

Representing vagueness: Fuzzy sets and fuzzy logic

FUZZY SET THEORY



Fuzzy set theory is a means of specifying how well an object satisfies a vague description. For example, consider the proposition "Nate is tall." Is this true, given that Nate is 5' 10"? Most people would hesitate to answer "true" or "false," preferring to say, "sort of." Note that this is not a question of uncertainty about the external world — we are sure of Nate's height. Rather it is a case of vagueness or uncertainty about the meaning of the linguistic term "tall." So *most authors say that fuzzy set theory is not a method for uncertain reasoning at all.*

Another way to think of this is as similarity to a prototype — how close is Nate to our prototype for tall person? We could express this in logical notation by making *TallPerson* a constant and having a function *Similarity* that compares things to it:

$$\text{Similarity}(\text{Nate}, \text{TallPerson}) = \text{SortOf}$$

Fuzzy set theory takes a slightly different approach: it treats *TallPerson* as a fuzzy predicate and says that the truth value of *TallPerson(Nate)* is a number between 0 and 1, rather than being just *True* or *False*. The name "fuzzy set" derives from the interpretation of the predicate as implicitly defining a set of its members, a set that does not have sharp boundaries. Consider a kennel with 60 rottweilers and 40 retrievers. If we pick a dog at random, the probability of it being a rottweiler is 0.6; this is uncertainty. The uncertainty can be resolved by looking at the dog. But if a rottweiler and a retriever were to mate, then the puppies would be half rottweiler, half retriever. There is no uncertainty here, no additional evidence we can gather to determine the breed of the puppies. Rather, this is a case of fuzziness at the boundary between two breeds.

FUZZY LOGIC

Fuzzy logic takes a complex sentence such as *TallPerson(Nate) V Smart(Nate)* and determines its truth value as a function of the truth values of its components. The rules for evaluating the fuzzy truth, *T*, of a complex sentence are

$$T(A \wedge B) = \min(T(A), T(B))$$

$$T(A \vee B) = \max(T(A), T(B))$$

$$T(\neg A) = 1 - T(A)$$

Fuzzy logic is therefore a truth-functional system, and is thus subject to all the usual problems. It has the additional problem that it is inconsistent with normal propositional or first-order logic. We would like any logic to ensure that standard equivalences such as $A \vee \neg A \Leftrightarrow \text{True}$ hold, but in fuzzy logic, $T(A \vee \neg A) \neq T(\text{True})$.

Despite these serious semantic difficulties, fuzzy logic has been very successful in commercial applications, particularly in control systems for products such as automatic transmissions, trains, video cameras, and electric shavers. Elkan (1993) argues that these applications are successful because they have small rule bases, have no chaining of inferences, and have tunable parameters that can be adjusted to improve the system's performance (often by learning techniques). The fact that they are implemented with fuzzy operators is incidental to their success. Elkan predicts that when more complex applications are tackled with fuzzy logic, they will run into the same kinds of problems that plagued knowledge-based systems using certainty factors and other approaches. As one might expect, the debate continues.

15.7 SUMMARY

Both Chapter 9 and this chapter are concerned with reasoning properly, but there is a difference in just what that means. In first-order logic, proper reasoning means that conclusions follow from premises—that if the initial knowledge base faithfully represents the world, then the inferences also faithfully represent the world. In probability, we are dealing with beliefs, not with the state of the world, so "proper reasoning" means having beliefs that allow an agent to act rationally.

In Chapter 10, we saw a variety of approaches to the problem of implementing logical reasoning systems. In this chapter, we see much more of a consensus: efficient reasoning with probability is so new that there is one main approach—belief networks—with a few minor variations. The main points are as follows:

- **Conditional independence** information is a vital and robust way to structure information about an uncertain domain.
- **Belief networks** are a natural way to represent conditional independence information. The links between nodes represent the qualitative aspects of the domain, and the conditional probability tables represent the quantitative aspects.
- A belief network is a complete representation for the joint probability distribution for the domain, but is often exponentially smaller in size.
- Inference in belief networks means computing the probability distribution of a set of query variables, given a set of evidence variables.
- Belief networks can reason causally, diagnostically, in mixed mode, or intercausally. No other uncertain reasoning mechanism can handle all these modes.
- The complexity of belief network inference depends on the network structure. In **polytrees** (singly connected networks), the computation time is linear in the size of the network.
- There are various inference techniques for general belief networks, all of which have exponential complexity in the worst case. In real domains, the local structure tends to make things more feasible, but care is needed to construct a tractable network with more than a hundred nodes.
- It is also possible to use approximation techniques, including **stochastic simulation**, to get an estimate of the true probabilities with less computation.
- Various alternative systems for reasoning with uncertainty have been suggested. All the **truth-functional** systems have serious problems with mixed or intercausal reasoning.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The use of networks to represent probabilistic information began early in the twentieth century, with the work of Sewall Wright on the probabilistic analysis of genetic inheritance and animal growth factors (Wright, 1921; Wright, 1934). One of his networks appears on the cover of this

book. I. J. Good (1961) investigated the use of Bayesian inference in belief networks.⁵ The **influence diagram or decision network** representation for decision problems, which incorporated a DAG representation for random variables, was used in decision analysis in the late 1970s (see Chapter 16), but no interesting evaluation algorithms were developed for decision networks until 1986 (Shachter, 1986). Judea Pearl (1982a) developed the message-passing method for carrying out inference in networks that had the form of trees. Jin Kim (Kim and Pearl, 1983) extended the method to all singly connected networks. The backward-chaining algorithm given in the text was derived by the authors, but has much in common with the methods of Pearl (1988) and of Shachter et al. (1990). Gregory Cooper (1990) showed that the general problem of inference in unconstrained belief networks is NP-hard, and Paul Dagum and Mike Luby (1993) showed the corresponding approximation problem to be NP-hard.

David Spiegelhalter and Steffen Lauritzen pioneered the use of clustering for inference in multiply connected networks (Spiegelhalter, 1986; Lauritzen and Spiegelhalter, 1988). Finn Jensen and colleagues (1990) developed an object-oriented computational scheme for clustering. The scheme is implemented in the HUGIN system, an efficient and widely used tool for uncertain reasoning (Andersen et al., 1989). Eric Horvitz, H. Jacques Suermondt, and Gregory Cooper (1989) developed bounded cutset conditioning, building on earlier work on cutsets (Pearl, 1986). The logic sampling method is due to Max Henrion (1988), and likelihood weighting was developed by Fung and Chang (1989) and Shachter and Peot (1989). A large-scale application of likelihood weighting to medical diagnosis appears in Shwe and Cooper (1991). The latter paper also incorporates elements of the "Markov blanket" simulation scheme developed by Pearl (1987). The first expert system using belief networks was CONVINCE (Kim, 1983; Kim and Pearl, 1987). More recent systems include the MUNIN system for diagnosing neuromuscular disorders (Andersen et al., 1989) and the PATHFINDER system for pathology (Heckerman, 1991).

The extension of belief networks to handle continuous random variables is an important topic of current research. The basic technical problem is that the distributions must be representable by a finite number of parameters, and must come from a family of distributions that is closed under belief net updating. To date, only Gaussian distributions have been used. Networks with continuous variables but no discrete variables have been considered (Pearl, 1988; Shachter and Kenley, 1989). Such networks can be evaluated in polynomial time, regardless of topology. The inclusion of discrete variables has been investigated by Lauritzen and Wermuth (1989), and implemented in the cHUGIN system (Olesen, 1993). Currently, exact algorithms are known only for the case in which discrete variables are ancestors, not descendants, of continuous variables. Stochastic simulation algorithms, on the other hand, can handle arbitrary distributions and topologies.

Qualitative probabilistic networks (Wellman, 1990) provide a purely qualitative abstraction of belief networks, using the notion of positive and negative influences between variables. Wellman shows that in many cases such information is sufficient for optimal decision making without the need for precise specification of probability values. Work by Adnan Darwiche and Matt Ginsberg (1992) extracts the basic properties of conditioning and evidence combination from probability theory and shows that they can also be applied in logical and default reasoning.

⁵ I. J. Good was chief statistician for Turing's codebreaking team in World II. In *2001: A Space Odyssey*, Arthur C. Clarke credited Good and Minsky with making the breakthrough that led to the development of the HAL computer.

The literature on default and nonmonotonic reasoning is very large. Early work is collected in *Readings in Nonmonotonic Reasoning* (Ginsberg, 1987). Shoham (1988) discusses the nonmonotonic nature of reasoning about time and change, and proposes a semantics based on general preferences between possible models of a default knowledge base. Geffner (1992) provides an excellent introduction to default reasoning, including early philosophical work on default conditionals. Some informative computational complexity results on default reasoning have been derived by Kautz and Selman (1991). Bain and Muggleton (1991) provide methods for learning default rules, suggesting a role for them as compact representations of data containing regularities with exceptions. **Autoepistemic logic** (Moore, 1985b) attempts to explain nonmonotonicity as arising from the agent's reflection on its own states of knowledge. A comprehensive retrospective and summary of work in this area is given by Moore (1993).

Applications of nonmonotonic reasoning have been largely limited to improving the theoretical underpinnings of logic programming. The use of negation as failure in Prolog can be viewed as source of nonmonotonicity because adding new facts will rule out some potential derivations (Clark, 1978). An important subclass of the so-called default theories in the default logic of (Reiter, 1980) can be shown to be equivalent to logic programs in a language in which both negation as failure and classical logical negation are available (Gelfond and Lifschitz, 1991). The same is true of circumscriptive theories (Gelfond and Lifschitz, 1988). Autoepistemic logic can, in turn, be closely imitated by circumscription (Lifschitz, 1989).

Certainty factors were invented for use in the medical expert system MYCIN (Shortliffe, 1976), which was intended both as an engineering solution and as a model of human judgment under uncertainty. The collection *Rule-Based Expert Systems* (Buchanan and Shortliffe, 1984) provides a complete overview of MYCIN and its descendants. David Heckerman (1986) showed that a slightly modified version of certainty factors can be analyzed as a disguised form of reasoning with standard probability theory. The PROSPECTOR expert system (Duda *et al.*, 1979) used a rule-based approach in which the rules were justified by a global independence assumption. More recently, Bryan Todd has shown that a rule-based system can perform correct Bayesian inference, provided that the structure of the rule set exactly reflects a set of conditional independence statements that is equivalent to the topology of a belief network (Todd *et al.*, 1993).

Dempster-Shafer theory originates with a paper by Arthur Dempster (1968) proposing a generalization of probability to interval values, and a combination rule for using them. Later work by Glenn Shafer (1976) led to the Dempster-Shafer theory being viewed as a competing approach to probability. Ruspini *et al.* (1992) analyze the relationship between the Dempster-Shafer theory and standard probability theory from an AI standpoint. Shenoy (1989) has proposed a method for decision making with Dempster-Shafer belief functions.

Fuzzy sets were developed by Lotfi Zadeh (1965) in response to the perceived difficulty of providing exact inputs for intelligent systems. The text by Zimmermann (1991) provides a thorough introduction to fuzzy set theory. As we mentioned in the text, fuzzy logic has often been perceived incorrectly as a direct competitor to probability theory whereas in fact it addresses a different set of issues. **Possibility theory** (Zadeh, 1978) was introduced to handle uncertainty in fuzzy systems, and has much in common with probability. Dubois and Prade (1994) provide a thorough survey of the connections between possibility theory and probability theory. An interesting interview/debate with Lotfi Zadeh (the founder of fuzzy logic) and William Kahan (the Turing award winner) appears in Woehr (1994).

The three approaches to quantitative reasoning about uncertainty just surveyed—fuzzy logic, certainty factors, and Dempster-Shafer theory—were the three main alternatives to which AI researchers resorted after early probabilistic systems fell out of favor in the early 1970s, as described in Chapter 14. The later development of nonmonotonic logics in the early 1980s was also, in part, a response to the need for an effective method for handling uncertainty. The resurgence of probability depended mainly on the discovery of belief networks as a method for representing and using conditional independence information. This resurgence did not come without a fight—probabilistic methods were attacked both by logicians, who believed that numerical approaches to AI were both unnecessary and introspectively implausible, and by supporters of the other quantitative approaches to uncertainty. Peter Cheeseman's (1985) pugnacious "In Defense of Probability," and his later article "An Inquiry into Computer Understanding" (Cheeseman, 1988, with commentaries) give something of the flavor of the debate.

It is fair to say that certainty factors are now of historical interest only (for reasons noted above), and although fuzzy logic is a healthy, ongoing enterprise, it is increasingly perceived as a way of handling continuous-valued variables rather than uncertainty. Nonmonotonic logics continue to be of interest as a purely qualitative mechanism, although there is a good deal of work aimed at showing how nonmonotonic inference is best viewed as a special case of probabilistic reasoning with probabilities close to 0 or 1 (Goldszmidt *et al.*, 1990). Dempster-Shafer theory is still perceived by its supporters as a viable alternative to Bayesian systems. Nonprobabilists continue to be irritated by the dogmatic "Bayesianity" of the probabilists.

Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference (Pearl, 1988) is a comprehensive textbook and reference on belief networks, by one of the major contributors to the field. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms* (Neapolitan, 1990) gives a relatively readable introduction to belief networks from the standpoint of their use in expert systems. *Readings in Uncertain Reasoning* (Shafer and Pearl, 1990) is a voluminous collection of important papers about probability, belief networks, and other formalisms for reasoning about uncertainty, and their applications in AI. *Uncertainty in Artificial Intelligence* (Kanal and Lemmer, 1986) is an anthology containing a number of important early papers about belief networks and other matters having to do with reasoning under uncertainty. Although this volume is not numbered, a series of numbered anthologies with the same title have also been published. New research on probabilistic reasoning appears both in mainstream AI journals such as *Artificial Intelligence* and in more specialized journals such as the *International Journal of Approximate Reasoning*. The proceedings of the Conferences on Uncertainty in Artificial Intelligence (known as UAI) are an excellent source for current research.

EXERCISES

- 15.1 Consider the problem of dealing with a car that will not start, as diagrammed in Figure 15.5.
- Extend the network with the Boolean variables *IcyWeather* and *StarterMotorWorking*.
 - Give reasonable conditional probability tables for all the nodes.

- c. How many independent values are contained in the joint probability distribution for eight Boolean nodes, assuming no conditional independence relations hold among them?
- d. How many independent probability values do your network tables contain?
- e. The conditional probability table for *Starts* is a canonical one. Describe, in English, what its structure will be in general, as more possible causes for not starting are added.

15.2 In your local nuclear power station, there is an alarm that senses when a temperature gauge exceeds a given threshold. The gauge measures the core temperature. Consider the Boolean variables *A* (alarm sounds), *FA* (alarm is faulty), and *FC* (gauge is faulty), and the multivalued nodes *G* (gauge reading) and *T* (actual core temperature).

- a. Draw a belief net for this domain, given that the gauge is more likely to fail when the core temperature gets too high.
- b. Is your network a polytree?
- c. Suppose there are just two possible actual and measured temperatures, Normal and High; and that the gauge gives the incorrect temperature $x\%$ of the time when it is working, but $y\%$ of the time when it is faulty. Give the conditional probability table associated with *G*.
- d. Suppose the alarm works unless it is faulty, in which case it never goes off. Give the conditional probability table associated with *A*.
- e. Suppose the alarm and gauge are working, and the alarm sounds. Calculate the probability that the core temperature is too high.
- f. In a given time period, the probability that the temperature exceeds threshold *isp*. The cost of shutting down the reactor is c_s ; the cost of not shutting it down when the temperature is in fact too high is c_m (*m* is for meltdown). Assuming the gauge and alarm to be working normally, calculate the maximum value for x for which the gauge is of any use (i.e., if x is any higher than this, we have to shut down the reactor all the time).
- g. Suppose we add a second temperature gauge *H*, connected so that the alarm goes off when either gauge reads High. Where do *H* and F_H (the event of *H* failing) go in the network?
- h. Are there circumstances under which adding a second gauge would mean that we would need *more* accurate (i.e., more likely to give the correct temperature) gauges? Why (not)?

15.3 Two astronomers, in different parts of the world, make measurements M_1 and M_2 of the number of stars *N* in some small region of the sky, using their telescopes. Normally, there is a small possibility of error by up to one star. Each telescope can also (with a slightly smaller probability) be badly out of focus (events F_1 and F_2), in which case the scientist will undercount by three or more stars. Consider the three networks shown in Figure 15.12.

- a. Which of these belief networks correctly (but not necessarily efficiently) represent the above information?
- b. Which is the best network?
- c. Give a reasonable conditional probability table for the values of $\mathbf{P}(M_1|N)$. (For simplicity, consider only the possible values 1, 2, and 3 in this part.)
- d. Suppose $M_1 = 1$ and $M_3 = 3$. What are the possible numbers of stars?
- e. Of these, which is the most likely number?

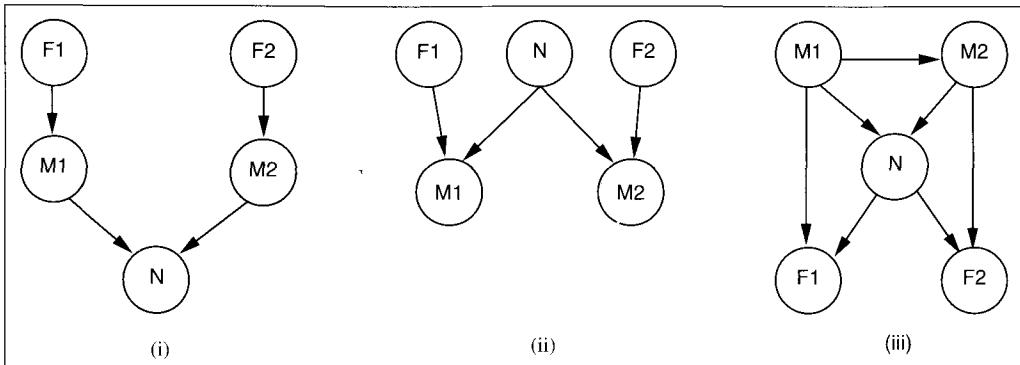


Figure 15.12 Three possible networks for the telescope problem.

$$\frac{1}{2} \times 6 = 3$$

15.4 You are an AI consultant for a credit card company, and your task is to construct a belief net that will allow the company to determine whether or not to grant a person a card.

