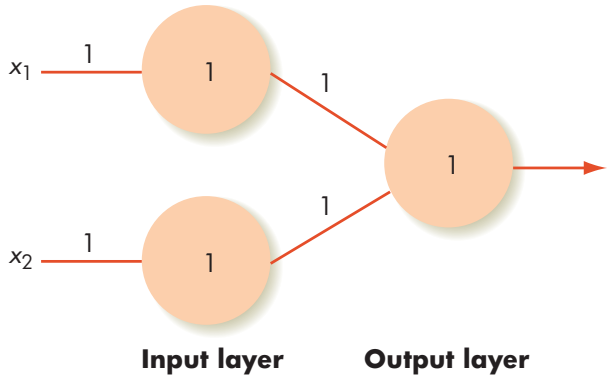


FIGURE 15.7

A Simple Neural Network—OR Gate



In the neural network shown in Figure 15.7, we have eliminated connections of weight 0. Here x_1 and x_2 have binary values of 0 or 1. If x_1 or x_2 or both have the value 1, then a signal of 1 is passed to one or both of the neurons in the input layer, causing one or both of them to fire, which causes the single neuron in the output layer to fire and produce an output of 1. If both x_1 and x_2 have the value 0, then neither neuron in the input layer fires, the single neuron in the output layer does not fire, and the network output is 0. This neural network is acting like an OR gate.

It turns out to be impossible to build such a network to represent the Boolean operation called exclusive OR, or XOR, whose truth table is shown in Figure 15.8. Here the output is true (1) when one or the other input is true, but not when both are true. In Figure 15.9, no matter what values we give for the weights and thresholds, it is not possible to generate this behavior. If exactly one input signal of 1 is enough to fire the output neuron, which is the desired behavior, then two input signals of 1 can only increase the tendency for the output neuron to fire. To represent the XOR operation requires a "hidden layer" of neurons between the input and output layers. Neural networks with a hidden layer of neurons are useful for recognition tasks, where we want a certain pattern of output signals for a certain pattern of input signals. The XOR network, for example, recognizes when its two binary inputs do not agree.

Conventional computer processing works on a knowledge base where the information is stored as data in specific memory cells that can be accessed by the program as needed. In a neural network, both the knowledge representation and also the "programming" are stored in the network itself as the weights of the connections and the thresholds of the neurons. If you want to build a neural network that performs in a certain way, how do you determine

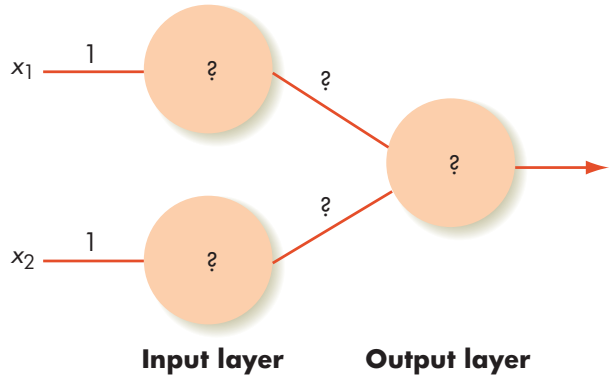
FIGURE 15.8

The Truth Table for XOR

Inputs		Output
x_1	x_2	
0	0	0
1	0	1
0	1	1
1	1	0

FIGURE 15.9

An Attempt at an XOR Network



these values? In a simple network, trial and error can produce a solution, but such is not the case for a network with thousands of neurons. Fortunately, the right answer doesn't have to be found the first time. Remember that neural networks are modeled on the human brain; you learned to recognize your best friend through repeated "learning experiences" that modified your knowledge base until you came to associate certain features or characteristics with that individual.

Similarly, a neural network can learn from experience by modifying the weights on its connections (even making some connections "disappear" by assigning them 0 weights). A network can be given an initial set of weights and thresholds that is simply a first guess. The network is then presented with **training data**, for which the correct outputs are known. The actual output from the network is compared to the correct output for one set of input values from the training data. For those output neurons that produce correct values, their threshold values and the weights on their inputs do not change. Output neurons that produce erroneous values can err in one of two ways. If an output neuron fires when it is not supposed to, then the positive (excitatory) input values coming into it are adjusted downward, and the negative (inhibitory) weights coming into it are adjusted upward. If it fails to fire when it is supposed to, the opposite adjustment is made. But before these adjustments take place, information on the errors is passed back from each erroneous output neuron to the neurons in the hidden layer that are connected to it. Each hidden-layer neuron adds these error counts to derive an estimate of its own error. This estimate is used to calculate the adjustments to be made on the weights of the connections coming to it from the input-layer neurons. Finally, the weights are all adjusted, and then the process is repeated for the next set of input values from the training data.