- a. What are the evidence variables? These are the variables for which you can obtain information, on the basis of which it is legal to make decisions, and that are relevant to the decision. Thus, *Age* might be one of your variables, but *VotingRecord* and *HairColor* would not.
 - b. What is the output variable (i.e., what proposition is the company going to examine the probabilities of in order to determine whether to grant a card)? This should not be *Decision* with values *yes* and *no* because the company has control over the value of this variable.
 - c. Construct your network by incrementally adding variables in causal order, as described in the chapter. You may wish to add intermediate nodes such as *Reliability* and *FutureIncome*. (Remember that the company cares about what will happen in the future, not about the past *per se*.) Set the conditional probabilities for each node. Write commentary to describe your reasoning in choosing your variables and links. If you find that a node has too many predecessors (so that the conditional probability table becomes too big), that is a good hint that you need some intermediate variables.
 - d. Build a file of test data corresponding to your evidence variables. As far as possible these should be real people! Do some interviewing to get real data if you can; try to get a wide variety of cases, from deadbeats to trust-fund babies. Run the data through your network to see if the net's results correspond to your own judgements. Examine not only the output variable, but the value of other intermediate variables. You may need to go through several iterations of steps (c) and (d).
 - e. Write a report showing various test cases and explaining the advantages of your approach. Your report should be enough to convince the company to adopt your product.

卷之三

15.5 You are an AI consultant for an auto insurance company. Your task is construct a belief network that will allow the company to decide how much financial risk they run from various policy holders, given certain data about the policy holders. (In case you think all class projects

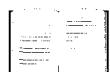
are just toys to amuse sadistic instructors, bear in mind that a 1 % improvement in risk assessment is worth well over a billion dollars a year.)

In order to design a belief network, you need *output variables* and *evidence variables*. The output variables are those for which the insurance company is trying to get a probability distribution for their possible values. The evidence variables are those variables for which you can obtain information, on the basis of which it is legal to make decisions, and that are relevant to the decision.

Output variables represent the costs of various catastrophic events that the insurance company might have to reimburse. (We do not consider the ways in which the company tries to avoid payment.) In the automobile insurance domain, the major output variables are medical cost (*MedCost*), property cost (*PropCost*), and intangible liability cost (*ILiCost*). Medical and property costs are those incurred by *all* individuals involved in an accident; auto theft or vandalism might also incur property costs. Intangible liability costs are legal penalties for things like "pain and suffering," punitive damages, and so forth, that a driver might incur in an accident in which he or she is at fault.

Evidence variables for this domain include the driver's age and record; whether or not he or she owns another car; how far he or she drives per year; the vehicle's make, model and year; whether it has safety equipment such as an airbag and antilock brakes; where it is garaged and whether it has an antitheft device.

Build a network for this problem. You will need to decide on suitable domains for the variables, bearing in mind the need to discretize (unless you plan to use a stochastic simulation algorithm). You will also need to add intermediate nodes such as *DrivingSkill* and *AutoRuggedness*. Write commentary to describe your reasoning in choosing your domains, variables, links, and inference algorithm. If you find that a node has too many predecessors (so that the conditional probability table becomes too big), that is a good hint that you need some intermediate variables. Generate a few reasonable-looking test cases to get a feeling for what your network does with them. How would you convince the insurance company to adopt your product?



15.6 Is probabilistic reasoning monotonic or nonmonotonic? Do these concepts even apply to probabilities?

16

MAKING SIMPLE DECISIONS

In which we see how an agent should make decisions so that it gets what it wants—on average, at least.

In this chapter, we return to the idea of utility theory that was introduced in Chapter 14 and show how it is combined with probability theory to yield a decision-theoretic agent—one that can make rational decisions based on what it believes and what it wants. Such an agent can make decisions in contexts where uncertainty and conflicting goals leave a logical agent with no way to decide.

Section 16.1 introduces the basic principle of decision theory: the maximization of expected utility. Section 16.2 shows that the behavior of any rational agent can be captured by supposing a utility function that is being maximized. Section 16.3 discusses the nature of utility functions in more detail, and in particular their relation to individual quantities such as money. Section 16.4 shows how to handle utility functions that depend on several quantities. In Section 16.5, we describe the implementation of decision-making systems. In particular, we introduce a formalism called **decision networks** (also known as **influence diagrams**) that extends belief networks by incorporating actions and utilities. The remainder of the chapter discusses issues that arise in applications of decision theory to expert systems.

16.1 COMBINING BELIEFS AND DESIRES UNDER UNCERTAINTY

In the *Port-Royal Logic*, written in 1662, the French philosopher Arnauld stated that

To judge what one must do to obtain a good or avoid an evil, it is necessary to consider not only the good and the evil in itself, but also the probability that it happens or does not happen; and to view geometrically the proportion that all these things have together.

These days, it is more common in scientific texts to talk of utility rather than good and evil, but the principle is exactly the same. An agent's preferences between world states are captured by a **utility function**, which assigns a single number to express the desirability of a state. Utilities are combined with the outcome probabilities for actions to give an expected utility for each action.

We will use the notation $U(S)$ to denote the utility of state S according to the agent that is making the decisions. For now, we will consider states as complete snapshots of the world, similar to the **situations** of Chapter 7. This will simplify our initial discussions, but it can become rather cumbersome to specify the utility of each possible state separately. In Section 16.4, we will see how states can be decomposed under some circumstances for the purpose of utility assignment.

An nondeterministic action A will have possible outcome states $Result_i(A)$, where i ranges over the different outcomes. Prior to execution of A , the agent assigns probability $P(Result_i(A)|Do(A), E)$ to each outcome, where E summarizes the agent's available evidence about the world, and $Do(A)$ is the proposition that action A is executed in the current state. Then we can calculate the **expected utility** of the action given the evidence, $EU(A|E)$, using the following formula:

$$EU(A|E) = \sum_i P(Result_i(A)|E, Do(A))U(Result_i(A)) \quad (16.1)$$

EXPECTED UTILITY

MAXIMUM EXPECTED UTILITY

The principle of **maximum expected utility** (MEU) says that a rational agent should choose an action that maximizes the agent's expected utility.

In a sense, the MEU principle could be seen as defining all of AI.¹ All an intelligent agent has to do is calculate the various quantities, maximize over its actions, and away it goes. But this does not mean that the AI problem is *solved* by the definition!

Although the MEU principle defines the right action to take in any decision problem, the computations involved can be prohibitive, and sometimes it is difficult even to formulate the problem completely. Knowing the initial state of the world requires perception, learning, knowledge representation, and inference. Computing $P(Result_i(A)|Do(A), E)$ requires a complete causal model of the world and, as we saw in Chapter 15, NP-complete updating of belief nets. Computing the utility of each state, $U(Result_i(A))$, often requires search or planning, because an agent does not know how good a state is until it knows where it can get to from that state. So, decision theory is not a panacea that solves the AI problem. But it does provide a framework in which we can see where all the components of an AI system fit in.

The MEU principle has a clear relation to the idea of performance measures introduced in Chapter 2. The basic idea is very simple. Consider the possible environments that could lead to an agent having a given percept history, and consider the different possible agents that we could design. *If an agent maximizes a utility function that correctly reflects the performance measure by which its behavior is being judged, then it will achieve the highest possible performance score, if we average over the possible environments in which the agent could be placed.* This is the central justification for the MEU principle itself.

In this chapter, we will only be concerned with single or **one-shot decisions**, whereas Chapter 2 defined performance measures over environment histories, which usually involve many decisions. In the next chapter, which covers the case of **sequential decisions**, we will show how these two views can be reconciled.



ONE-SHOT DECISIONS

¹ Actually, it is not quite true that AI is, even in principle, a field that attempts to build agents that maximize their expected utility. We believe, however, that understanding such agents is a good place to start. This is a difficult methodological question, to which we return in Chapter 27.

16.2 THE BASIS OF UTILITY THEORY

Intuitively, the principle of Maximum Expected Utility (MEU) seems like a reasonable way to make decisions, but it is by no means obvious that it is the *only* rational way. After all, why should maximizing the *average* utility be so special—why not try to maximize the sum of the cubes of the possible utilities, or try to minimize the worst possible loss? Also, couldn't an agent act rationally just by expressing preferences between states without giving them numeric values? Finally, why should a utility function with the required properties exist at all? Perhaps a rational agent can have a preference structure that is too complex to be captured by something as simple as a single real number for each state.

Constraints on rational preferences

These questions can be answered by writing down some constraints on the preferences that a rational agent should have, and then showing that the MEU principle can be derived from the constraints. Writing down these constraints is a way of defining the semantics of preferences. The idea is that, given some preferences on individual atomic states, the theory should allow one to derive results about preferences for complex decision-making scenarios. This is analogous to the way that the truth value of a complex logical sentence is derived from the truth value of its component propositions, and the way the probability of a complex event is derived from the probability of atomic events.

LOTTERRIES

In the language of utility theory, the complex scenarios are called **lotteries** to emphasize the idea that the different attainable outcomes are like different prizes, and that the outcome is determined by chance. The lottery L , in which there are two possible outcomes—state A with probability p and state B with the remaining probability—is written

$$L = [p, A; 1 - p, B]$$

In general, a lottery can have any number of outcomes. Each outcome can be either an atomic state or another lottery. A lottery with only one outcome can be written either as A or $[1, A]$. It is the decision-maker's job to choose among lotteries. Preferences between prizes are used to determine preferences between lotteries. The following notation is used to express preferences or the lack of a preference between lotteries or states:

$A > B$ A is preferred to B

$A \sim B$ the agent is indifferent between A and B

$A \nsim B$ the agent prefers A to B or is indifferent between them

Now we impose reasonable constraints on the preference relation, much as we imposed rationality constraints on degrees of belief in order to obtain the axioms of probability in Chapter 14. One reasonable constraint is that preference should be **transitive**, that is, if $A > B$ and $B > C$, then we would expect $A > C$. We argue for transitivity by showing that an agent whose preferences do not respect transitivity would behave irrationally. Suppose, for example, that an agent has the nontransitive preferences $A > B > C > A$, where A , B , and C are goods that can be freely

exchanged. If the agent currently has A, then we could offer to trade C for A and some cash. The agent prefers C, and so would be willing to give up some amount of cash to make this trade. We could then offer to trade B for C, extracting more cash, and finally trade A for B. This brings us back where we started from, except that the agent has less money. It seems reasonable to claim that in this case the agent has not acted rationally.

The following six constraints are known as the axioms of utility theory. They specify the most obvious semantic constraints on preferences and lotteries.

ORDERABILITY

- ◊ **Orderability:** Given any two states, a rational agent must either prefer one to the other or else rate the two as equally preferable. That is, an agent should know what it wants.

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

TRANSITIVITY

- ◊ **Transitivity:** Given any three states, if an agent prefers A to B and prefers B to C, then the agent must prefer A to C.

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

CONTINUITY

- 0 **Continuity:** If some state B is between A and C in preference, then there is some probability p for which the rational agent will be indifferent between getting B for sure and the lottery that yields A with probability p and C with probability $1 - p$.

$$A \succ B \succ C \Rightarrow \exists p [p, A; 1 - p, C] \sim B$$

SUBSTITUTABILITY

- ◊ **Substitutability:** If an agent is indifferent between two lotteries, A and B, then the agent is indifferent between two more complex lotteries that are the same except that B is substituted for A in one of them. This holds regardless of the probabilities and the other outcome(s) in the lotteries.

$$A \sim B \Rightarrow [p, A; 1 - p, C] \sim [p, B; 1 - p, C]$$

MONOTONICITY

- ◊ **Monotonicity:** Suppose there are two lotteries that have the same two outcomes, A and B. If an agent prefers A to B, then the agent must prefer the lottery that has a higher probability for A (and vice versa).

$$A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1 - p, B] \succ [q, A; 1 - q, B])$$

DECOMPOSABILITY

- ◊ **Decomposability:** Compound lotteries can be reduced to simpler ones using the laws of probability. This has been called the "no fun in gambling" rule because it says that an agent should not prefer (or disprefer) one lottery just because it has more choice points than another.²

$$[p, A; 1 - p, [q, B; 1 - q, C]] \sim [p, A; (1 - p)q, B; (1 - p)(1 - q), C]$$

... and then there was Utility

Notice that the axioms of utility theory do not say anything about utility. They only talk about preferences. Preference is assumed to be a basic property of rational agents. The existence of a utility function follows from the axioms of utility:

² It is still possible to account for an agent who enjoys gambling by reifying the act of gambling itself: just include in the appropriate outcome states the proposition "participated in a gamble," and base utilities in part on that proposition.

1. Utility principle

If an agent's preferences obey the axioms of utility, then there exists a real-valued function U that operates on states such that $U(A) > U(B)$ if and only if A is preferred to B , and $U(A) = U(B)$ if and only if the agent is indifferent between A and B .

$$\begin{aligned} U(A) > U(B) &\Leftrightarrow A \succ B \\ U(A) = U(B) &\Leftrightarrow A \sim B \end{aligned}$$

2. Maximum Expected Utility principle

The utility of a lottery is the sum of the probabilities of each outcome times the utility of that outcome.

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$

In other words, once the probabilities and utilities of the possible outcome states are specified, the utility of a compound lottery involving these states can be computed. $U([p_1, S_1; p_2, S_2; \dots])$ is completely determined by $U(S_i)$ and the probability values.

It is important to remember that the existence of a utility function that describes an agent's preference behavior does not necessarily mean that the agent is *explicitly* maximizing that utility function in its own deliberations. As we showed in Chapter 2, rational behavior can be generated in any number of ways, some of which are more efficient than explicit utility maximization. By observing an agent's preferences, however, it is possible to construct the utility function that represents what it is that the agent's actions are trying to achieve.

16.3 UTILITY FUNCTIONS

Utility is a function that maps from states to real numbers. Is that all we can say about utility functions? Strictly speaking, that is it. Beyond the constraints listed earlier, an agent can have any preferences it likes. For example, an agent might prefer to have a prime number of dollars in its bank account; in which case, if it had \$16 it would give away \$3. It might prefer a dented 1973 Ford Pinto to a shiny new Mercedes. Preferences can also interact: for example, it might only prefer prime numbers of dollars when it owns the Pinto, but when it owns the Mercedes, it might prefer more dollars to less.

If all utility functions were as arbitrary as this, however, then utility theory would not be of much help because we would have to observe the agent's preferences in every possible combination of circumstances before being able to make any predictions about its behavior. Fortunately, the preferences of real agents are usually more systematic. Conversely, there are systematic ways of designing utility functions that, when installed in an artificial agent, cause it to generate the kinds of behavior we want.

The utility of money

Utility theory has its roots in economics, and economics provides one obvious candidate for a utility measure: money (or more specifically, an agent's total net assets). The almost universal exchangeability of money for all kinds of goods and services suggests that money plays a significant role in human utility functions.³

If we restrict our attention to actions that only affect the amount of money that an agent has, then it will usually be the case that the agent prefers more money to less, all other things being equal. We say that the agent exhibits a **monotonic preference** for money. This is not, however, sufficient to guarantee that money behaves as a utility function. Technically speaking, money behaves as a **value function or ordinal utility** measure, meaning just that the agent prefers to have more rather than less when considering *definite amounts*. To understand monetary decision making under uncertainty, we also need to examine the agent's preferences between lotteries involving money.

Suppose you have triumphed over the other competitors in a television game show. The host now offers you a choice: you can either take the \$1,000,000 prize or you can gamble it on the flip of a coin. If the coin comes up heads, you end up with nothing, but if it comes up tails, you get \$3,000,000. If you're like most people, you would decline the gamble and pocket the million. Are you being irrational?

Assuming you believe that the coin is fair, the **expected monetary value** (EMV) of the gamble is $\frac{1}{2}(\$0) + \frac{1}{2}(\$3,000,000) = \$1,500,000$, and the EMV of taking the original prize is of course \$1,000,000, which is less. But that does not necessarily mean that accepting the gamble is a better decision. Suppose we use S_n to denote the state of possessing total wealth \$n, and that your current wealth is \$k. Then the expected utilities of the two actions of accepting and declining the gamble are

$$\begin{aligned} EU(\text{Accept}) &= \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+3,000,000}) \\ EU(\text{Decline}) &= U(S_{k+1,000,000}) \end{aligned}$$

To determine what to do, we need to assign utilities to the outcome states. Utility is not directly proportional to monetary value because the utility — the positive change in lifestyle — for your first million is very high (or so we are told), whereas the utility for additional millions is much smaller. Suppose you assign a utility of 5 to your current financial status (S_k), a 10 to the state $S_{k+3,000,000}$, and an 8 to the state $S_{k+1,000,000}$. Then the rational action would be to decline, because the expected utility of accepting is only 7.5 (less than the 8 for declining). On the other hand, suppose that you happen to have \$500,000,000 in the bank already (and appear on game shows just for fun, one assumes). In this case, the gamble is probably acceptable, provided that you prefer more money to less, because the additional benefit of the 503rd million is probably about the same as that of the 501st million.

In 1738, long before television game shows, Bernoulli came up with an even more compelling example, known as the St. Petersburg paradox. Suppose you are offered a chance to play a game in which a fair coin is tossed repeatedly until it comes up heads. If the first heads appears on the nth toss, you win 2^n dollars. How much would you pay for a chance to play this game?

MONOTONIC
PREFERENCE

VALUE FUNCTION
ORDINAL UTILITY

EXPECTED
MONETARY VALUE

³ This despite the assurances of a well-known song to the effect that the best things in life are free.

The probability of the first head showing up on the n th toss is $1/2^n$, so the expected monetary value (EMV) of this game is

$$EMV(St.P.) = \sum_i P(Heads_i)MV(Heads_i) \approx \sum_i \frac{1}{2^i} 2^i = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = \infty$$

Thus, an agent who is trying to maximize its expected monetary gain should be willing to pay *any* finite sum for a chance to play this game. Somehow this does not seem rational to most people. Bernoulli resolved the alleged paradox by positing that the utility of money is measured on a logarithmic scale (at least for positive amounts):

$$U(S_{k+n}) = \log_2 n \quad (\text{for } n > 0)$$

With this utility function, the expected utility of the game is 2:

$$EU(St.P.) = \sum_i P(Heads_i)U(Heads_i) \approx \sum_i \frac{1}{2} \log_2 2^i = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \dots = 2$$

which means that a rational agent with the given utility scale should be willing to pay up to \$4 for a chance to play the game, because $U(S_{k+4}) = \log_2 4 = 2$.

Bernoulli chose \log_2 as the utility function just to make this problem work out. However, in a pioneering study of actual utility functions, Grayson (1960) found an almost perfect fit to the logarithmic form. One particular curve, for a certain Mr. Beard, is shown in Figure 16.1(a). The data obtained for Mr. Beard's preferences are consistent with a utility function

$$U(S_{k+n}) = -263.31 + 22.09 \log(n + 150,000)$$

for the range between $n = -\$150,000$ and $n = \$800,000$.

We should not assume that this is the definitive utility function for monetary value, but it is likely that most people have a utility function that is concave for positive wealth. Going into debt is usually considered disastrous, but preferences between different levels of debt can display a reversal of the concavity associated with positive wealth. For example, someone already

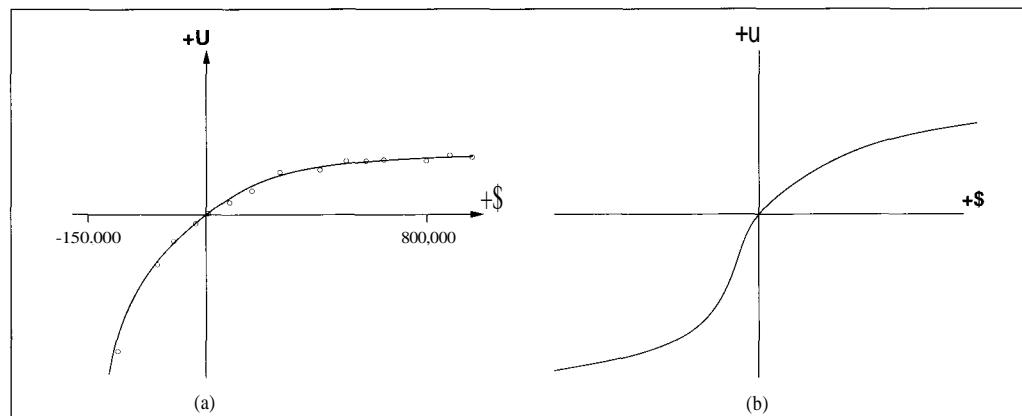


Figure 16.1 The utility of money. (a) Empirical data for Mr. Beard over a limited range. (b) A typical curve for the full range.

\$10,000,000 in debt might well accept a gamble on a fair coin with a gain of \$10,000,000 for heads and a loss of \$20,000,000 for tails.⁴ This yields the S-shaped curve shown in Figure 16.1(b).

If we restrict our attention to the positive part of the curves, where the slope is decreasing, then for any lottery L , the utility of being faced with that lottery is less than the utility of being handed the expected monetary value of the lottery as a sure thing:

$$U(S_L) < U(S_{EMV(L)})$$

RISK-AVERSE

That is, agents with curves of this shape are **risk-averse**: they prefer a sure thing with a payoff that is less than the expected monetary value of a gamble. On the other hand, in the "desperate" region at large negative wealth in Figure 16.1(b), the behavior is **risk-seeking**. The value an agent will accept in lieu of a lottery is called the **certainty equivalent** of the lottery. Studies have shown that most people will accept about \$400 in lieu of a gamble that gives \$1000 half the time and \$0 the other half—that is, the certainty equivalent of the lottery is \$400. The difference between the expected monetary value of a lottery and its certainty equivalent is called the **insurance premium**. Risk aversion is the basis for the insurance industry, because it means that insurance premiums are positive. People would rather pay a small insurance premium than gamble with the price of their house against the possibility of a fire. From the insurance company's point of view, the risk is very low because the law of large numbers makes it almost certain that the claims it pays will be substantially less than the premiums it receives.

RISK-SEEKING
CERTAINTY
EQUIVALENTINSURANCE
PREMIUM

RISK-NEUTRAL

NORMALIZED
UTILITIES

STANDARD LOTTERY

Notice that for *small* changes in wealth relative to the current wealth, almost any curve will be approximately linear. An agent that has a linear curve is said to be **risk-neutral**. For gambles with small sums, therefore, we expect risk neutrality. In a sense, this justifies the simplified procedure that proposed small gambles to assess probabilities and to justify the axioms of probability in Chapter 14.

Utility scales and utility assessment

The axioms of utility do not specify a unique utility function for an agent, given its preference behavior. It is simple to see that an agent using a utility function

$$U'(S) = k_1 + k_2 U(S),$$

where k_1 is a constant and k_2 is any *positive* constant, will behave identically to an agent using $U(S)$, provided they have the same beliefs.⁵ The scale of utilities is therefore somewhat arbitrary.

One common procedure for assessing utilities is to establish a scale with a "best possible prize" at $U(S) = u_T$ and a "worst possible catastrophe" at $U(S) = u_\perp$. **Normalized utilities** use a scale with $u_\perp = 0$ and $MT = 1$. Utilities of intermediate outcomes are assessed by asking the agent to indicate a preference between the given outcome state S and a **standard lottery** $[p, MT; (1-p), u_\perp]$. The probability p is adjusted until the agent is indifferent between S and the standard lottery. Assuming normalized utilities, the utility of S is given by p .

In medical, transportation, and environmental decision problems, among others, people's lives are at stake. In such cases, u_\perp is the value assigned to immediate death (or perhaps many

⁴ Such behavior might be called desperate, but it is nonetheless perfectly rational if one is already in a desperate situation.

⁵ In Chapter 5, we saw that move choice in *deterministic* games is unchanged by any monotonic transformation of the evaluation function, but in backgammon, where positions are lotteries, only linear transformations preserve move choice.

HUMAN JUDGMENT AND FALLIBILITY

Decision theory is a **normative** theory: it describes how a rational agent *should* act. The application of economic theory would be greatly enhanced if it were also a **descriptive** theory of actual human decision making. However, there is experimental evidence indicating that people systematically violate the axioms of utility theory. An example is given by the psychologists Tversky and Kahneman (1982), based on an example by the economist Allais (1953). Subjects in this experiment are given a choice between lotteries A and B, and then between C and D:

A :	80% chance of \$4000	C :	20% chance of \$4000
B :	100% chance of \$3000	D :	25% chance of \$3000

The majority of subjects choose B over A and C over D. But if we assign $U(0) = 0$, then the first of these choices implies that $0.8U(\$4000) < U(\$3000)$, whereas the second choice implies exactly the reverse. In other words, there seems to be no utility function that is consistent with these choices. One possible conclusion is that humans are simply irrational by the standards of our utility axioms. An alternative view is that the analysis does not take into account **regret**—the feeling that humans know they would experience if they gave up a certain reward (B) for an 80% chance at a higher reward, and then lost. In other words, if A is chosen, there is a 20% chance of getting no money and feeling like a complete idiot.

Kahneman and Tversky go on to develop a descriptive theory that explains how people are risk-averse with high-probability events, but are willing to take more risks with unlikely payoffs. The connection between this finding and AI is that the choices our agents can make are only as good as the preferences they are based on. If our human informants insist on contradictory preference judgments, there is nothing our agent can do to be consistent with them.

Fortunately, preference judgments made by humans are often open to revision in the light of further consideration. In early work at Harvard Business School on assessing the utility of money, Keeney and Raiffa (1976, p. 210) found the following:

A great deal of empirical investigation has shown that there is a serious deficiency in the assessment protocol. Subjects tend to be too risk-averse in the small and therefore . . . the fitted utility functions exhibit unacceptably large risk premiums for lotteries with a large spread. . . . Most of the subjects, however, can reconcile their inconsistencies and feel that they have learned an important lesson about how they want to behave. As a consequence, some subjects cancel their automobile collision insurance and take out more term insurance on their lives.

Even today, human (ir)rationality is the subject of intensive investigation.



deaths). Although nobody feels comfortable with putting a value on human life, it is a fact that trade-offs are made all the time. Aircraft are given a complete overhaul at intervals determined by trips and miles flown, rather than after every trip. Car bodies are made with relatively thin sheet metal to reduce costs, despite the decrease in accident survival rates. Leaded fuel is still widely used even though it has known health hazards. Paradoxically, a refusal to "put a monetary value on life" means that life is often *undervalued*. Ross Shachter relates an experience with a government agency that commissioned a study on removing asbestos from schools. The study assumed a particular dollar value for the life of a school-age child, and argued that the rational choice under that assumption was to remove the asbestos. The government agency, morally outraged, rejected the report out of hand. It then decided against asbestos removal.

Some attempts have been made to find out the value that people place on their own lives. Two common "currencies"⁶ used in medical and safety analysis are the **micromort** (a one in a million chance of death) and the **QALY**, or quality-adjusted life year (equivalent to a year in good health with no infirmities). A number of studies across a wide range of individuals have shown that a micromort is worth about \$20 (1980 dollars). We have already seen that utility functions need not be linear, so this does not imply that a decision maker would kill himself for \$20 million. Again, the local linearity of any utility curve means that micromort and QALY values are useful for small incremental risks and rewards, but not necessarily for large risks.

MICROMORT
QALY

16.4 MULTIATTRIBUTE UTILITY FUNCTIONS

MULTIATTRIBUTE
UTILITY THEORY

Decision making in the field of public policy involves both millions of dollars and life and death. For example, in deciding what levels of a carcinogenic substance to allow into the environment, policy makers must weigh the prevention of deaths against the economic hardship that might result from the elimination of certain products and processes. Siting a new airport requires consideration of the disruption caused by construction; the cost of land; the distance from centers of population; the noise of flight operations; safety issues arising from local topography and weather conditions; and so on. Problems like these, in which outcomes are characterized by two or more attributes, are handled by **multiattribute utility theory**, or MAUT.

We will call the attributes X_1 , X_2 , and so on; their values will be x_1 , x_2 , and so on; and a complete vector of attribute values will be $\mathbf{x} = \langle x_1, x_2, \dots \rangle$. Each attribute is generally assumed to have discrete or continuous scalar values. For simplicity, we will assume that each attribute is defined in such a way that, all other things being equal, higher values of the attribute correspond to higher utilities. For example, if we choose as an attribute in the airport problem *AbsenceOfNoise*, then the greater its value, the better the solution. In some cases, it may be necessary to subdivide the range of values so that utility varies monotonically within each range.

REPRESENTATION
THEOREMS

The basic approach adopted in multiattribute utility theory is to identify regularities in the preference behavior we would expect to see, and to use what are called **representation theorems**.

⁶ We use quotation marks because these measures are definitely not currencies in the standard sense of being exchangeable for goods at constant rates regardless of the current "wealth" of the agent making the exchange.

to show that an agent with a certain kind of preference structure has a utility function

$$U(x_1, \dots, x_n) = f[f_1(x_1), \dots, f_n(x_n)]$$

where f is, we hope, a simple function such as addition. Notice the similarity to the use of belief networks to decompose the joint probability of several random variables.

Dominance

STRICT DOMINANCE

Suppose that airport site S_1 costs less, generates less noise pollution, and is safer than site S_2 . One would not hesitate to reject S_2 . We say that there is **strict dominance** of S_1 over S_2 . In general, if an option is of lower value on all attributes than some other option, it need not be considered further. Strict dominance is often very useful in narrowing down the field of choices to the real contenders, although it seldom yields a unique choice. Figure 16.2(a) shows a schematic diagram for the two-attribute case.

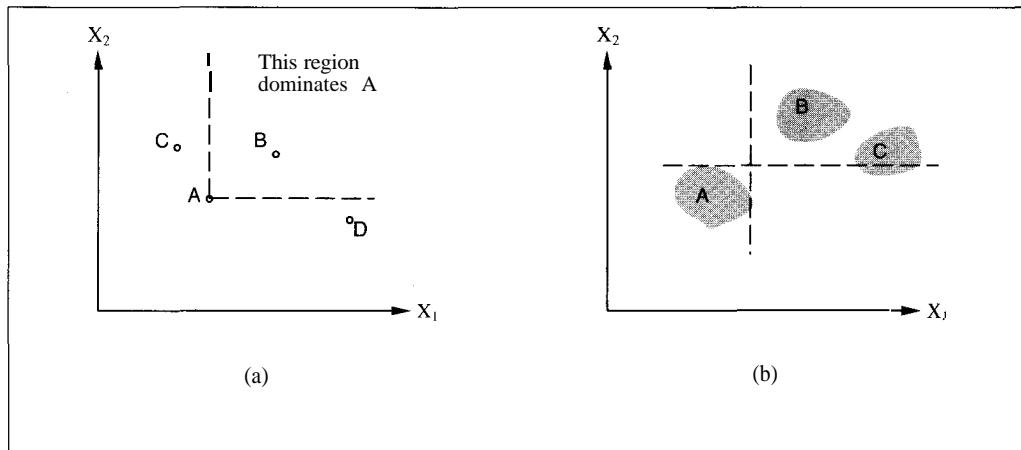


Figure 16.2 Strict dominance. (a) Deterministic: Option A is strictly dominated by B but not by C or D. (b) Uncertain: A is strictly dominated by B but not by C.

STOCHASTIC DOMINANCE

That is fine for the deterministic case, in which the attribute values are known for sure. What about the general case, where the action outcomes are uncertain? A direct analogue of strict dominance can be constructed, where, despite the uncertainty, all possible concrete outcomes for S_1 strictly dominate all possible outcomes for S_2 . (See Figure 16.2(b) for a schematic depiction.) Of course, this will probably occur even less often than in the deterministic case.

Fortunately, there is a more useful generalization called **stochastic dominance**, which occurs very frequently in real problems. Stochastic dominance is easiest to understand in the context of a single attribute. Suppose that we believe the cost of siting the airport at S_1 to be normally distributed around \$3.7 billion, with standard deviation \$0.4 billion; and the cost at S_2 to be normally distributed around \$4.0 billion, with standard deviation \$0.35 billion. Figure 16.3(a) shows these distributions, with cost plotted as a negative value. Then, given only the information

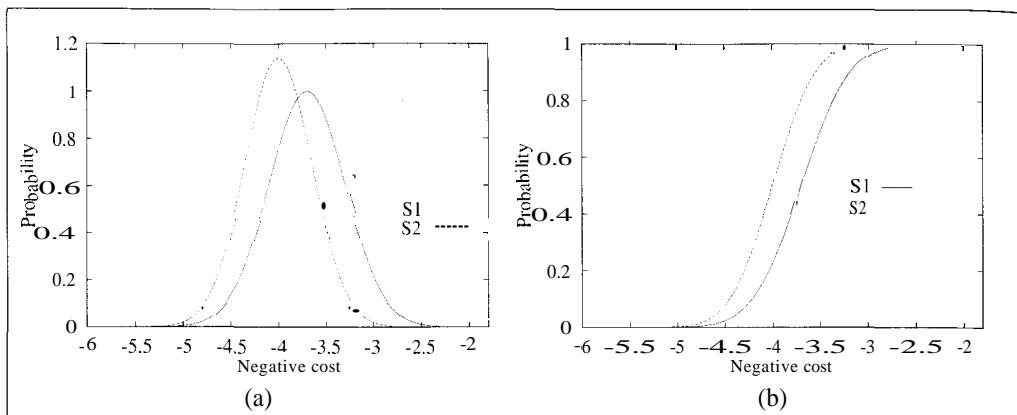


Figure 16.3 Stochastic dominance. (a) S_1 stochastically dominates S_2 on cost. (b) Cumulative distributions for the negative cost of S_1 and S_2 .

that utility decreases with cost, we can say that S_1 stochastically dominates S_2 — that is, S_2 can be discarded. It is important to note that this does *not* follow from comparing the expected costs. For example, if we knew the cost of S_1 to be *exactly* \$3.7 billion, then we would be *unable* to make a decision without additional information on the utility of money.⁷

The exact relationship between the attribute distributions needed to establish stochastic dominance is best seen by examining the *cumulative* distributions, shown in Figure 16.3(b). The cumulative distribution measures the probability that the cost is less than or equal to any given amount — that is, it integrates the original distribution. If the cumulative distribution for S_1 is always to the right of the cumulative distribution for S_2 , then stochastically speaking S_1 is cheaper than S_2 . Formally, if two actions A_1 and A_2 lead to probability distributions $p_1(x)$ and $p_2(x)$ on attribute X , then A_1 stochastically dominates A_2 on X if

$$\forall x \int_{-\infty}^x p_1(x') dx' \geq \int_{-\infty}^x p_2(x') dx$$

If an action is stochastically dominated by another action on all attributes, then it can be discarded.

The stochastic dominance condition might seem rather technical, and perhaps not so easy to determine without extensive probability calculations. In fact, it can be decided very easily in many cases. Suppose, for example, that the construction cost depends on the distance to centers of population. The cost itself is uncertain, but the greater the distance, the greater the cost. If S_1 is less remote than S_2 , then S_1 will dominate S_2 on cost. Although we will not present them here, there exist algorithms for propagating this kind of qualitative information among uncertain variables in **qualitative probabilistic networks**, enabling a system to make rational decisions based on stochastic dominance without ever needing to use numerical probabilities or utilities.

⁷ It might seem odd that *more* information on the cost of S_1 could make the agent *less* able to decide. The paradox is resolved by noting that the decision reached in the absence of exact cost information is less likely to be correct.

Preference structure and multiattribute utility

Suppose we have n attributes, each of which has m distinct possible values. This gives a set of possible outcomes of size m^n . In the worst case, the agent's utility function yields an arbitrary set of preferences over these m^n states with no regularities beyond those implied by the basic axioms of utility. Multiattribute utility theory is based on the supposition that most utility functions have much more structure than that, allowing us to use simplified decision procedures.