This **back propagation algorithm**, so named for the error estimates that are passed back from the output layer, eventually causes the network to settle into a stable state where it can correctly respond, to any desired degree of accuracy, to all inputs in the training set. In effect, the successive changes in weights have reinforced good behavior and discouraged bad behavior (much as we train our pets) until the paths for good behavior are imprinted on the connections (as in Fido's brain). The network has "learned" what its connection weights should be, and its ability to recognize the training data is embedded

Can You Hear Me Now?

In 1999, two biomedical engineers at the University of Southern California, Theodore Berger and Jim-Shih Liaw, created the Berger-Liaw Neural Network Speaker-Independent Speech Recognition System. This is a neural network that, after minimal training, can recognize words spoken by any individual. Furthermore, it can pick out individual words from a noisy background far better than a human listener can. Even more remarkable, this neural net uses only 11 neurons connected by 30 links. The key to the success of this neural net is that the researchers incorporated a temporal dimension; neurons do not all fire for the same time duration, nor in lockstep. A demonstration of this speech recognition system can be found at *http://www.usc.edu/ext-relations/news_service/real/real_video.html*.

somehow in the collective values of these weights. At the end of its training, the neural network is ready to go to work on new recognition problems that are similar to, but not the same as, the training data and for which the answers are unknown.

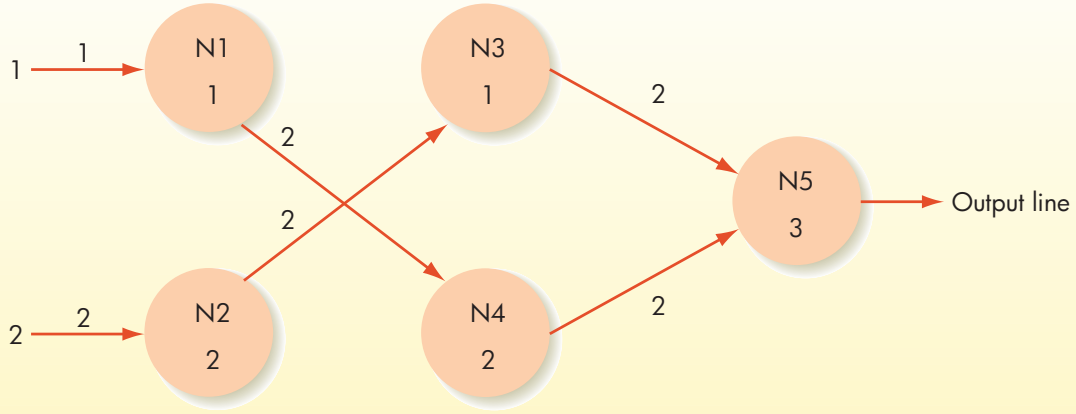
Neural networks have found their way into dozens of real-world applications. A few of these are handwriting recognition, speech recognition, recognizing patterns indicative of credit card fraud, recognizing bad credit risks for loans, predicting the odds of susceptibility to cancer, limited visual recognition systems, segmenting magnetic resonance images in medicine, adapting mirror shapes for astronomical observations, and discovering the best routing algorithm in a large communications network (a problem we mentioned in Chapter 7). With the ever-lower cost of massively parallel networks, it appears that neural networks will continue to find new applications.

LABORATORY EXPERIENCE 20

In this lab experience, you will train a neural network simulator for a character recognition task. You can choose one or more characters to present to a neural network's input layer and then train the network to correctly recognize these characters. After the network is trained, you can input garbled variations of these characters that are similar to, but not identical to, the training data characters and test the network's ability to correctly identify them. You can vary the level of "garbling" in the input to see at what point the network fails in its recognition task.

PRACTICE PROBLEM

If input line 1 is stimulated in the following neural network (and line 2 is not stimulated), will the output line fire? Explain.



15.5 Reasoning Tasks

We noted that one of the characteristics of human reasoning seems to be the ability to draw on a large body of facts and past experience to come to a conclusion. In this section we look at several ways in which artificial intelligence specialists try to get computers to emulate this characteristic.

15.5.1 *Intelligent Searching*

Earlier in this book, we presented two algorithms for searching—sequential search and binary search. These search algorithms look for a perfect match between a specific target value and an item in a list. The amount of work involved is $\Theta(n)$ for sequential search and $\Theta(\lg n)$ for binary search.

A **decision tree** for a search algorithm illustrates the possible next choices of items to search if the current item is not the target. In a sequential search, there is only one item to try next: the next item in the list. The decision tree for sequential search is linear, as shown in Figure 15.10. A decision tree for a binary search, such as the one shown in Figure 15.11, reflects the fact that if the current item is not the target, there are only two next choices: the midpoint of the sublist before this node or the midpoint of the sublist after this node. Furthermore, the binary search algorithm specifies which of the two nodes to try next.