Preferences without uncertainty

PREFERENCE INDEPENDENCE

Let us begin by considering the case in which there is no uncertainty in the outcomes of actions, and we are just considering preferences between concrete outcomes. In this case, the basic regularity in preference structure is called **preference independence**. Two attributes X_1 and X_2 are preferentially independent of a third attribute X_3 if the preference between outcomes $\langle x_1, x_2, x_3 \rangle$ and $\langle x'_1, x'_2, x_3 \rangle$ does not depend on the particular value x_3 for attribute X_3 .

MUTUAL PREFERENTIAL INDEPENDENCE

Going back to the airport example, where we have (among other attributes) *Noise*, *Cost*, and *Deaths* to consider, one may propose that *Noise* and *Cost* are preferentially independent of *Deaths*. For example, if we prefer a state with 20,000 people residing in the flight path and a construction cost of \$4 billion to a state with 70,000 people residing in the flight path and a cost of \$3.7 billion when the safety level is 0.06 deaths per million passenger miles in both cases, then we would have the same preference when the safety level is 0.13 or 0.01; and the same independence would hold for preferences between any other pair of values for *Noise* and *Cost*. It is also apparent that *Cost* and *Deaths* are preferentially independent of *Noise*, and that *Noise* and *Deaths* are preferentially independent of *Cost*. We say that the set of attributes $\{\text{Noise}, \text{Cost}, \text{Deaths}\}$ exhibits **mutual preferential independence** (MPI). MPI says that whereas each attribute may be important, it does not affect the way in which one trades off the other attributes against each other.



Mutual preferential independence is something of a mouthful, but thanks to a remarkable theorem due to the economist Debreu (1960), we can derive from it a very simple form for the agent's value function: *If attributes X_1, \dots, X_n are mutually preferentially independent, then the agent's preference behavior can be described as maximizing the function*

$$V(S) = \sum_i V_i(X_i(S))$$

where each V_i is a value function referring only to the attribute X_i . For example, it might well be the case that the airport decision can be made using a value function

$$V(S) = -\text{Noise} \times 10^4 - \text{Cost} - \text{Deaths} \times 10^{12}$$

ADDITIONAL VALUE FUNCTION

A value function of this type is called an **additive value function**. Additive functions are an extremely natural way to describe an agent's value function, and are valid in many real-world situations. Even when MPI does not strictly hold, as might be the case at extreme values of the attributes, an additive value function can still provide a good approximation to the agent's preferences. This is especially true when the violations of MPI occur in portions of the attribute ranges that are unlikely to occur in practice.

Preferences with uncertainty

When uncertainty is present in the domain, we will also need to consider the structure of preferences between lotteries and to understand the resulting properties of utility functions, rather than just value functions. The mathematics of this problem can become quite complicated, so we will give just one of the main results to give a flavor of what can be done. The reader is referred to Keeney and Raiffa (1976) for a thorough survey of the field.

UTILITY
INDEPENDENCE

MUTUALLY UTILITY-
INDEPENDENT

MULTIPLICATIVE
UTILITY FUNCTION

The basic notion of **utility independence** extends preference independence to cover lotteries: a set of attributes X is utility-independent of a set of attributes Y if preferences between lotteries on the attributes in X are independent of the particular values of the attributes in Y. A set of attributes is **mutually utility-independent** (MUI) if each subset is utility-independent of the remaining attributes. Again, it seems reasonable to propose that the airport attributes are MUI.

MUI implies that the agent's behavior can be described using a **multiplicative utility function** (Keeney, 1974). The general form of a multiplicative utility function is best seen by looking at the case for three attributes. For simplicity, we will use U_i to mean $U_i(X_i(s))$:

$$U = k_1 U_1 + k_2 U_2 + k_3 U_3 + k_1 k_2 U_1 U_2 + k_2 k_3 U_2 U_3 + k_3 k_1 U_3 U_1 + k_1 k_2 k_3 U_1 U_2 U_3$$

Although this does not look very simple, it contains three single-attribute utility functions and just three constants. In general, an n -attribute problem exhibiting MUI can be modelled using n single-attribute utilities and n constants. Each of the single-attribute utility functions can be developed independently of the other attributes, and this combination will be guaranteed to generate the correct overall preferences. Additional assumptions are required to obtain a purely additive utility function.

16.5 DECISION NETWORKS

INFLUENCE DIAGRAM
DECISION NETWORK

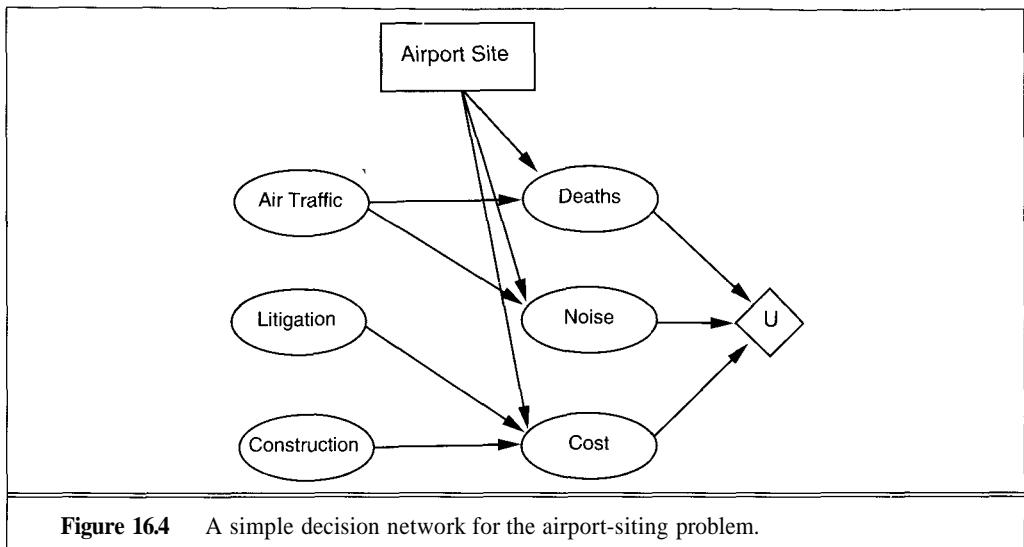
In this section, we will look at a general mechanism for making rational decisions. The notation is often called a **influence diagram** (Howard and Matheson, 1984), but we will use the more descriptive term **decision network**. Decision networks combine belief networks with additional node types for actions and utilities. We will use the airport siting problem as an example.

Representing a decision problem using decision networks

In its most general form, a decision network represents information about the agent's current state, its possible actions, the state that will result from the agent's action, and the utility of that state. It therefore provides a substrate for implementing utility-based agents of the type first introduced in Section 2.3. Figure 16.4 shows a decision network for the airport siting problem. It illustrates the three types of nodes used:

CHANCE NODES

- 0 **Chance nodes** (ovals) represent random variables, just as they do in belief nets. The agent may be uncertain about the construction cost, the level of air traffic and the potential for litigation, as well as the *Deaths*, *Noise*, and total *Cost* variables, each of which also depends



on the site chosen. Each chance node has associated with it a conditional probability table (CPT) that is indexed by the state of the parent nodes. In decision networks, the parent nodes can include decision nodes as well as chance nodes. Note that each of the current-state chance nodes could be part of a large belief network for assessing construction costs, air traffic levels, or litigation potential. For simplicity, these are omitted.

DECISION NODES

0 Decision nodes (rectangles) represent points where the decision-maker has a choice of actions. In this case, the *AirportSite* action can take on a different value for each site under consideration. The choice influences the cost, safety, and noise that will result. In this chapter, we will assume that we are dealing with a single decision node. Chapter 17 deals with cases where more than one decision must be made.

UTILITY NODES

◊ **Utility nodes** (diamonds) represent the agent's utility function.⁸ The utility node has as parents all those variables describing the outcome state that directly affect utility. The table associated with a utility node is thus a straightforward tabulation of the agent's utility as a function of the attributes that determine it. As with canonical CPTs, multiattribute utility functions can be represented by a structured description rather than a simple tabulation.

ACTION-UTILITY TABLES

A simplified form is also used in many cases. The notation remains identical, but the chance nodes describing the outcome state are omitted. Instead, the utility node is connected directly to the current-state nodes and the decision node. In this case, rather than representing a utility function on states, the table associated with the utility node represents the *expected* utility associated with each action, as defined in Equation (16.1). We therefore call such tables **action-utility tables**. Figure 16.5 shows the action-utility representation of the airport problem.

Notice that because the *Noise*, *Deaths*, and *Cost* chance nodes in Figure 16.4 refer to future states, they can never have their values set as evidence variables. Thus, the simplified version

⁸ These nodes are often called **value nodes** in the literature. We prefer to maintain the distinction between utility and value functions, as discussed earlier, because the outcome state may represent a lottery.

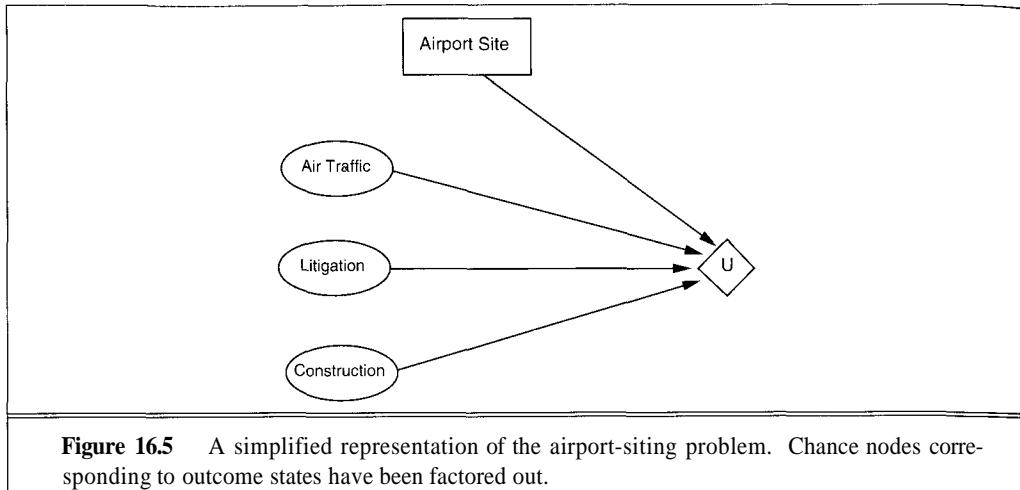


Figure 16.5 A simplified representation of the airport-siting problem. Chance nodes corresponding to outcome states have been factored out.

that omits these nodes can be used whenever the more general form can be used. Although the simplified form contains fewer nodes, the omission of an explicit description of the outcome of the siting decision means that it is less flexible with respect to changes in circumstances. For example, in Figure 16.4, a change in aircraft noise levels can be reflected by a change in the conditional probability table associated with the *Noise* node, whereas a change in the weight accorded to noise pollution in the utility function can be reflected by a change in the utility table. In the action-utility diagram, Figure 16.5, on the other hand, all such changes have to be reflected by changes to the action-utility table. Essentially, the action-utility formulation is a *compiled* version of the original formulation.

Evaluating decision networks

Actions are selected by evaluating the decision network for each possible setting of the decision node. Once the decision node is set, it behaves exactly like a chance node that has been set as an evidence variable. The algorithm for evaluating decision networks is the following:

1. Set the evidence variables for the current state.
2. For each possible value of the decision node:
 - (a) Set the decision node to that value.
 - (b) Calculate the posterior probabilities for the parent nodes of the utility node, using a standard probabilistic inference algorithm.
 - (c) Calculate the resulting utility for the action.
3. Return the action with the highest utility.

This is a straightforward extension of the belief network algorithm, and can be incorporated directly into the agent design given in Figure 14.1. We will see in Chapter 17 that the possibility of executing several actions in sequence makes the problem much more interesting.

16.6 THE VALUE OF INFORMATION



I

INFORMATION VALUE
THEORY

In the preceding analysis, we have assumed that all relevant information, or at least all available information, is provided to the agent before it makes its decision. In practice, this is hardly ever the case. *One of the most important parts of decision making is knowing what questions to ask.* For example, a doctor cannot expect to be provided with the results of *all possible* diagnostic tests and questions at the time a patient first enters the consulting room.⁹ Tests are often expensive and sometimes hazardous (both directly and because of associated delays). Their importance depends on two factors: whether the different possible outcomes would make a significant difference to the optimal course of action, and the likelihood of the various outcomes.

This section describes **information value theory**, which enables an agent to choose what information to acquire. The acquisition of information is achieved by **sensing actions**, as described in Chapter 13. Because the agent's utility function seldom refers to the contents of the agent's internal state, whereas the whole purpose of sensing actions is to affect the internal state, we must evaluate sensing actions by their effect on the agent's subsequent actions. Information value theory is therefore a special kind of sequential decision making.

A simple example

Suppose an oil company is hoping to buy one of n indistinguishable blocks of ocean drilling rights. Let us assume further that exactly one of the blocks contains oil worth C dollars, and that the price of each block is C/n dollars. If the company is risk-neutral, then it will be indifferent between buying a block or not.

Now suppose that a seismologist offers the company the results of a survey of block number 3, which indicates definitively whether the block contains oil. How much should the company be willing to pay for the information? The way to answer this question is to examine what the company would do if it had the information:

- With probability $1/n$, the survey will show that block 3 contains the oil. In this case, the company will buy block 3 for C/n dollars, and make a profit of $C - C/n = (n - 1)C/n$ dollars.
- With probability $(n - 1)/n$, the survey will show that the block contains no oil, in which case the company will buy a different block. Now the probability of finding oil in one of the other blocks changes from $1/n$ to $1/(n - 1)$, so the company makes an expected profit of $C(n - 1) - C/n = C/n(n - 1)$ dollars.

Now we can calculate the expected profit given the survey information:

$$\frac{1}{n} \times \frac{(n - 1)C}{n} + \frac{n - 1}{n} \times \frac{C}{n(n - 1)} = C/n$$

Therefore, the company should be willing to pay the seismologist up to C/n dollars for the information: the information is worth as much as the block itself.

⁹ In the United States, the only question that is always asked beforehand is whether the patient has insurance.

The value of information derives from the fact that *with* the information, one's course of action may change to become more appropriate to the actual situation. One can discriminate according to the situation, whereas without the information one has to do what's best on average over the possible situations. In general, the value of a given piece of information is defined to be the difference in expected value between best actions before and after information is obtained.

VALUE OF PERFECT INFORMATION

A general formula

It is simple to derive a general mathematical formula for the value of information. Usually, we assume that exact evidence is obtained about the value of some random variable E_j , so the phrase **value of perfect information** (VPI) is used. Let the agent's current knowledge be E . Then the value of the current best action a is defined by

$$EU(\alpha|E) = \max_A \sum_i U(Result_i(A))P(Result_i(A)|E, Do(A))$$

and the value of the new best action (after the new evidence E_j is obtained) will be

$$EU(\alpha_{E_j}|E, E_j) = \max_A \sum_i U(Result_i(A)) P(Result_i(A)|E, Do(A), E_j)$$

But E_j is a random variable whose value is *currently* unknown, so we must average over all possible values e_{jk} that we might discover for E_j , using our *current* beliefs about its value. The value of discovering E_j is then defined as

$$VPI_E(E_j) = \left(\sum_k P(E_j = e_{jk}|E) EU(\alpha_{e_{jk}}|E, E_j = e_{jk}) \right) - EU(\alpha|E)$$

In order to get some intuition for this formula, consider the simple case where there are only two actions A_1 and A_2 from which to choose. Their current expected utilities are U_1 and U_2 . The information E_j will yield some new expected utility U'_1 and U'_2 for the actions, but before we obtain E_j , we will have some probability distributions over the possible values of U'_1 and U'_2 (which we will assume are independent).

Suppose that A_1 and A_2 represent two different routes through a mountain range in winter. A_1 is a nice, straight highway through a low pass, and A_2 is a winding dirt road over the top. Just given this information, A_1 is clearly preferable, because it is quite likely that the second route is blocked by avalanches, whereas it is quite unlikely that the first route is blocked by traffic. U_1 is therefore clearly higher than U_2 . It is possible to obtain satellite reports E_j on the actual state of each road, which would give new expectations U'_1 and U'_2 for the two crossings. The distributions for these expectations are shown in Figure 16.6(a). Obviously, in this case, it is not worth the expense of obtaining satellite reports, because it is so unlikely that they will cause a change of plan. With no change of plan, information has no value.

Now suppose that we are choosing between two different winding dirt roads of slightly different lengths, and we are carrying a seriously injured passenger. Then, although U_1 and U_2 may be quite close, the distributions of U'_1 and U'_2 are very broad. There is a significant possibility that the second route will turn out to be clear whereas the first is blocked, and in this case the difference in utilities will be very high. The VPI formula indicates that it might be worth getting the satellite reports. This situation is shown in Figure 16.6(b).

Now suppose that we are choosing between the two dirt roads in summertime, when blockage by avalanches is unlikely. In this case, satellite reports might show one route to be more scenic than the other because of flowering alpine meadows, or perhaps wetter because of errant streams. It is therefore quite likely that we would change our plan if we had the information. But in this case, the difference in value between the two routes is still likely to be very small, so we will not bother to obtain the reports. This situation is shown in Figure 16.6(c).