The classical search problem benefits from two simplifications:

1. The search domain (the set of items being searched) is a linear list. At each point in the search, if the target is not found, the choice of where to look next is highly constrained.
2. We seek a perfect match, so the comparison of the target against the list item results in a binary decision—either they match or they do not.

FIGURE 15.10

Decision Tree for Sequential Search

Suppose, however, that condition 1 does not hold; the search domain is such that after any one node has been searched, there are a huge number of next choices to try, and there is no algorithm to dictate the next choice. Figure 15.12 attempts to portray this scenario. In the terminology of artificial intelligence, such a figure is called a **state-space graph**, and we seek to perform a **state-space search** to find a **solution path** through the graph. The idea is that each node of the graph represents a "state" of our problem, and we have some "goal state" or states in mind. For example, in a game of tic-tac-toe, our initial state is the empty game grid, and our goal state is a winning configuration. A solution path takes us from the initial state to a winning configuration, and the graph nodes along the way represent the intermediate configurations. In addition to finding a winning sequence of moves for a board game (tic-tac-toe, checkers, chess, and so forth), many other types of problems, such as finding the shortest path through a network or finding the most successful investment strategy in the stock market, fall into the state-space search category. In some of these problems, condition 2 of the classical search problem—that of seeking an exact match with a specified target value—is not present either. We simply want to acquire as many characteristics of the desired goal as possible, and we need some measure of when we are "close enough."

A brute force approach for a solution path traces all branches of the state-space graph so that all possible choices are tested and no test cases are repeated. This becomes a massive bookkeeping task because the number of branches grows exponentially. Given that time and computing resources are limited, an intelligent search needs to be employed. An intelligent search narrows the number of branches that must be tried and thereby puts a cap on the otherwise exponential growth of the problem. Intelligent

FIGURE 15.11

Decision Tree for Binary Search

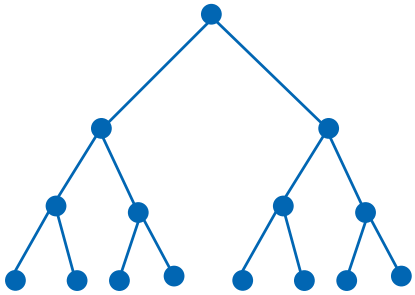
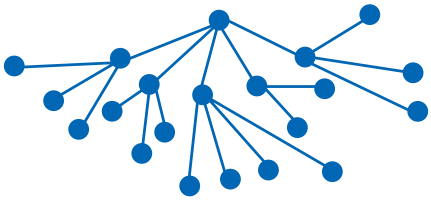


FIGURE 15.12

A State-Space Graph with Exponential Growth



searching involves applying some **heuristic** (which means, roughly, an "educated guess") to evaluate the differences between the present state and the goal state and to move us to a new state that minimizes those differences—namely, the state that maximizes our progress toward the goal state.

An intelligent chess-playing strategy, for example, is one that makes an appropriate first move and that, at each step, makes a move more likely than others to lead to a winning board configuration. Even a grand master of chess cannot pursue the brute force approach of mentally trying out all the possible next moves, all the possible moves following from each of those moves, and so on, for very many steps. (In Section 1.2 we showed that using a brute force approach, a computer would require a billion billion billion years to make its first move!) Intelligent searching is required. There must be a deep storehouse of experience that can be "consulted" on the basis of the present configuration of the board. A grandmaster–level player may need a mental database of around 50,000 of these board configurations, each with its associated information about the best next move.

Building a machine that can beat a human at chess was long thought to be a supreme test of artificial intelligence—machines that "think." Successfully playing chess, it was believed, surely epitomized logical reasoning, true "intelligence." Chess is difficult for humans. Yet, the rules for chess are straightforward; it is simply the size of the state-space that is overwhelming. As artificial intelligence researchers delved deeper into supposedly "simpler" problems such as visual recognition—things we humans do easily—it became clear that these were the harder challenges for machines. Playing chess came to be viewed as the last of the "easy" hard problems. Nonetheless, the contest of human being versus machine at the chessboard is irresistibly fascinating.

15.5.2 *Swarm Intelligence*

Recall that the connectionist architecture—neural networks—draws its inspiration from nature, namely, the human brain. Another approach to achieving a desired end, swarm intelligence, also draws its inspiration from nature, modeling the behavior of, for example, a colony of ants. Each ant is an unsophisticated creature with limited capabilities, yet acting as a collective, an ant colony can accomplish remarkable tasks. Ants can find the shortest route from a nest to a food source, carry large items, emigrate as a colony from one location to another, and form bridges. An ant "communicates" with other ants by laying down a scent trail, called a pheromone trail; other ants follow this trail and reinforce its strength by laying down their own pheromones. Given a choice, ants have a higher probability of following the strongest pheromone trail. Hence, the ant that took the shortest

The Games People Play

In May 1997, international attention was focused on a historic chess match between world champion Garry Kasparov and the IBM chess-playing computer known as Deep Blue. (IBM has since built machines hundreds of times more powerful than Deep Blue, such as the 2004 Blue Gene/L, developed for biomolecular research. These machine names are a clever play on "Big Blue," IBM's nickname in the corporate world.) Kasparov and Deep Blue played neck and neck. Both relied on the respective strengths of their "species," Kasparov utilizing recognition and reasoning, and Deep Blue churning out its high-speed computations (see Figure 15.2). In the final game, Kasparov lost the match by falling for a well-known trap. Kasparov's error, which was considered a major blunder for a player of his ability, probably reflected his weariness and the emotional strain of competing against an unexpectedly strong, utterly impassive foe. These human frailties, of course, were not shared by Deep Blue.

Gary Kasparov vs Deep Blue © Laurence Kesterson/CORBIS Sygma

Kasparov	Deep Blue
Could evaluate up to 3 chess positions per second, or 540 in the 3 minutes allowed between moves	Could evaluate up to 200,000,000 chess positions per second, or 50 billion in 3 minutes
Selected which few positions to evaluate on the basis of recognition of successful strategies or tactical approaches	Evaluated a large number of random positions to determine the optimal move
Used his brain, including experience and intuition	Used its 512 communicating processors, which act algorithmically following their C programming
Could assess his opponent's weaknesses and dynamically adjust his playing strategy	Could be modified by its development team between games to change its approach
Was subject to human emotions and weaknesses	Did not tire or have emotional responses to alter its play

The Chinook project, involving a group of computer scientists at the University of Alberta in Canada, headed by Professor Jonathan Schaeffer, began in 1989. This project was to develop a checkers-playing computer program. In 1992, an early version of Chinook competed in the World Checkers Championship against Dr. Marion Tinsley, the world champion who had won every tournament he had played in since 1950. The program lost this match but was the winner in a 1994 rematch. All this occurred before the Kasparov-Deep Blue contest, and the Guinness Book of World Records in 1996 accorded Chinook the honor of being the first computer program to win a human world championship. In April of 2007, it was announced that Chinook was finally perfected. From the standard starting positions used in tournament play, Chinook can never lose; the best a skilled opponent can achieve is a draw. Multiple computers—as many as 200 at a time—worked simultaneously to carry out the 10^{14} (100 trillion) computations needed to determine how Chinook should make its moves so that it never loses. You can play against a "reduced strength" version of Chinook at *http://www.cs.ualberta.ca/~chinook/play*.

path to food and returned to tell about it lays down a trail that other ants follow and reinforce faster than the trail laid down by an ant that took a longer path. Because pheromone trails evaporate quickly, the collective intelligence of the colony is constantly updated to respond to current conditions of its environment.

The **swarm intelligence model** captures this behavior by using simple agents (analogous to the ants) that operate independently, can sense certain aspects of their environment, and can change their environment. Research is underway to use such simple agents in telecommunications networks to avoid the complexity of a centralized control system to compute and distribute routing tables within a network. Researchers are also applying swarm intelligence to vehicle routing, job scheduling, and sensing of biological or chemical contaminants. The "ants," if you will, may even "genetically evolve" and acquire additional capabilities over time.

15.5.3 *Intelligent Agents*

Swarm intelligence rests in the colony as a whole, which seems to acquire "knowledge" that is greater than the sum of its parts. At the opposite end of the spectrum are **intelligent agents**. An intelligent agent is a form of software technology that is designed to interact collaboratively with a user somewhat in the mode of a personal assistant. Imagine that you have hired your

ANTS in Space!

NASA's Goddard Space Flight Center is developing one of the most interesting applications of swarm intelligence. The ANTS (Autonomous NanoTechnology Swarm) project involves objects shaped like three-sided pyramids with a base (tetrahedrons). Called TETwalkers, these objects have a small motor at each vertex with connecting struts between motors. The motors can retract or extend the struts, thereby changing the center of gravity of the TETwalker and causing it to tumble over. Such tumblings, repeated over and over, are the method of locomotion for the TETwalker. Future versions of the TETwalker will be much smaller, and capable of being joined together in swarms.