In summary, we can say that *information has value to the extent that it is likely to cause a change of plan, and to the extent that the new plan will be significantly better than the old plan.*

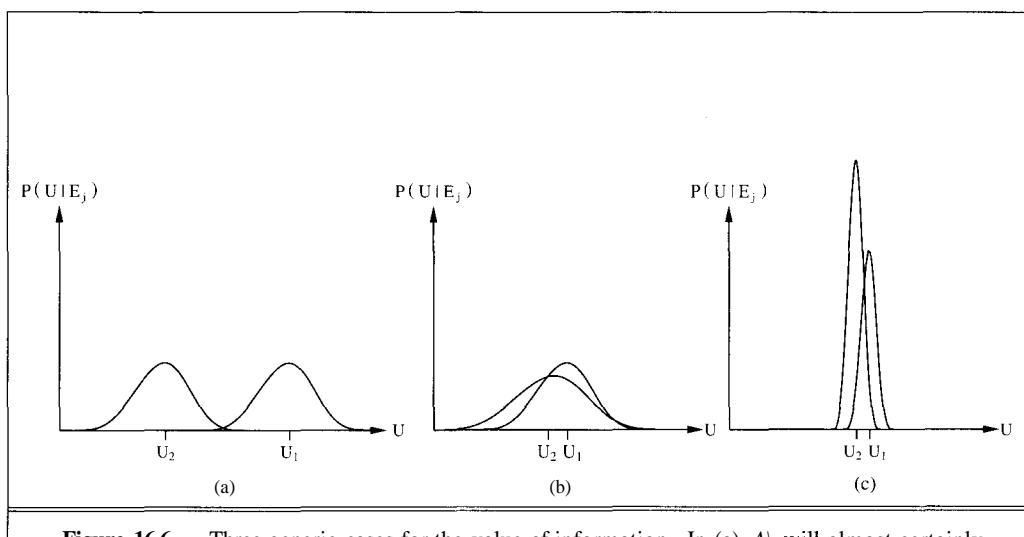


Figure 16.6 Three generic cases for the value of information. In (a), A_1 will almost certainly remain superior to A_2 , so the information is not needed. In (b), the choice is unclear and the information is crucial. In (c), the choice is unclear but because it makes little difference, the information is less valuable.

Properties of the value of information

One might ask if it is possible for information to be deleterious—can it actually have negative expected value? Intuitively, one should expect this to be impossible. After all, one could in the worst case just ignore the information and pretend one has never received it. This is confirmed by the following theorem, which applies to any decision-theoretic agent: *The value of information is nonnegative:*

$$\forall j, E \quad VPI_E(E_j) \geq 0$$

proof as an exercise (Exercise 16.11). It is important to remember that VPI depends on the current state of information, which is why it is subscripted. It can change as more information is acquired. In the extreme case, it will become zero if the variable in question already has a known



value. Thus, VPI is not additive. That is,

$$VPI_E(E_j E_k) \neq VPI_E(E_j) + VPI_E(E_k) \quad (\text{in general})$$

It is, however, order-independent, which should be intuitively obvious. That is,

$$VPI_E(E_j, E_k) = VPI_E(E_j) + VPI_{E,E_j}(E_k) = VPI_E(E_k) + VPI_{E,E_k}(E_j)$$

Order independence distinguishes sensing actions from ordinary actions, and simplifies the problem of calculating the value of a sequence of sensing actions.

Implementing an information-gathering agent

As we mentioned earlier, a sensible agent should ask questions of the user in a reasonable order, should avoid asking questions that are irrelevant, should take into account the importance of each piece of information in relation to its cost, and should stop asking questions when appropriate. All of these capabilities can be achieved by using the value of information as a guide.

Figure 16.7 shows the overall design of an agent that can gather information intelligently before acting. For now, we will assume that with each observable evidence variable E_j , there is an associated cost, $Cost(E_j)$, which reflects the cost of obtaining the evidence through tests, consultants, questions, or whatever. The agent requests what appears to be the most valuable piece of information, compared to its cost. We assume that the result of the action $Request(E_j)$ is that the next percept provides the value of E_j . If no observation is worth its cost, the agent selects a non-information-gathering action.

```
function INFORMATION-GATHERING-AGENT(percept) returns an action
  static: D, a decision network
    integrate percept into D
    j  $\leftarrow$  the value that maximizes  $VPI(E_j) - Cost(E_j)$ 
    if  $VPI(E_j) > Cost(E_j)$ 
      then return REQUEST(Ej)
    else return the best action from D
```

Figure 16.7 Design of a simple information-gathering agent. The agent works by repeatedly selecting the observation with the highest information value, until the costs of observing are greater than the benefits.

The agent algorithm we have described implements a form of information gathering that is called **myopic**. This is because it uses the VPI formula short-sightedly, calculating the value of information assuming that only a single evidence variable will be acquired. If there is no single evidence variable that will help a lot, a myopic agent may hastily take an action when it would have been better to request two or more variables first, and then take action. Myopic control is based on the same heuristic idea as greedy search, and often works well in practice. (For example, it has been shown to outperform expert physicians in selecting diagnostic tests.)

However, a perfectly rational information-gathering agent should consider all possible sequences of information requests terminating in an external action. Because VPI is order-independent, this is somewhat simplified by the fact that any permutations of a given sequence of information requests has the same value. Thus, one need consider only subsets of the possible information requests, without worrying about ordering.

16.7 DECISION-THEORETIC EXPERT SYSTEMS

DECISION ANALYSIS

The field of **decision analysis**, which evolved in the 1950s and 1960s, studies the application of decision theory to actual decision problems. It is used to help make rational decisions in important domains where the stakes are high, such as business, government, law, military strategy, medical diagnosis and public health, engineering design, and resource management. The process involves a careful study of the possible actions and outcomes as well as the preferences placed on each outcome. It is traditional in decision analysis to talk about two roles: the **decision maker** states preferences between outcomes, and the **decision analyst** enumerates the possible actions and outcomes and elicits preferences from the decision maker to determine the best course of action. Until the early 1980s, the use of computers in decision analysis was quite limited, and the main purpose of analysis was seen as helping humans to make decisions that actually reflect their own preferences.

DECISION MAKER DECISION ANALYST

As we discussed in Chapter 15, early expert system research concentrated on answering questions, rather than making decisions. Those systems that did recommend actions rather than providing opinions on matters of fact generally did so using condition-action rules, rather than with explicit representations of outcomes and preferences. The eventual emergence of belief networks made it possible to build large-scale systems that generated sound probabilistic inferences from evidence. The addition of decision networks means that expert systems can be developed that recommend optimal decisions, reflecting the preferences of the user as well as the available evidence.

There are many advantages that accrue from the inclusion of explicit utility models and calculations in the expert system framework. The expert benefits from the process of making his or her (or the client's) preferences explicit, and the system can automate the action selection process as well as the process of drawing conclusions from evidence. A system that incorporates utilities can avoid one of the most common pitfalls associated with the consultation process: confusing likelihood and importance. A common strategy in early medical expert systems, for example, was to rank possible diagnoses in order of likelihood, and report the most likely. Unfortunately, this can be disastrous, because it will miss cases of relatively rare, but treatable, conditions that are easily confused with more common diseases. The confusion of Hodgkin's disease (a form of cancer) with mononucleosis (a mild and very common viral infection) is a classic case in point. (For the majority of patients in general practice, moreover, the most *likely* diagnosis is "There's nothing wrong with you.") Obviously, a testing or treatment plan should depend both on probabilities and utilities. Finally, the availability of utility information helps in the knowledge engineering and consultation process, as we now explain.

The knowledge engineering process required for building and using decision-theoretic expert system is as follows:

- **Determine the scope of the problem.** Determine what are the possible actions, outcomes, and evidence to consider. Normally, the analyst will have to interview one or more experts in the domain to discover the important factors. Note that we recommended the same sort of determination as the first step of knowledge engineering in Section 8.2.
- **Lay out the topology.** Once all the relevant factors are determined, we need to know which ones are influenced by which others. It is particularly important to understand which aspects of the outcome state determine its utility.
- **Assign probabilities.** In decision networks, the conditional probabilities reflect not only causal influences between random variables, but also the effects of actions.
- **Assign utilities.** A utility function is often assessed using the techniques described earlier. Computer programs exist that automate the task of extracting preferences for various lotteries and constructing a utility function. Identifying the preference structure of multiattribute utility functions is also vital in reducing the dimensionality of the assessment problem. It can reduce the number of questions exponentially.
- **Enter available evidence.** For each specific case in which the system is used, there may be some initial evidence available.
- **Evaluate the diagram.** Calculate the optimal action according to the existing evidence.
- **Obtain new evidence.** Calculate the value of information, comparing it with the costs of acquisition, and perform the appropriate observations, if any. Notice that purely inferential expert systems, without utilities, cannot decide what new evidence to acquire.
- **Perform sensitivity analysis.** This important step checks to see if the best decision is sensitive to small changes in the assigned probabilities and utilities by systematically varying these parameters and running the evaluation again. If small changes lead to significantly different decisions, then it may be worthwhile to spend more resources to collect better data. If all variations lead to the same decision, then the user will have more confidence that it is the right decision.

Sensitivity analysis is particularly important, because one of the main criticisms of probabilistic approaches to expert systems is that it is too difficult to assess the numerical probabilities required. Sensitivity analysis often reveals that many of the numbers need only be specified very approximately—within, say, 0.2 of the value that might be obtained from an exhaustive analysis. Some systems allow probabilities to be specified as ranges. This leads to ranges for the utilities of actions. If the range of one action dominates the ranges of all others, then no further probability assessment need occur. For example, it might be the case that the optimal siting of an airport is insensitive to the predicted air traffic over a large range of values, given the system's beliefs about the other relevant factors, so that the user can remain unruffled by his or her lack of expertise on air traffic prediction.

16.8 SUMMARY

This chapter shows how to combine utility theory with probability to enable an agent to select actions that will maximize its expected performance.

- **Probability theory** describes what an agent should believe on the basis of evidence, **utility theory** describes what an agent wants, and **decision theory** puts the two together to describe what an agent should do.
- We can use decision theory to build a system that makes decisions by considering all possible actions and choosing the one that leads to the best expected outcome. Such a system is known as a **rational agent**.
- Utility theory shows that an agent whose preferences between lotteries are consistent with a set of simple axioms can be described as possessing a utility function; furthermore, the agent selects actions as if maximizing its expected utility.
- **Multiattribute utility theory** deals with utilities that depend on several distinct attributes of states. **Stochastic dominance** is a particularly useful technique for making unambiguous decisions even without precise utility values for attributes.
- **Decision networks** provide a simple formalism for expressing and solving decision problems. They are a natural extension of belief networks, containing decision and utility nodes in addition to chance nodes.
- Sometimes solving a problem involves finding more information before making a decision. The **value of information** is defined as the expected improvement in utility compared to making a decision without the information.
- **Expert systems** that incorporate utility information have additional capabilities compared to pure inference systems. In addition to being able to make decisions, they can decide to acquire information based on its value, and they can calculate the sensitivity of their decisions to small changes in probability and utility assessments.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

One of the earliest applications of the principle of maximum expected utility (although a deviant one involving infinite utilities) was Pascal's Wager, first published as part of the *Port-Royal Logic* (Arnauld, 1662). The derivation of numerical utilities from preference (utility ordering) was first carried out by Ramsey (1931); the axioms for preference in the present text are closer in form to those rediscovered in *Theory of Games and Economic Behavior* (Von Neumann and Morgenstern, 1944). A good presentation of these axioms, in the course of a discussion on risk preference, is given by Howard (1977). Ramsey had derived subjective probabilities (not just utilities) from an agent's preferences; Savage (1954) and Jeffrey (1983) carry out more recent constructions of this kind. Von Winterfeldt and Edwards (1986) provide a modern perspective on decision analysis and its relationship to human preference structures.

The St. Petersburg paradox was first presented by Bernoulli (1738). Jeffrey (1983) presents a resolution of the paradox based not on logarithmic utility functions, but on denying that playing the game is a real possibility (because no one could have a bank from which rewards of arbitrarily high utility could be paid out).

The micromort utility measure is discussed by Howard (1989). QALYs are much more widely used in medical and social policy decision-making than are micromorts; see (Russell, 1990) for a typical example of an argument for a major change in public health policy on grounds of increased expected utility measured in QALYs.

The book *Decisions with Multiple Objectives: Preferences and Value Trade-Offs* (Keeney and Raiffa, 1976) gives a thorough introduction to multiattribute utility theory. It describes early computer implementations of methods for eliciting the necessary parameters for a multiattribute utility function, and includes extensive accounts of real applications of the theory. In AI, the principal reference for MAUT is Wellman's (1985) paper, which includes a system called URP (Utility Reasoning Package) that can use a collection of statements about preference independence and conditional independence to analyze the structure of decision problems. The use of stochastic dominance together with qualitative probability models was investigated extensively by Wellman (1988; 1990). Wellman and Doyle (1992) provide a preliminary sketch of how a complex set of utility-independence relationships might be used to provide a structured model of a utility function, in much the same way that belief networks provide a structured model of joint probability distributions.

Decision theory has been a standard tool in economics, finance, and management science since the 1950s. Until the 1980s, decision trees were the main tool used for representing simple decision problems. Decision networks or influence diagrams were introduced by Howard and Matheson (1984), although the underlying concepts were developed much earlier by a group (including Howard and Matheson) at SRI (Miller *et al.*, 1976). Howard and Matheson's method involved the derivation of a decision tree from a decision network, but in general the tree is of exponential size. Shachter (1986) developed a method for making decisions based directly on a decision network, without the creation of an intermediate decision tree. The collection by Oliver and Smith (1990) has a number of useful articles on decision networks, as does the 1990 special issue of the journal *Networks*. Papers on decision networks and utility modelling also appear regularly in the journal *Management Science*.

Information value theory was first analyzed by Ron Howard (1966). His paper ends with the following remark:

If information value theory and associated decision theoretic structures do not in the future occupy a large part of the education of engineers, then the engineering profession will find that its traditional role of managing scientific and economic resources for the benefit of man has been forfeited to another profession.

To date, the implied revolution in managerial methods has not occurred, although as the use of information value theory in systems such as Pathfinder becomes more widespread, it may yet become part of every decision-maker's armory.

Surprisingly few AI researchers adopted decision-theoretic tools after the early applications in medical decision making described in Chapter 14. One of the few exceptions was Jerry Feldman, who applied decision theory to problems in vision (Feldman and Yakimovsky, 1974) and

planning (Feldman and Sproull, 1977). After the resurgence of interest in probabilistic methods in AI in the 1980s, decision-theoretic expert systems gained widespread acceptance (Horvitz *et al.*, 1988)—in fact, from 1991 onward, the cover design of the journal *Artificial Intelligence* has depicted a decision network, although some artistic license appears to have been taken with the direction of the arrows.

EXERCISES

16.1 (Adapted from David Heckerman.) This exercise concerns the **Almanac Game**, which is used by decision analysts to calibrate numeric estimations. For each of the questions below, give your best guess of the answer, that is, a number that you think is as likely to be too high as it is to be too low. Also give your guess at a 25th percentile estimate, that is, a number that you think has a 25% chance of being too high, and a 75% chance of being too low. Do the same for the 75th percentile. (Thus, you should give three estimates in all—low, median, and high—for each question.)

- a. Number of passengers who flew between New York and Los Angeles in 1989.
- b. Population of Warsaw in 1992.
- c. Year in which Coronado discovered the Mississippi River.
- d. Number of votes received by Jimmy Carter in the 1976 presidential election.
- e. Number of newspapers in the U.S. in 1990.
- f. Height of Hoover Dam in feet.
- g. Number of eggs produced in Oregon in 1985.
- h. Number of Buddhists in the world in 1992.
- i. Number of deaths due to AIDS in the U.S. in 1981.
- j. Number of U.S. patents granted in 1901.

The correct answers appear after the last exercise for this chapter. From the point of view of decision analysis, the interesting thing is not how close your median guesses came to the real answers, but rather how often the real answer came within your 25% and 75% bounds. If it was about half the time, then your bounds are accurate. But if you're like most people, you will be more sure of yourself than you should be, and fewer than half the answers will fall within the bounds. With practice, you can calibrate yourself to give realistic bounds, and thus be more useful in supplying information for decision making. Try this second set of questions and see if there is any improvement:

- a. Year of birth of Zsa Zsa Gabor.
- b. Maximum distance from Mars to the sun in miles.
- c. Value in dollars of exports of wheat from the U.S. in 1992.
- d. Tons handled by the port of Honolulu in 1991.
- e. Annual salary in dollars of the governor of California in 1993.

- f. Population of San Diego in 1990.
- g. Year in which Roger Williams founded Providence, R.I.
- h. Height of Mt. Kilimanjaro in feet.
- i. Length of the Brooklyn Bridge in feet.
- j. Number of deaths due to automobile accidents in the U.S. in 1992.

16.2 Tickets to the state lottery cost \$1. There are two possible prizes: a \$10 payoff with probability 1/50, and a \$1,000,000 payoff with probability 1/2,000,000. What is the expected monetary value of a lottery ticket? When (if ever) is it rational to buy a ticket? Be precise—show an equation involving utilities. You may assume that $U(\$10) = 10 \times U(\$1)$, but you may not make any assumptions about $U(\$1,000,000)$. Sociological studies show that people with lower income buy a disproportionate number of lottery tickets. Do you think this is because they are worse decision makers or because they have a different utility function?

16.3 Assess your own utility for different incremental amounts of money. Do this by running a series of preference tests between some definite amount M_1 and a lottery $[p, M_2; (1 - p), 0]$. Choose different values of M_1 and M_2 , and vary p until you are indifferent between the two choices. Plot the resulting utility function.



LEXICOGRAPHIC PREFERENCE

16.4 Write a computer program to automate the process in Exercise 16.3. Try your program out on several people of different net worth and political outlook. Comment on the consistency of your results, both across individuals and within the set of choices made by a single individual.

16.5 It has sometimes been suggested that **lexicographic preference** is a form of rational behavior that is not captured by utility theory. Lexicographic preferences rank attributes in some order X_1, \dots, X_n , and treat each attribute as infinitely more important than attributes later in the order. In choosing between two prizes, the value of attribute X_i only matters if the prizes have the same values for X_1, \dots, X_{i-1} . In a lottery, an infinitesimal probability of a tiny improvement in a more important attribute is considered better than a dead certainty of a huge improvement in a less important attribute. For example, in the airport-siting problem, it might be proposed that preserving human life is of paramount importance, and therefore if one site is more dangerous than another, it should be ruled out immediately, without considering the other attributes. Only if two sites are equally safe should they be compared on other attributes such as cost.