NASA envisions using artificial intelligence to allow these swarms to make decisions and change shape as circumstances arise. For example, a swarm could configure itself as a snake-like body that slithers across the surface of a distant planet. When it finds something interesting to report, it can reconfigure itself to grow an antenna and transmit data back to earth. Reconfiguration can also make the swarm resistant to damage—if a few TETwalkers are "injured," the swarm can just reconfigure itself to work around them.

For further information on TETwalkers, see *http://www.nasa.gov/vision/universe/roboticexplorers/ants.html*. And you can download a (faintly creepy!) video of a Tetwalker in motion at *http://ants.gsfc.nasa.gov/features/steppin.MPG*.

Photo Credit: NASA

own (human) personal assistant. In the beginning, you must tell your assistant what to do and how you want it done. Over time, however, your assistant comes to know more about you and soon can anticipate what tasks need to be done and how to perform them, what items to bring to your attention, and so forth. Your assistant becomes more valuable as he or she becomes more self-directed, always acting with your best interests in mind. You, in turn, put more and more trust in your assistant.

Like the human personal assistant, an intelligent agent begins to not merely wait for user commands but initiates communication, takes action, and performs tasks on its own on the basis of its increasing knowledge of your needs and preferences. Here are some examples that exist today:

- A personalized Web search engine that allows you to profile items of interest to you and then automatically delivers appropriate information from the Web. For example, you may request updated weather conditions for your geographic area, along with news items related to sports and European trade. At periodic time intervals, this **push technology** downloads your updated, personalized information to your screen to be displayed whenever no other task is active.
- A more intelligent version of this personalized Web searcher that enables you to "vote" on each article it sends you and then dynamically adjusts what it sends in the future as it learns about your preferences.
- An even more intelligent search agent that not only narrows down choices from topics you have chosen but can suggest new, related topics for you to explore. This is accomplished by having your agent communicate with similar agents on the Web, even when you are not online. If your agent knows of your interest in French cuisine, for example, it communicates with other agents to find those that represent users with the same interest. It may learn from these agents that many of their users are also interested in Cajun cooking. Your agent then judges whether these suggestions are coming from agents whose recommendations on the whole have been well received by you in the past. If so, it asks whether you also want information about Cajun cooking. If you do not agree to this proposal, your agent notes what agents made that suggestion and, on the next pass, gives less consideration to their ideas. The more agents that participate, the more accurate each one becomes at "understanding" the interests of its user.
- An online catalog sales company that uses an agent to monitor incoming orders and make suggestions. For example, a customer who orders a camera may be presented with a list of related accessories for sale, such as tripods and lens filters.
- A manufacturing plant that uses an intelligent agent to negotiate with suppliers on the price and scheduling of parts delivery to maximize efficiency of production.

Intelligent agent technology has been an area of interest in artificial intelligence for many years. However, intelligent agents need to display significantly greater learning capabilities and "common sense" before most users will trust them to make autonomous decisions regarding the allocation of time and money. Until then, they will be relegated to presenting suggestions to

The Demise of Clippit

From 1997 to 2003, Microsoft Office software featured the Office Assistant, a form of intelligent agent that featured an animated paperclip known as Clippit. Clippit tried to assist the user based on its analysis of the task the user was engaged in. For example, if the user typed an address and then "Dear," Clippit might pop up and say, "*It looks like you're writing a letter. Would you like help?*"

Clippit was widely viewed as annoying and intrusive rather than helpful, and it disappeared from later Office versions. The field of HCI (human-computer interaction) studies, in part, design principles to provide better bridges between the task the user wants to do and the computer's understanding of and support of that task. The well-intentioned Clippit clearly missed the boat!

their human users. However, when a sufficient level of trust in intelligent agent technology has been achieved, and when human users are willing to allow their software to make independent decisions, we will begin to see such exciting new applications as

- *Financial agents* that negotiate with one another over the Web for the sale and purchase of goods and services, using price/cost parameters set by the sellers and buyers.
- *Travel and tourism agents* (electronic, not human) that book airline flights, rent automobiles, and make hotel reservations for you on the basis of your destination, schedule, price range, and preferences.
- *Office manager agents* that screen incoming telephone calls and e-mail, putting meetings on their users' schedules, and drafting replies.

15.5.4 *Expert Systems*

Although intelligent agents incorporate a body of knowledge to "filter" their choices and thereby appear to capture certain aspects of human reasoning, they still perform relatively limited tasks. Consider the more unstructured scenario of managing the triage center in a busy hospital emergency room. The person in charge draws on (1) past experience and training to recognize various medical conditions (which may involve many recognition subtasks), (2) understanding of those conditions and their probable consequences, and (3) knowledge about the hospital's capabilities and resources in general and at the moment. From this knowledge base, a chain of reasoning is followed that leads, for example, to a decision to treat patient A immediately in a particular fashion and to let patient B wait. We consider this to be evidence of quite general "logical reasoning" in humans.

Artificial intelligence simulates this kind of reasoning through the use of **rule-based systems**, which are also called **expert systems** or **knowledge-based systems**. (The latter term is a bit confusing, because all "intelligent activity" rests on some base of knowledge.) A rule-based system attempts to mimic the human ability to engage pertinent facts and string them together in

a logical fashion to reach some conclusion. A rule-based system must therefore contain these two components:

- A knowledge base: a set of facts about the subject matter
- An inference engine: a mechanism for selecting the relevant facts and for reasoning from them in a logical way

Note that the knowledge base contains facts about a *specific* subject domain to narrow the scope to a manageable size.

The facts in the knowledge base consist of certain simple assertions. For example, let's say that the domain of inquiry is U.S. presidents. Three simple assertions are

1. Lincoln was president during the Civil War.
2. Kennedy was president before Nixon.
3. FDR was president before Kennedy.

Another type of fact is a *rule*, a statement of the form *if . . . then . . .* , which says that whenever the clause following "if" is true, so is the clause following "then." For example, here are two rules that, taken together, define what it means for one president to precede another in office. In these rules, *X, Y*, and *Z* are variables.

I. If *X* was president before *Y*, then *X* precedes *Y*.

II. If *X* was president before *Z* and *Z* precedes *Y*, then *X* precedes *Y*.

Here we are using a formal language to represent the knowledge base.

What conclusions can be reached from this collection of three assertions and two rules? Assertion 2 says that Kennedy was president before Nixon. This matches the "if" clause of rule I, where *X* is Kennedy and *Y* is Nixon. From this, the "then" clause of rule I yields a new assertion, that Kennedy precedes Nixon, which we'll call assertion 4. Now assertion 3 says that FDR was president before Kennedy, and assertion 4 says that Kennedy precedes Nixon. This matches the "if" clause of rule II, where *X* is FDR, *Z* is Kennedy, and *Y* is Nixon. From this, the "then" clause of rule II yields a new assertion, that FDR precedes Nixon, which we'll call assertion 5. Hence,

4. Kennedy precedes Nixon.
5. FDR precedes Nixon.

are two new conclusions or assertions. These assertions were previously unknown and were obtained from what was known through a process of logical reasoning. The knowledge base has been extended. We could also say that the system has *learned* two new pieces of knowledge.

If this example sounds familiar, it is because it is part of the example we used in Chapter 10 to illustrate the logic programming language Prolog. Prolog provides one means of implementing an inference engine for a rule-based system.

The inference engine is basically using the following pattern of reasoning:

Given that the rule

If A then B

and the fact

A

are both in the knowledge base, then the fact

B

can be inferred or concluded.

This reasoning process, as we noted in Chapter 10, goes by the Latin name of **modus ponens**, which means "method of assertion." It gives us a method for making new assertions. We humans use this deductive reasoning process all the time. However, it is also suitable for computerization because it is basically a matching algorithm that can be implemented by brute force trial and error. Systems like Prolog, however, apply some additional guidelines in their search for matches to speed up the process; that is, they employ a form of intelligent searching.

Inference engines for rule-based systems can proceed in several ways. **Forward chaining** begins with assertions and tries to match those assertions to the "if" clauses of rules, thereby generating new assertions. These may in turn be matched with "if" clauses, generating still more assertions. This is the process we used in our example. **Backward chaining** begins with a proposed conclusion and tries to match it with the "then" clauses of rules. It then looks at the corresponding "if" clauses and tries to match those with assertions, or with the "then" clauses of other rules. This process continues until all "if" clauses that arise have been successfully matched with assertions, in which case the proposed conclusion is justified, or until no match is possible, in which case the proposed conclusion is rejected. Backward chaining in our example says, Here's a hypothesis: FDR precedes Nixon, and the system works backwards to justify this hypothesis.

In addition to the knowledge base and the inference engine, most rule-based systems also have an **explanation facility**. This allows the user to see the assertions and rules used in arriving at a conclusion, as a sort of check on the path of reasoning or for the user's own enlightenment.

Of course, a rule-based system about some particular domain is only as good as the assertions and rules that make up the knowledge base. The builder of such a system acquires the information for the knowledge base by consulting "experts" in the domain and mining their expertise. This process, called **knowledge engineering**, requires a great deal of interaction with the human expert, much of it in the domain environment. If the domain expert is the manager of a chemical processing plant, for example, a decision to "turn down valve A whenever the temperature in pipe P exceeds 235°F and valves B and C are both closed" may be such an ingrained behavior that the expert won't remember it as part of a question and answer session on "what you do on your job." It only emerges by on-site observation. For the hospital example, one might need to follow people around in the emergency room, observe their decisions, and later question them on why those decisions were made. It is also possible to incorporate probabilities to model the thinking process, for example, "If the patient has fever and stomach pains, the probability of appendicitis is 73% and the probability of gall bladder problems is 27%, therefore I first check A and then B."