- a. Give a precise definition of lexicographic preference between deterministic outcomes.
- b. Give a precise definition of lexicographic preference between lotteries.
- c. Does lexicographic preference violate any of the axioms of utility theory? If so, give an example. (*Hint:* consider pair-wise preference comparisons of three different possibilities.)
- d. Suggest a set of attributes for which you might exhibit lexicographic preferences.

16.6 Show that if X_1 and X_2 are preferentially independent of X_3 , and X_2 and X_3 are preferentially independent of X_1 , then it follows that X_3 and X_1 are preferentially independent of X_2 .



16.7 Encode the airport-siting problem as shown in Figure 16.4, provide reasonable probabilities and utilities, and solve the problem for the case of choosing among three sites. What happens

if changes in technology mean that each aircraft generates half as much noise? What if noise avoidance becomes three times more important?

- 16.8** Repeat Exercise 16.7, using the action-utility representation shown in Figure 16.5.
- 16.9 For either of the airport-siting diagrams constructed in Exercises 16.7 and 16.8, to which conditional probability table entry is the utility most sensitive, given the available evidence?

- 16.10** (Adapted from Pearl (1988).) A used-car buyer can decide to carry out various tests with various costs (e.g., kick the tires, take the car to a qualified mechanic), and then, depending on the outcome of the tests, decide which car to buy. We will assume that the buyer is deciding whether to buy car c_1 , that there is time to carry out at most one test, and that t_1 is the test of c_1 and costs \$50.

A car can be in good shape (quality q^+) or bad shape (quality q^-), and the tests may help to indicate what shape the car is in. Car c_1 costs \$1,500, and its market value is \$2,000 if it is in good shape; if not, \$700 in repairs will be needed to make it in good shape. The buyer's estimate is that c_1 has a 70% chance of being in good shape.

- a. Calculate the expected net gain from buying c_1 , given no test.
- b. Tests can be described by the probability that the car will pass or fail given that the car is in good or bad shape. We have the following information:

$$P(\text{pass}(c_1, t_1) | q^+(c_1)) = 0.8$$

$$P(\text{pass}(c_1, t_1) | q^-(c_1)) = 0.35$$

Use Bayes' theorem to calculate the probability that the car will pass (or fail) its test, and hence the probability that it is in good (or bad) shape given each possible test outcome.

- c. Calculate the optimal decisions given either a pass or a fail, and their expected utilities.
- d. Calculate the value of information of the test, and derive an optimal conditional plan for the buyer.

- 16.11** Prove that the value of information is nonnegative, as stated in Section 16.6.

- 16.12** How much is a micromort worth to you? Devise a protocol to determine this.

The answers for Exercise 16.1 (where M stands for million): First set: 3M, 1.6M, 1541, 41M, 1611, 221, 649M, 295M, 132, 25546. Second set: 1917, 155M, 4500M, 11M, 120000, 1.1M, 1636, 19340, 1595, 41710.

17

MAKING COMPLEX
DECISIONS

In which we examine methods for deciding what to do today, given that we will have a chance to act again tomorrow.

In this chapter, we address the computational issues involved in making decisions. Whereas Chapter 16 was concerned with single decision problems, in which the utility of each action's outcome was well-known, in this chapter we will be concerned with **sequential decision problems**, where the agent's utility depends on a sequence of decisions. Sequential decision problems, which include utilities, uncertainty, and sensing, generalize the search and planning problems described in Parts II and IV.

The chapter divides roughly into two parts. Sections 17.1 through 17.3 deal with classical techniques from control theory, operations research, and decision analysis that were developed to solve sequential decision problems under uncertainty. They operate in much the same way that the search algorithms of Part II solved sequential decision problems in deterministic domains: by looking for a sequence of actions that leads to a good state. The difference is that what they return is not the fixed sequence of actions, but rather a **policy**—that is, a set of situation-action rules for each state—arrived at by calculating utilities for each state.

The second part, Sections 17.4 through 17.6, develops a complete sketch of a decision-theoretic agent using a richer representation of states in terms of random variables in a belief network. We also show how to efficiently update the network over time, and how to be able to safely forget things about the past.

17.1 SEQUENTIAL DECISION PROBLEMS

Suppose that an agent is situated in the environment shown in Figure 17.1. Beginning in the start state, it must execute a sequence of actions. The environment terminates when the agent reaches one of the states marked +1 or -1. In each location, the available actions are called *North*, *South*, *East*, and *West*. We will assume for now that the agent knows which state it is in initially, and that it knows the effects of all of its actions on the state of the world.

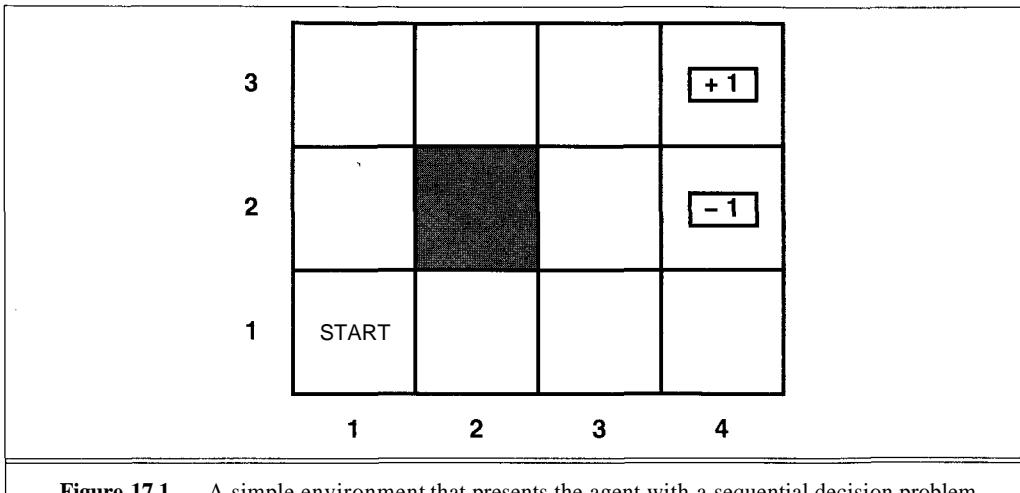


Figure 17.1 A simple environment that presents the agent with a sequential decision problem.

TRANSITION MODEL

In the deterministic version of the problem, each action reliably moves one square in the intended direction, except that moving into a wall results in no change in position. In the stochastic version, the actions are unreliable. Each action achieves the intended effect with probability 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction. For example, from the start square (1,1), the action *North* moves the agent to (1,2) with probability 0.8, but with probability 0.1, it moves *East* to (2,1), and with probability 0.1, it moves *West*, bumps into the wall, and stays in (1,1). We will use the term **transition model** (or just "model," where no confusion can arise) to refer to the set of probabilities associated with the possible transitions between states after any given action. The notation M_{ij}^a means the probability of reaching state j if action a is done in state i .

The tricky part is the utility function. Other than the terminal states (the ones marked +1 and -1), there is no indication of a state's utility. So we have to base the utility function on a sequence of states—an **environment history**—rather than on a single state. Let us suppose that the utility for a sequence will be the terminal state's value minus 1/25th the length of the sequence, so a sequence of length 6 that leads to the +1 box has utility 0.76.

In the deterministic case, with knowledge of the initial state and the effects of actions, the problem can be solved directly by the search algorithms described in Chapter 3. This is true *regardless of* whether the environment is accessible or inaccessible. The agent knows exactly which state it will be in after any given action, so there is no need for sensing.

In the more general, stochastic case, the agent will not know exactly which state it will reach after any given sequence of actions. For example, if the agent is in location (3,2), then the action sequence [*North*, *East*] might end up in any of five states (see Exercise 17.1), and reaches the +1 state at (4,3) with probability only 0.64.

One tempting way to deal with action sequences would be to consider sequences as long actions. Then one could simply apply the basic Maximum Expected Utility principle to sequences. The rational action would then be the first action of the optimal sequence. Now, although this approach is closely related to the way that search algorithms work, it has a fundamental flaw. It

assumes that the agent is required to *commit* to an entire sequence of actions before executing it. If the agent has no sensors, then this is the best it can do. But if the agent can acquire new sensory information after each action, then committing to an entire sequence is irrational. For example, consider the sequence *[North, East]*, starting at (3,2). With probability 0.1, *North* bumps the agent into the wall, leaving it still in (3,2). In this case, carrying on with the sequence and executing *East* would be a bad choice.

In reality, the agent will have the opportunity to choose a new action after each step, *given whatever additional information its sensors provide*. We therefore need an approach much more like the conditional planning algorithms of Chapter 13, rather than the search algorithms of Chapter 3. Of course, these will have to be extended to handle probabilities and utilities. We will also have to deal with the fact that the "conditional plan" for a stochastic environment may have to be of infinite size, because it is possible, although unlikely, for the agent to get stuck in one place (or in a loop) no matter how hard it tries not to.

We begin our analysis with the case of **accessible** environments. In an accessible environment, the agent's percept at each step will identify the state it is in. If it can calculate the optimal action for each state, then that will completely determine its behavior. No matter what the outcome of any action, the agent will always know what to do next.

A complete mapping from states to actions is called a **policy**. Given a policy, it is possible to calculate the expected utility of the possible environment histories generated by that policy. The problem, then, is not to calculate the optimal action sequence, but to calculate the optimal policy—that is, the policy that results in the highest expected utility. An optimal policy for the world in Figure 17.1 is shown in Figure 17.2(a). Notice that because the cost of taking a step is fairly small compared to the penalty for ending up in (4,2) by accident, the optimal policy for the state (3,1) is conservative. The policy recommends taking the long way round, rather than taking the short cut and thereby risking entering (4,2). As the cost of taking a step is increased, the optimal policy will, at some point, switch over to the more direct route (see Exercise 17.4). As the cost of a step is decreased, the policy will become *extremely* conservative. For example, if the cost is 0.01, the policy for the state (3,2) is to head *West* directly into the wall, thereby avoiding any chance of falling into (4,2).

Once a policy has been calculated from the transition model and the utility function, it is a trivial matter to decide what to do. A policy represents the agent function explicitly, and is therefore a description of a simple reflex agent, computed from the information used for a utility-based agent. Figure 17.3 shows the corresponding agent design.

The problem of calculating an optimal policy in an accessible, stochastic environment with a known transition model is called a **Markov decision problem** (MDP), after the Russian statistician Andrei A. Markov. Markov's work is so closely associated with the assumption of accessibility, that decision problems are often divided into "Markov" and "non-Markov." More strictly, we say the **Markov property** holds if the transition probabilities from any given state depend only on the state and not on previous history. The next two sections give algorithms for calculating optimal policies in Markov decision problems.

In an **inaccessible** environment, the percept does not provide enough information to determine the state or the associated transition probabilities. In the operations research literature, such problems are called **partially observable Markov decision problems**, or **POMDP**. Methods used for MDPs are not directly applicable to POMDPs. For example, suppose our agent

POLICY

MARKOV DECISION PROBLEM
MDP

MARKOV PROPERTY

POMDP

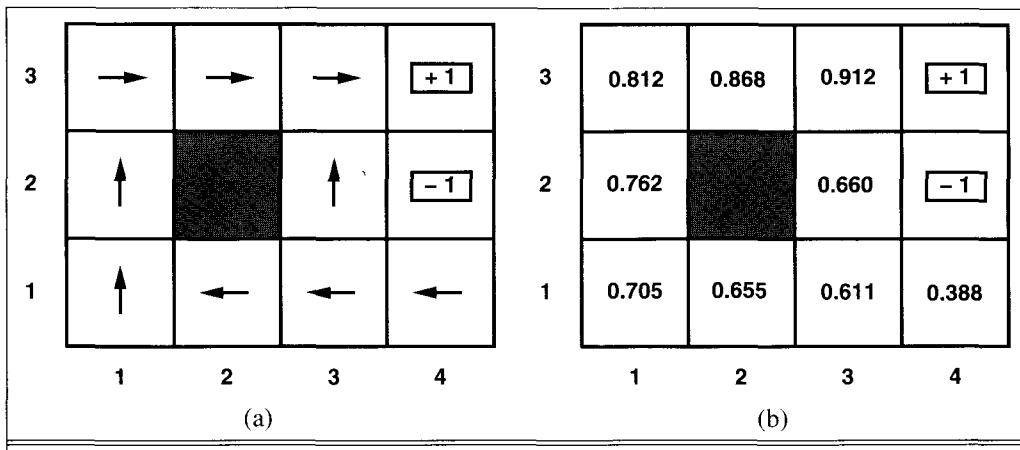


Figure 17.2 (a) An optimal policy for the stochastic environment. (b) The utilities of the states.

```

function SIMPLE-POLICY-AGENT(percept) returns an action
  static: M, a transition model
    U, a utility function on environment histories
    P, a policy, initially unknown

  if P is unknown then P *-- the optimal policy given U, M
  return P[percept]

```

Figure 17.3 An agent that calculates and uses an optimal policy.

is equipped with a sonar ring that gives it the distance to the nearest wall in each of the four directions. For such an agent, the locations (2,1) and (2,3) are indistinguishable, yet different actions are needed in each. Furthermore, the Markov property does not hold for percepts (as opposed to states), because the next percept does not depend just on the current *percept* and the action taken.

The correct approach for POMDPs is to calculate a probability distribution over the possible states given all previous percepts, and to base decisions on this distribution. Although the optimal decision is not uniquely determined by the current percept, it *is* determined uniquely (up to ties) by the agent's probability distribution over the possible states that it could be in. For example, the sonar-equipped agent might believe that it is in state (2,1) with probability 0.8 and in state (2,3) with probability 0.2. The utility of action A is then

$$0.8 \times \text{utility of doing A in (2,1)} + \\ 0.2 \times \text{utility of doing A in (2,3)}$$

This seems simple enough. Unfortunately, in POMDPs, calculating the utility of an action in a state is made more difficult by the fact that actions will cause the agent to obtain new percepts, which will cause the agent's beliefs to change in complex ways. Essentially, the agent must take

into account the *information* that it might obtain, as well as the state it will reach. POMDPs therefore include the value of information (Section 16.6) as one component of the decision problem.

The standard method for solving a POMDP is to construct a new MDP in which this probability distribution plays the role of the state variable. Unfortunately, the new MDP is not easy to solve. The new state space is characterized by real-valued probabilities, and is therefore infinite. Exact solution methods for POMDPs require some fairly advanced tools, and are beyond the scope of this book. The bibliographical notes at the end of this chapter provide pointers to suitable additional reading.

Instead of trying to find exact solutions, one can often obtain a good approximation using a limited lookahead. (See, for example, the algorithms in Chapter 5.) Section 17.4 shows how this approach can be realized for POMDPs using the technology of decision networks. Before tackling POMDPs, however, we first present the most common solution methods for making decisions in accessible worlds.

17.2 VALUE ITERATION

VALUE ITERATION

In this section, we present an algorithm for calculating an optimal policy called **value iteration**. The basic idea is to calculate the utility of each state, $U(state)$, and then use the state utilities to select an optimal action in each state.

The difficult part about calculating $U(state)$ is that we do not know where an action will lead. We can think of a sequence of actions as generating a tree of possible histories, with the current state as the root of the tree, and each path from the root to a leaf representing a possible history of states. We use the notation $H(state, policy)$ to denote the history tree starting from *state* and taking action according to *policy*. This can be thought of as a random variable that is dependent on the transition model *M*. Then the utility of a state *i* is given by the expected utility of the history beginning at that state and following an optimal policy:

$$U(i) = EU(H(i, policy^*) | M) = \sum P(H(i, policy^*) | M) U_h(H(i, policy^*)) \quad (17.1)$$

where *policy** is an optimal policy defined by the transition model *M* and the utility function on histories *U*/. We will explain shortly how to derive an optimal policy.

SEPARABILITY

Having a utility function on states is only useful to the extent that it can be used to make rational decisions, using the Maximum Expected Utility principle (Equation (16.1), page 472). In the case of sequential decisions, we have to be quite careful about this. For a utility function on states (*U*) to make sense, we require that the utility function on histories (*U*_h) have the property of **separability**. A utility function *U*_h is separable if and only if we can find a function *f* such that

$$U_h([s_0, s_1, \dots, s_n]) = f(s_0, U_h([s_1, \dots, s_n]))$$

ADDITIVE

(Exercise 17.2 asks you to construct a utility function violating this property.) The simplest form of separable utility function is **additive**:

$$U_h([s_0, s_1, \dots, s_n]) = R(s_0) + U_h([s_1, \dots, s_n])$$

REWARD FUNCTION

where R is called a **reward function**.¹ Consider again the utility function defined for Figure 17.1: an environment history of length n terminating in a state of value v has a utility of $v - (1/25)n$. This utility function is separable and additive, and the reward function R is $-1/25$ for nonterminal states, $+1$ for state $(4,3)$ and -1 for state $(4,2)$. As we discuss in what follows, utility functions over histories are almost always additive in practice. Notice that additivity was implicit in our use of path cost functions in heuristic search algorithms (Chapter 4).

Given an additive utility function U_h , we can recover the standard Maximum Expected Utility principle that an optimal action is one with maximal expected utility of outcome states:

$$\text{policy}^*(i) = \arg \max_a \sum_j M_{ij}^a U(j) \quad (17.2)$$

where M_{ij} is the probability of reaching state j if action a is taken in state i , and $\arg \max_a f(a)$ returns the value of a with the highest value for $f(a)$. Similarly, the utility of a state can be expressed in terms of the utility of its successors:

$$U(i) = R(i) + \max_a \sum_j M_{ji}^a U(j) \quad (17.3)$$

DYNAMIC PROGRAMMING

Equation (17.3) is the basis for **dynamic programming**, an approach to solving sequential decision problems developed in the late 1950s by Richard Bellman (1957).