Rule-based systems have been implemented in many domains, including specific forms of medical diagnosis, computer chip design, monitoring of manufacturing processes, financial planning, purchasing decisions for retail stores, automotive troubleshooting, and diagnosis of failures in electronic systems. They will no doubt be even more commonplace in the future.

PRACTICE PROBLEMS

1. Given the assertion "Frank is bald" and the rule "If *X* is bald, then *X* is tall," what conclusion can be inferred? If Frank were known to be tall, would that necessarily imply that he was bald?
2. Given the assertion "Frank is not bald" and the rule "If *X* is bald, then *X* is tall," what conclusion can be inferred?

15.6 Robotics

The term "robot" implies a device, often human-like in form, that has the ability to gather sensory information from its surroundings and to autonomously (i.e., without direct human instructions) perform mechanical actions of some sort in response. The term "robot" was used in a play written in 1921 by Czech author Karel Capek. The play was entitled *R.U.R*, short for *Rossum's Universal Robots*. In the play, a scientist invents robots that perform simple repetitive tasks to help people but who take over the world when their human owners try to use them to fight wars. The word "robot" comes from the Czech word "robota," meaning slave-like labor. Robots have been part of science fiction ever since—think C3-PO and R2-D2 in the *Star Wars* movies, or WALL-E (**W**aste **A**llocation **L**oad **L**ifter **E**arth-class) in the 2008 Disney-Pixar animated movie of the same name.

Fact has not yet caught up with science fiction, but today there are a surprising number of applications of robots. Here's a partial list of the uses to which robots are put in manufacturing, science, the military, medicine, and in the consumer marketplace.

- Assembling automobile parts
- Packaging food and drugs
- Placing and soldering wires in circuits

Finding Water on Mars

NASA's Phoenix Mission to Mars in the summer of 2008 obtained conclusive proof that frozen water is contained in the Martian soil. The robotic arm of the Phoenix Mars Lander, shown here with part of the Lander's solar panel, scooped up a small frozen soil sample and delivered it to a heater that analyzed the resultant vapors and sent data back to earth, confirming that the sample contained water ice. The presence of water is considered a key factor for the possibility of life. Scientists will study whether this frozen water ever is or was available on Mars in liquid form.

NASA/JPL-Caltech/University of Arizona/Texas A&M University

- Bomb disposal
- Welding
- Radiation and chemical spill detection
- Inspection of sewer lines and oil pipes
- Exploration (the Mars Rovers)
- Underwater salvage
- Microsurgery
- Wheelchair navigation
- Emergency search and rescue
- Vacuum cleaning
- Window washing
- Lawn mowing
- Home or commercial security sentry duty

Robots currently perform tasks that are too repetitive or dangerous for humans. Robots, of course, do not get bored, they can perform the same tasks with the same precision every time, and they can work in hostile environments.

Groups of robotic devices can act together to investigate wide-ranging geographic areas that, again, would be difficult for human workers. Scientists at the University of Washington in Seattle have built robotic "fish." Each fish is about 20 inches long, with the ability to flap its tail and two fins; using this propulsion and steering mechanism, the fish can swim in any direction and make tight turns. Communicating with each other using radio signals, the fish can swim together as a coordinated group or move independently. Such technology will eventually allow tracking of marine wildlife or assessment of environmental damage in the earth's oceans.

Researchers at the Harvard Microrobotics Laboratory are perfecting a robotic "fly." Robotic flying insects are not just "small airplanes"—the flight dynamics of insects are much more complex because of an insect's ability to hover, fly in any direction, land on walls, etc. Once perfected, swarms of robotic insects could be released in a disaster area to search for survivors. Because of their size, they could penetrate hard-to-access areas of debris and transmit signals to (human or robotic) rescue workers.

Collecting data on the world's ice shelves, to study climate change, is the goal of researchers at the Georgia Institute of Technology and Pennsylvania State University. They have created small robotic "snowmobiles" called SnoMotes that will be able to travel autonomously over assigned terrain, take measurements, and report their findings. After an initial failure at building a rugged robot from scratch, the team turned to a toy snowmobile designed to operate in snow conditions and to withstand abuse by children. In addition to a camera, sensors, and computing equipment, the robot needs to be fitted with a heater to protect the electronic equipment in below-zero environments. Like the robotic fish, the SnoMotes will communicate with each other to decide which robot samples which location for the most efficient overall coverage.