The simplest dynamic programming context involves an n -step decision problem, where the states reached after n steps are considered terminal states and have known utilities. If there are $|A|$ possible actions at each step, then the total complexity of a naive approach—exhaustive enumeration—would be $O(|A|^n)$. The dynamic programming approach starts by calculating the utilities of all states at step $n - 1$ in terms of the utilities of the terminal states. One then calculates the utilities of states at step $n - 2$, and so on. Because calculating the utility of one state, using Equation (17.3), costs $O(|A|)$, the total cost of solving the decision problem is no more than $O(n|A||S|)$, where $|S|$ is the number of possible states. In small state spaces, this can be a huge saving.² Dynamic programming has since become a field of its own, with a huge array of applications and a large library of techniques for different types of separable utility functions.

In most of the decision problems that AI is interested in (including the world of Figure 17.1), the environment histories are potentially of unbounded length because of loops. This means that there is no n for which to start the n -step dynamic programming algorithm. Fortunately, there is a simple algorithm for approximating the utilities of states to any degree of accuracy using an iterative procedure. We apply Equation (17.3) repeatedly, on each step updating the utility of each state based on the old utility estimates of the neighboring states:

$$U_{t+1}(i) \leftarrow R(i) + \max_a \sum_j M_{ij}^a U_t(j) \quad (17.4)$$

where $U_t(i)$ is the utility estimate for state i after t iterations. As $t \rightarrow \infty$, the utility values will converge to stable values given certain conditions on the environment. The algorithm, called **VALUE-ITERATION**, is shown in Figure 17.4.

¹ Thus, the utility function is additive, in the sense defined in Chapter 16, given attributes corresponding to the rewards received in each state in the sequence. There are some problems in applying this equation to infinite histories, which will be discussed later.

² The saving is of the same sort as that achieved by checking repeated states during search (Section 3.6).

```

function VALUE-ITERATION( $M, R$ ) returns a utility function
  inputs:  $M$ , a transition model
     $R$ , a reward function on states
  local variables:  $U$ , utility function, initially identical to  $R$ 
     $U'$ , utility function, initially identical to  $R$ 

  repeat
     $U \leftarrow U'$ 
    for each state  $i$  do
       $U'[i] \leftarrow R[i] + \max_a \sum_j M_{ij}^a U[j]$ 
    end
  until CLOSE-ENOUGH( $U, U'$ )
  return  $U$ 

```

Figure 17.4 The value iteration algorithm for calculating utilities of states.

Given a utility function on states, it is trivial to calculate a corresponding policy using Equation (17.2). Furthermore, the policy will actually be optimal, as proved by Bellman and Dreyfus (1962). We can apply value iteration to the environment shown in Figure 17.1, which yields the utility values shown in Figure 17.2(b). In Figure 17.5, we show the utility values of some of the states at each iteration step of the algorithm. Notice how the states at different distances from (4,3) accumulate negative reward until, at some point, a path is found to (4,3) whereupon the utilities start to increase.

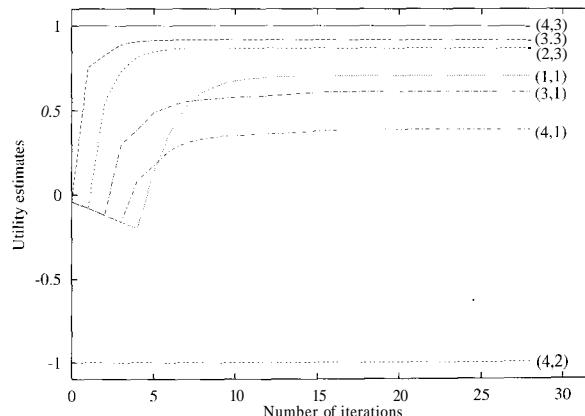


Figure 17.5 The utility values for selected states at each iteration step in the application of VALUE-ITERATION to the 4x3 world in Figure 17.1.

RMS ERROR

POLICY LOSS

How long should value iteration be allowed to run? Do we require the values to converge? These are nontrivial questions. There are two obvious ways to measure the progress of value iteration. The first uses the **RMS error** (RMS stands for "root mean square") of the utility values compared to the correct values. The second assumes that the estimated utility values are not in themselves important—what counts is the policy that they imply. The policy corresponding to an estimated utility function U , is derived using Equation (17.2). We can measure the quality of a policy using the expected **policy loss**—the difference between the expected utility obtained by an agent using the policy, compared with an agent using the optimal policy. Figure 17.6 shows how both measures approach zero as the value iteration process proceeds. Notice that the policy (which is chosen from a discrete, finite set of possible policies) becomes exactly optimal long before the utility estimates have converged to their correct values.

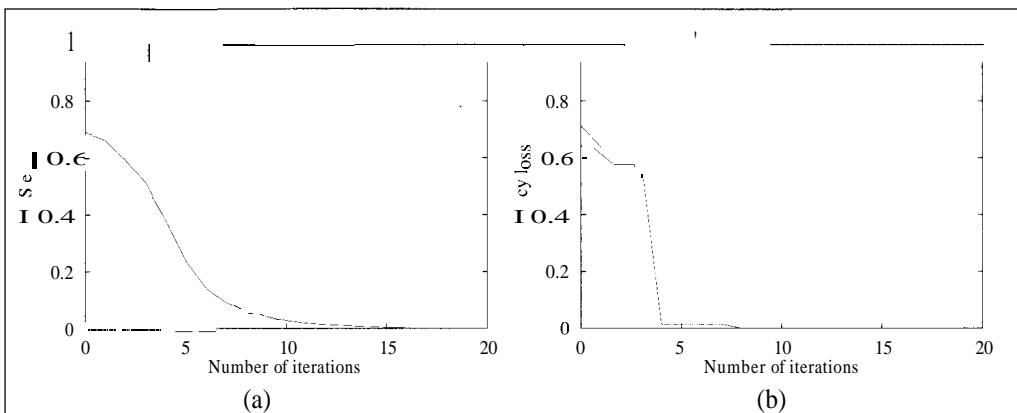


Figure 17.6 (a) The RMS (root mean square) error of the utility estimates compared to the correct values, as a function of iteration number during value iteration. (b) The expected policy loss compared to the optimal policy.

17.3 POLICY ITERATION

POLICY ITERATION

VALUE DETERMINATION

In the previous section, we observed that the optimal policy is often not very sensitive to the exact utility values. This insight suggests an alternative way to find optimal policies. The **policy iteration** algorithm works by picking a policy, then calculating the utility of each state *given that policy*. It then updates the policy at each state using the utilities of the successor states (Equation (17.2)), and repeats until the policy stabilizes. The step in which utility values are determined from a given policy is called **value determination**. The basic idea behind policy iteration, as compared to value iteration, is that value determination should be simpler than value iteration because the action in each state is fixed by the policy. The policy iteration algorithm is shown in Figure 17.7.

```

function POLICY-ITERATION( $M, R$ ) returns a policy
  inputs:  $M$ , a transition model
     $R$ , a reward function on states
  local variables:  $U$ , a utility function, initially identical to  $R$ 
     $P$ , a policy, initially optimal with respect to  $U$ 

  repeat
     $U \leftarrow \text{VALUE-DETERMINATION}(P, U, M, R)$ 
     $\text{unchanged?} \leftarrow \text{true}$ 
    for each state  $i$  do
      if  $\max_a \sum_j M_{ij}^a U[j] > \sum_j M_{ij}^{P[i]} U[j]$  then
         $P[i] \leftarrow \arg \max_a \sum_j M_{ij}^a U[j]$ 
       $\text{unchanged?} \leftarrow \text{false}$ 
    end
  until  $\text{unchanged?}$ 
  return  $P$ 

```

Figure 17.7 The policy iteration algorithm for calculating an optimal policy.

The VALUE-DETERMINATION algorithm can be implemented in one of two ways. The first is a simplification of the VALUE-ITERATION algorithm, replacing Equation (17.4) with

$$U_{t+1}(i) \leftarrow R(i) + \sum_j M_{ij}^{Policy(i)} U_t(j)$$

and using the current utility estimates from policy iteration as the initial values. (Here $Policy(i)$ is the action suggested by the policy in state i .) While this can work well in some environments, it will often take a very long time to converge in the early stages of policy iteration. This is because the policy will be more or less random, so that many steps can be required to reach terminal states.

The second approach is to solve for the utilities directly. Given a fixed policy P , the utilities of states obey a set of equations of the form

$$U(i) = R(i) + \sum_j M_{ij}^{P(i)} U_t(j)$$

For example, suppose P is the policy shown in Figure 17.2(a). Then using the transition model M , we can construct the following set of equations:

$$\begin{aligned} u_{(1,1)} &= 0.8u_{(1,2)} + 0.1u_{(1,1)} + 0.1u_{(2,1)} \\ u_{(1,2)} &= 0.8u_{(1,3)} + 0.2u_{(1,2)} \end{aligned}$$

and so on. This gives a set of 11 linear equations in 11 unknowns, which can be solved by linear algebra methods such as Gaussian elimination. For small state spaces, value determination using exact solution methods is often the most efficient approach.

HOW IMMORTAL AGENTS DECIDE WHAT TO DO

Making decisions is not easy when one lives forever. The total reward obtained by a policy can easily be unbounded. Because one cannot easily compare infinities, it is difficult to say which policy is rational; moreover, both value iteration and policy iteration will fail to terminate. If the agent's lifetime is finite but contains millions of steps, then these algorithms are intractable. These difficulties arise from fundamental problems associated with specifying utilities over histories so that the resulting "optimal" behavior makes intuitive sense. The same issues arise for humans. Should one live fast and die young, or live an unexciting life to a ripe old age?

One of the most common approaches is to use **discounting**. A discounting function considers rewards received in future time steps to be less valuable than rewards received in the current time step. Suppose that an environment history H contains a stream of rewards, such that the agent receives reward R_i at the i th future time step. The standard method of discounting uses the utility function $U(H) = \sum_i \gamma^i R_i$, where γ is the **discount factor**. Provided $0 < \gamma < 1$, this sum will converge to a finite amount. Discounting can be interpreted in at least three different ways:

- As a trick to get rid of the infinities. Essentially, it is a smoothed-out version of the limited-horizon algorithms used in game-playing—the smaller the value of γ , the shorter the effective horizon.
- As an accurate model of both animal and human preference behavior. In economics, it is widely used in assessing the value of investments.
- As a natural preference-independence assumption associated with rewards over time. Discounting follows from an assumption of **stationarity**. Stationarity means the following: if two reward sequences R_1, R_2, R_3, \dots and S_1, S_2, S_3, \dots begin with the same reward (i.e., $R_1 = S_1$) then the two sequences should be preference-ordered the same way as the sequences R_2, R_3, \dots and S_2, S_3, \dots . If stationarity seems like a reasonable assumption, then discounting is a reasonable way to make decisions.

If the agent's lifetime is long (in number of steps) compared to the number of states, then the optimal policy will be *repetitive*. For example, a taxi driver aiming to maximize his income will usually adopt a standard set of waiting patterns for times when he does not have a passenger. The **system gain** is defined as the average reward obtained per unit time. It can be shown that after an initial "transient" period, any optimal policy has a constant system gain. This fact can be used to compute optimal policies—for example, telling the taxi driver where to wait at different times of day—using a version of policy iteration.

17.4 DECISION-THEORETIC AGENT DESIGN

In this section, we outline a comprehensive approach to agent design for environments with uncertainty. It ties together belief and decision networks with the techniques for sequential decision problems discussed earlier. It addresses the problem of large state spaces by decomposing the state description into a set of random variables, much as the planning algorithms in Part IV used logical representations to decompose the state space used by search algorithms. We begin by describing the basic approach, which harks back to the sketch of the **utility-based agent** provided in Chapter 2. We then show how sensing works in an uncertain, partially accessible environment. Section 17.5 extends the idea of belief networks to cover environments that change over time, and then Section 17.6 includes decisions, providing a complete agent design.

The decision cycle of a rational agent

Figure 17.8 repeats the schematic agent design for rational agents first shown in Figure 14.1. At each step, the processing done by the agent is called the **decision cycle**. In this section, we will make components of the cycle more precise. We begin with the first step, that of determining the current state of the world.

```
function DECISION-THEORETIC-AGENT(percept)returns action
    calculate updated probabilities for current state based on
        available evidence including current percept and previous action
    calculate outcome probabilities for actions
        given action descriptions and probabilities of current states
    select action with highest expected utility
        given probabilities of outcomes and utility information
    return action
```

Figure 17.8 A decision-theoretic agent (repeat of earlier figure).

In general, we will assume that we have a set of random variables X , that refer to the current state of the world. We will call these the **state variables**. For example, if the agent is a robot moving in the X - Y plane, then we might use X_t and Y_t to refer to the robot's position at time t , and X_v and Y_v to refer to the velocity. Notice the similarity to the propositional version of situation calculus used in Chapter 6 for the first logical-agent design. This similarity is not a coincidence: probability theory essentially combines propositional logic with uncertainty. As in situation calculus, it is important to distinguish between beliefs about a changing world and changing beliefs about a given world. The former is achieved by having different propositions referring to different times, and the latter by conditioning the probability of a given proposition on additional evidence. Thus, if the percept history up to and including time t is E_1, \dots, E_t ,

(where each \mathbf{E}_i may also consist of observations on several random variables), and the previous actions have been $A_1 \dots A_{t-1}$, then what we are interested in is

$$\mathbf{P}(\mathbf{X}_t | \mathbf{E}_1 \dots \mathbf{E}_t, A_1 \dots A_{t-1}),$$

that is, the probability distribution over the current state given all available evidence. We refer to this quantity as $Bel(\mathbf{X}_t)$ —the belief about the state at time t , given all evidence up to time t .

This is a rather complicated expression, and direct evaluation is out of the question because it requires conditioning on a large number of variables. As in Chapter 14, we can use conditional independence statements to simplify this expression. The main assumption is that the problem is *Markovian*—the probability distribution for the current state of the world depends only on the previous state and the action taken in it. If X_t is the state of the world at time t and A_t is the action taken at time t , then we have

$$\mathbf{P}(\mathbf{X}_t | \mathbf{X}_1 \dots \mathbf{X}_{t-1}, A_1 \dots A_{t-1}) = \mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1}, A_{t-1}) \quad (17.5)$$

Whether the Markov property holds depends on which state variables the agent is tracking, and on the details of the environment. For example, in the case of a robot moving in the X-Y plane, the previous position and velocity might well be enough to predict the current position and velocity, given the previous motor command—one can simply use Newton's laws to calculate the new position and velocity. On the other hand, if the robot is battery-powered, then the effect of an action will depend on whether the battery is exhausted. Because this in turn depends on how much power was used by all previous commands, the Markov property is violated. We can restore the Markov property by including $BatteryLevel_t$ as one of the state variables that comprise X_t .

We also assume that each percept depends only on the state at the time. This amounts to the assertion that percepts (E_t) are causally determined by the state of the world:

$$\mathbf{P}(\mathbf{E}_t | \mathbf{X}_1 \dots \mathbf{X}_t, A_1 \dots A_{t-1}, E, \dots E, \dots) = \mathbf{P}(\mathbf{E}_t | \mathbf{X}_t) \quad (17.6)$$

Finally, we assume that a similar equation holds for actions. The action taken depends only on the percepts the agent has received to date:

$$\mathbf{P}(A_{t-1} | A_1 \dots A_{t-2}, \mathbf{E}_1 \dots \mathbf{E}_{t-1}) = \mathbf{P}(A_{t-1} | \mathbf{E}_1 \dots \mathbf{E}_{t-1}) \quad (17.7)$$

This final assertion is valid because of the structure of the agent itself: its only input from the outside is the percept at each time step.

Taken together, Equations (17.5), (17.6), and (17.7) allow us to simplify the calculation of the current state estimate $Bel(\mathbf{X}_t)$. The calculation takes place in two phases:

PREDICTION PHASE

- ◊ **Prediction phase:** first, we predict the probability distribution over states we *would have expected*, given our knowledge of the previous state and how actions affect states. We call this \widehat{Bel} , and calculate it by adding up the probabilities of arriving in a given state at time t for each of the states we could have been in at time $t-1$:

$$\widehat{Bel}(\mathbf{X}_t) = \sum_{\mathbf{X}_{t-1}} \mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1} = \mathbf{x}_{t-1}, A_{t-1}) Bel(\mathbf{X}_{t-1} = \mathbf{x}_{t-1}) \quad (17.8)$$

where \mathbf{x}_{t-1} ranges over all possible values of the state variables \mathbf{X}_{t-1} .

ESTIMATION PHASE

- 0 **Estimation phase:** now we have a distribution over the current state variables, given everything but the most recent observation. The estimation phase updates this using the

percept E_t . Because both the state variables and the percept refer to the same time, this is a simple matter of Bayesian updating, using $\widehat{Bel}(\mathbf{X}_t)$ as the prior:

$$Bel(\mathbf{X}_t) = \alpha \mathbf{P}(\mathbf{E}_t | \mathbf{X}_t) \widehat{Bel}(\mathbf{X}_t) \quad (17.9)$$

where α is a normalization constant.

Exercise 17.5 asks you to derive these equations from the assumptions listed earlier.

KALMAN FILTERING

It is worth noting that the equations for Bel and \widehat{Bel} are a generalization of the technique known in classical control theory as **Kalman filtering** (Kalman, 1960). Kalman filtering assumes that each state variable is real-valued and distributed according to a Gaussian distribution; that each sensor suffers from unbiased Gaussian noise; that each action can be described as a vector of real values, one for each state variable; and that the new state is a linear function of the previous state and the action. These assumptions, taken together, allow prediction and estimation to be implemented by some simple matrix calculations, even with a large number of state variables. Kalman filtering is universally applied in monitoring and controlling all sorts of dynamical systems, from chemical plants to guided missiles. It has good success even in domains where not all the assumptions are satisfied.

SENSOR MODEL ACTION MODEL

Given a probability distribution over the current state, it is a simple matter to carry out the remaining steps of the decision cycle, which involve projecting forward the possible results of the available actions and choosing the one with maximal expected utility. The belief update equations also allow us to design an agent that keeps around just the current belief vector for the state variables. The complete design is shown in Figure 17.9. Although the formulas look quite complicated, bear in mind that they simply instantiate the basic design given in Figure 17.8. Furthermore, the conditional probabilities appearing in the various expressions are exactly what we would expect to see. We have $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$, the **sensor model**, which describes how the environment generates the sensor data; and we have $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1}, A_{t-1})$, the **action model**, which describes the effects of actions (and similarly for $t, t+1$).³ These are the same kinds of information that we have used throughout the book to make decisions. The action model generalizes the transition model used earlier for sequential decision problems. The sensor model was not used there, of course, because we assumed an accessible environment in which the percept and the state can be equated.