More and more, however, robots are being developed to interact with humans in a less "robotic" and more "human-like" way to perform tasks for the disabled, to watch over small children, and to entertain and provide companionship. Japan has an interest in developing "humanoid" robots to help care for its aging population.

One of the more well-known humanoid robots is ASIMO (**A**dvanced **S**tep in **I**nnovative **M**obility), built by Honda Motor Company, a Japanese corporation. As the name suggests, much of the focus of the design of this robot over earlier models was refinement of the robot's motion capabilities, extending the

range of its arm movement and improving the smoothness and stability of its two-legged walking, including walking on uneven ground and navigating stairs. This jointed robot is designed to measure the forces acting on it at each step. If these forces get out of balance, threatening a fall, adjustments are made in the placement of the foot and the position of the torso to regain balance. One only has to watch a toddler learning to walk to see the challenges this represents. ASIMO's capabilities continue to be developed. ASIMO can open and close doors while passing through a doorway, guide office guests to a meeting room and serve refreshments on a tray, recognize when it needs recharging, and walk to the nearest recharging station and hook up to be recharged. In May 2008, ASIMO conducted the Detroit Symphony Orchestra in a concert for young people! Go to *http://world.honda.com/ASIMO/* for more detailed information about ASIMO.

Research is ongoing in the field of robotics. Robotics involves the aspects of artificial intelligence we discussed earlier, namely recognition tasks and reasoning tasks. Through some sort of elementary vision, auditory, or tactile system, the robot must not only gather sensory information, but filter out the possibly vast amount of data its surroundings might present to it to "recognize," that is, make sense of, the important features. Then the robot must make decisions—reason about—the information it has recognized to be able to take some action. There is also the additional challenge of the mechanics and electronics needed to make the robot respond physically.

Two strategies characterize robotics research. The **deliberative strategy** says that the robot must have an internal representation of its environment and that its actions in response to some stimuli are programmed into the robot based on this model of the environment. This strategy seems to reflect what we as humans think of as high-level cognitive reasoning—we have a mental model of our environment, we reflect on a stimulus from that environment, and make a reasoned decision about the next course of action. (This is a generalization of the expert system idea discussed earlier.) The **reactive strategy** uses heuristic algorithms to allow the robot to respond directly to stimuli from its environment without filtering through some line of reasoning based on its internal understanding of that environment. This stimulus-response approach seems to reflect human subconscious behavior—holding out our hands to protect ourselves during a fall, for example, or drawing back from a hot flame. Proponents of the deliberative strategy argue that a robot cannot react meaningfully without processing the stimulus and planning a reaction based on its internal representation of the environment. Proponents of the reactive strategy say that such a requirement is too restrictive and does not allow the robot to respond freely to any or all new stimuli it might encounter. Note that we as humans use both our conscious thought processes and our subconscious reactions in our everyday life, so a combination of these strategies may be the most successful approach.

15.7 Conclusion

In this chapter we have touched on three basic elements of artificial intelligence: knowledge representation, recognition problems, and reasoning problems. We've discussed common approaches to building artificial intelligence systems: symbolic manipulation (expert systems), connectionist architectures (neural networks), and genetic or evolutionary approaches (swarm intelligence). We have also outlined some of the strategies and challenges in robotics design.

RoboCup

RoboCup is an annual international event that includes competitions in various categories between soccer-playing robots. The 12th annual RoboCup competition was held in Suzhou, China, in July 2008. The Dutch Robotics team participated in the TeenSize Humanoid League. The robotics initiative in the Netherlands is associated with the 3TU.Federation representing the three technical universities: Delft, Eindhoven, and Twente. Research groups from the three universities contribute different fields of expertise to the development of more intelligent humanoid robots, including TUlip, the Dutch entry in RoboCup 2008. TUlip is rendered here in a protective suit of orange, the colors of the (human) Dutch National soccer team. TUlip is 1.2 meters tall and has a walking speed of 0.5 m/sec. Go to *http://site.dutchrobocup.com* for more information on the Dutch Robotics initiative

The RoboCup event is not all fun and games; it is intended to promote research in artificial intelligence through robotics. It includes a symposium to discuss robotics research related to the technological challenges of soccer-playing robots, such as vision, recognition, and locomotion. Solutions being found for the soccer field are the same ones that will lead to new autonomous robots more capable of carrying out useful tasks.

Credit: Used with permission from 3TU.Federation.

We have mentioned only a few of the many application areas of AI. Others include speech recognition, natural language translation, image analysis, target recognition, and game playing. Today we can use speech recognition systems to control our telephones, appliances, and computers by talking to them, use a tablet PC with handwriting recognition software, and ask a Web page to translate text from one language to another.