The following sections describe sensor and action models in more detail, and show how the complete agent design can be implemented directly using the belief and decision network technology described in the previous chapters.

STATIONARY SENSOR MODEL

Sensing in uncertain worlds

We begin with the sensor model, which we defined previously as $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$, the probability of a percept given a state of the world. Actually, this is unnecessarily complicated, because it allows the sensor model to vary with time. We will instead assume a **stationary sensor model**:

$$\forall t \quad \mathbf{P}(\mathbf{E}_t | \mathbf{X}_t) = \mathbf{P}(\mathbf{E} | \mathbf{X})$$

³ The notation used to describe the action model might make it look as if it only allows for actions by the agent. In fact, because the actions of other agents are presumably themselves determined by the state variables \mathbf{X}_{t-1} , the model can certainly handle multiple agents as is.

```

function DECISION-THEORETIC-AGENT( $E_t$ ) returns an action
  inputs:  $E_t$ , the percept at time  $t$ 
  static:  $BN$ , a belief network with nodes  $X$ 
     $Bel(X)$ , a vector of probabilities, updated over time
   $\widehat{Bel}(X_t) \leftarrow \sum_{X_{t-1}} P(X_t | X_{t-1} = x_{t-1}, A_{t-1}) Bel(X_{t-1} = x_{t-1})$ 
   $Bel(X_t) = a P(E_t | P(X_t) \widehat{Bel}(X_t))$ 
   $action = \arg \max_{A_t} \sum_{X_t} \left[ Bel(X_t = x_t) \sum_{X_{t+1}} P(X_{t+1} = x_{t+1} | X_t = x_t, A_t) U(x_{t+1}) \right]$ 
  return  $action$ 

```

Figure 17.9 Detailed design for a decision-theoretic agent.

where E and X are random variables ranging over percepts and states, respectively. What this means is that given a state of the world, the chances of the sensor giving a certain reading will be the same today as it was yesterday. This does *not* mean, for example, that the sensor can never break; it just means that we have to include in X all the variables that are important to the sensor's performance. The advantage of the stationary sensor model is that the fixed model $P(E|X)$ can then be used at each time step.

The basic idea of a sensor model can be implemented very simply in a belief network, because the assumption embodied in Equation (17.6) effectively isolates the sensor variables for a given time step from the rest of the network. Figure 17.10(a) shows an abstract belief network fragment with generalized state and sensor variables. The sensor model itself is the conditional probability table associated with the percept node. The direction of the arrow is the crucial element here: the state of the world *causes* the sensor to take on a particular value.⁴ The sensor model is therefore an example of the general principle, stated in Chapter 7 and again in Chapter 15, that causal models are to be preferred when possible. If the sensor gives a perfect report of the actual state, then the sensor model—the conditional probability table—will be purely deterministic. Noise and errors in the sensor are reflected in the probabilities of "incorrect" readings. In fact, we have already seen examples of sensor models of this kind: in the burglar-alarm network (Figure 15.1), both *JohnCalls* and *MaryCalls* can be viewed as sensor nodes for the *Alarm* state variable. Their conditional probability tables, shown in Figure 15.2, show how reliable they are as sensors.

The next step is to break apart the generalized state and sensor variables into their components. Typically, each sensor only measures some small aspect of the total state. An example is shown in Figure 17.10(b), where temperature and pressure gauges measure the actual temperature and pressure of some system. Decomposing the overall sensor model in Figure 17.10(a) into its separate components greatly reduces the size of the CPTs, unless the sensors are very badly designed. As an example of how not to couple sensor nodes to state nodes: consider two sensors

⁴ Of course, the process of *inference* will go the other way: evidence arrives at the sensor node, and is propagated to the state variable. The inference process essentially *inverts* the sensor model. This is basically what makes a Kalman filter simple. If $P(E|X)$ is Gaussian, then $P(X|E)$ is also Gaussian with an identical distribution, so that inversion is trivial.

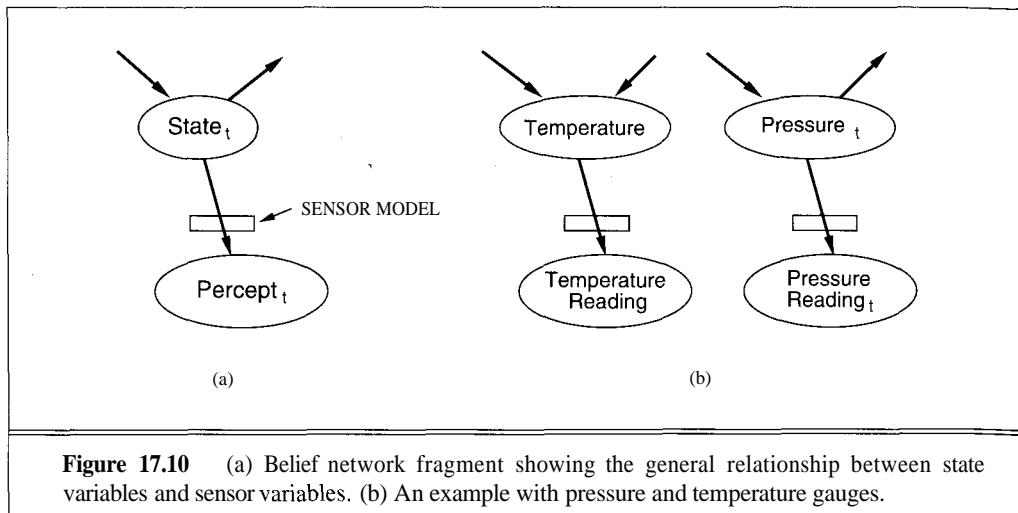


Figure 17.10 (a) Belief network fragment showing the general relationship between state variables and sensor variables. (b) An example with pressure and temperature gauges.

that (somehow) measure *Pressure/Temperature* and *Pressure x Temperature*. Each of the two state nodes would have to be connected to both sensor nodes, resulting in large CPTs for the sensor models.

Often, we will have several sensors that are measuring the same state variable. In Figure 17.11, we show an example where two gauges are being used to measure the actual temperature of some object, perhaps a superconductor. *The crucial thing to notice here is that the sensor values are conditionally independent of each other, given the actual value.* The reasoning here is similar to the reasoning for the conditional independence of symptoms given a disease. Although the sensors are not unconditionally independent—in fact, they will usually display approximately the same reading—they are correlated only inasmuch as they depend on the actual temperature. When we have multiple sensors for the same state variables, the resulting inference process is called **sensor fusion** or **data fusion**. To see how important this can be, consider the situation when Gauge 1 reads 13.6°K, while Gauge 2 reads 14.4°K. If each gauge is accurate to within 0.5°K, as represented in the sensor models, then the network will infer that the actual temperature is between 13.9°K and 14.1°K. Integrating the results of multiple sensors can provide greater accuracy than any one sensor on its own. This is true whether the sensors are similar, as in the case of temperature gauges, or very different, as in the case of sonar and infrared distance sensors used in robotics. Detailed sensor models have been built for both sonar and infrared, which have somewhat complementary performance. Sonar has a long range, but is subject to "ghost" images caused by multiple reflections and specularities. Infrared is only accurate over short distances. By using sensor fusion, it is often possible to create an accurate model of the robot's environment in cases where the sensors used separately would be lost.

Anyone with hands-on experience of robotics, computerized process control, or other forms of automatic sensing will readily testify to the fact that sensors fail. When a sensor fails, it does not necessarily send a signal saying, "Oh, by the way, the data I'm about to send you is a load of nonsense." Instead, it simply sends the nonsense. This can be dangerous if taken literally. For example, a robot's sonar distance sensor might start sending "infinity," meaning that no object



SENSOR FUSION
DATA FUSION

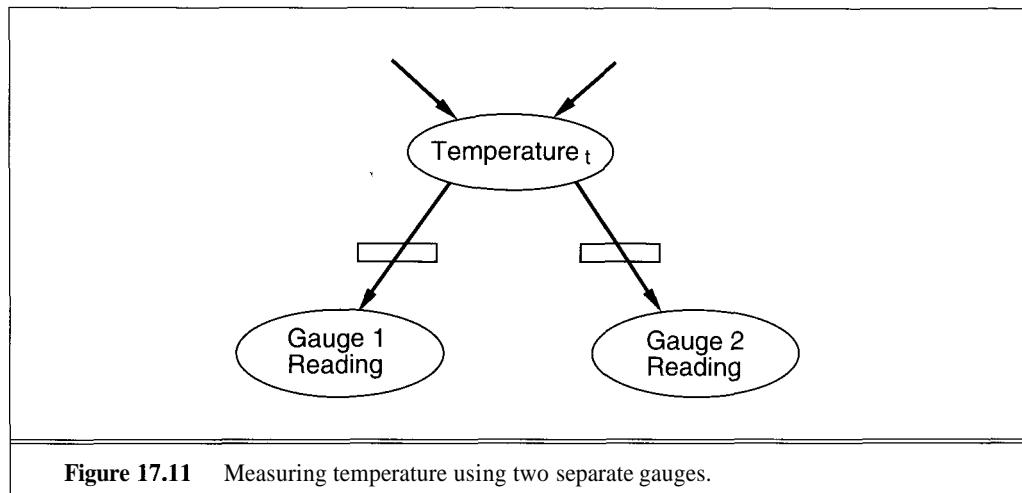
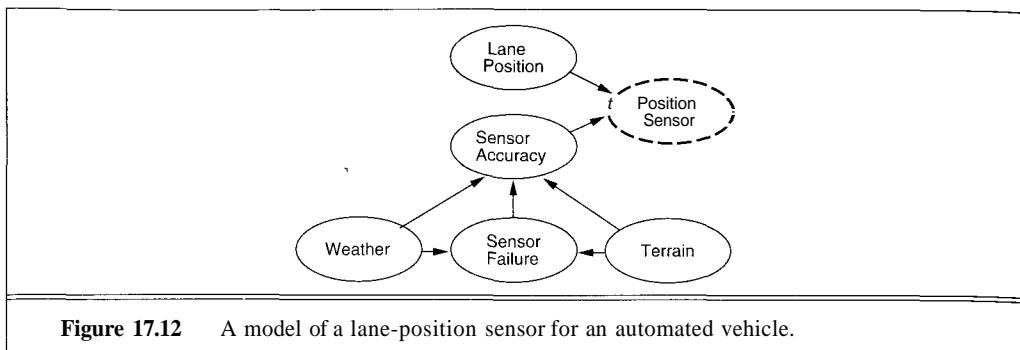


Figure 17.11 Measuring temperature using two separate gauges.

was detected within the sonar's range. This could be because the robot has wandered outside or it could be because the sonar's detector is broken. In the latter case, the robot could start crashing into walls. *In order for the system to handle sensor failure, the sensor model must include the possibility of failure.* For example, in the case of sonar, a sensor model that says that the sensor is accurate to within 10 cm explicitly *disallows* the possibility of failure, and therefore forces the robot to take the sonar reading literally. For any given *actual* distance, the sonar model should allow the possibility that the *observed* distance will be "infinity." Then the robot can handle sensor failure more appropriately. For example, if the robot is in a corridor, its *prediction* will be that the closest object remains about 60 cm away. If the sonar suddenly reports "infinity," then the most likely conclusion is that the sensor has failed, not that the corridor has disappeared. Furthermore, if the robot has more than one distance sensor, the sensor fusion process will automatically discount the readings of the failed sensor.

It is also possible to use more detailed models of sensor failure by incorporating additional state variables representing the condition of the sensor. Figure 17.12 shows a model of a vision-based lane-position sensor. Such sensors are used in autonomous vehicles to keep them in the center of their lane. They also could be used to sound a warning in a human-driven car when it starts to stray off the road. The sensor's accuracy is directly affected by rain and an uneven road surface. Furthermore, rain might also cause the sensor to fail by damaging the electronics, as might a bumpy road. Sensor failure in turn affects the sensor's accuracy. This kind of model is capable of some quite subtle reasoning. For example, if the system believes (perhaps from another sensor's input) that it is raining, then that will alter the sensor accuracy variable, raising the likelihood of larger error in the lane-position sensor. When an unexpected reading occurs, the system will be less likely to assume that the car is out of position. Conversely, a large discrepancy between expected and observed position can increase the system's belief that it is raining! A really serious discrepancy would raise the posterior probability of sensor failure; hence this kind of network can perform "diagnosis" of the sensors. In the next section, we will see how this capability can be extended by reasoning over time.



17.5 DYNAMIC BELIEF NETWORKS

We now consider the evolution of the state of the environment over time, and how this can be represented in a dynamic belief network. As we said earlier, the evolution of the environment is modelled by the conditional probability distribution $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1}, A_{t-1})$, which describes how the state depends on the previous state and the action of the agent. As with the sensor model, we make a stationarity assumption: the conditional probabilities are the same for all t . In this section, we will cover the case where the agent is passively monitoring and predicting a changing environment, rather than acting on it. The agent is thus concerned with a sequence of \mathbf{X} , values, where each one is determined solely by the previous one: $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1})$. This sequence is called a **state evolution model** or **Markov chain**. Monitoring and prediction is important in its own right, and it also makes the explanation simpler. In the next section, we will show how an agent can use the \mathbf{X} , to make decisions and take action.

In principle, we want to build a belief network with one node for each state and sensor variable, for each time step. A network of this kind is called a **dynamic belief network** (DBN). The generic structure of a DBN is shown in Figure 17.13. (In a real network for a specific problem, the state and percept nodes would be replaced by several nodes each, with appropriate connections. Notice also the resemblance to Figure 7.3.) If t is the current time step, then we have evidence for the percept nodes up to and including time t . The task of the network is then to calculate the probability distribution for the state at time t . One may also want to know how the state will evolve into the future—the probability distributions for $State_{t+1}$ and so on. This task is called **probabilistic projection**. Both tasks can be carried out using the standard algorithms from Chapter 15.

A little thought reveals that although the previous sentence is true, it may not be very useful. A dynamic belief network of the kind shown in Figure 17.13 could be extremely large, so the belief net algorithms could be extremely inefficient. Now we will see the benefit of all the work that went into Equations (17.8) and (17.9) (for Bel and Bel). We can implement the prediction and estimation phases as operations on the belief network. Furthermore, we need only

STATE EVOLUTION
MODEL
MARKOV CHAIN

DYNAMIC BELIEF
NETWORK

PROBABILISTIC
PROJECTION

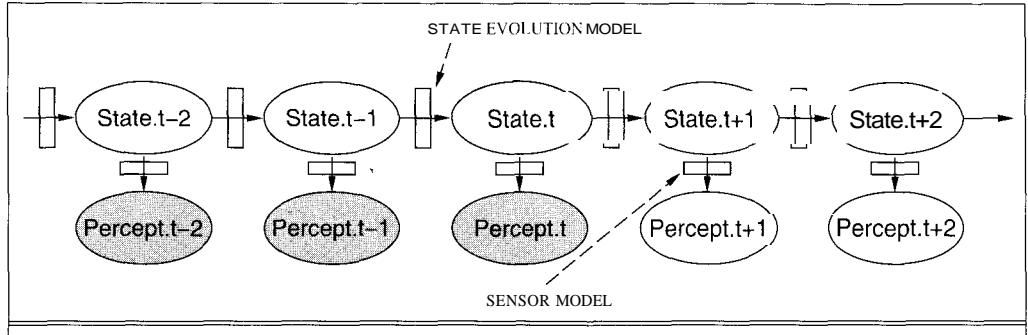


Figure 17.13 The generic structure of a dynamic belief network. The shaded nodes show the evidence that has been accumulated so far.

SLICES

keep enough network structure to represent *two* time steps (otherwise known as two **slices** of the network). Figure 17.14 shows the prediction-estimation process in operation, a truly beautiful thing. Each cycle of the process works as follows:

ROLLUP

- ◊ **Prediction:** we begin with a two-slice network; let us call the slices $t - 1$ and t . We assume that we have already calculated $\text{Bel}(\mathbf{X}_{t-1})$, incorporating all evidence up to and including E_{t-1} . Notice that slice $t - 1$ has no connections to previous slices. The state variables in $t - 1$ have prior probabilities associated with them (see the next step). We then calculate the belief vector $\widehat{\text{Bel}}(\mathbf{X}_t)$, according to Equation (17.8). This is actually the standard belief network updating process applied to evidence E_{t-1} .
- ◊ **Rollup:** now we *remove* slice $t - 1$. This requires adding a prior probability table for the state variables at time t . This prior is just $\widehat{\text{Bel}}(\mathbf{X}_t)$.
- ◊ **Estimation:** now we add the new percept E_t , applying standard belief network updating to calculate $\text{Bel}(\mathbf{X}_t)$, the probability distribution over the current state. We then add the slice for $t + 1$. The network is now ready for the next cycle.

This process implements the formal algorithm specified in Figure 17.9, using the belief network inference machinery for all the calculations. Notice that, as in the formal algorithm, the percept history is summarized in the belief vector for the current state—a summarization justified by Equation (17.5).

Probabilistic projection is also straightforward. We can take the network after step (c), add slices for future times, and apply a belief network inference algorithm to calculate the posterior probability distributions for the future states, given the current percept. Unlike the update cycle, this might be expensive because it involves inference in a temporally extended network. However, this network has a special property: none of the future nodes has any evidence associated with it. This means that a simple stochastic simulation technique such as logic sampling (see page 455) will work well, because every run can be consistent with the evidence. Given a desired accuracy level for the sampling process, the time complexity will usually be $O(n)$.

As an example of the application of dynamic belief networks, consider again the sensor-failure model shown in Figure 17.12. We can extend this into a DBN (Figure 17.15) by adding state evolution models for state variables *Weather*, *Terrain* and *SensorFailure*, as well as for the

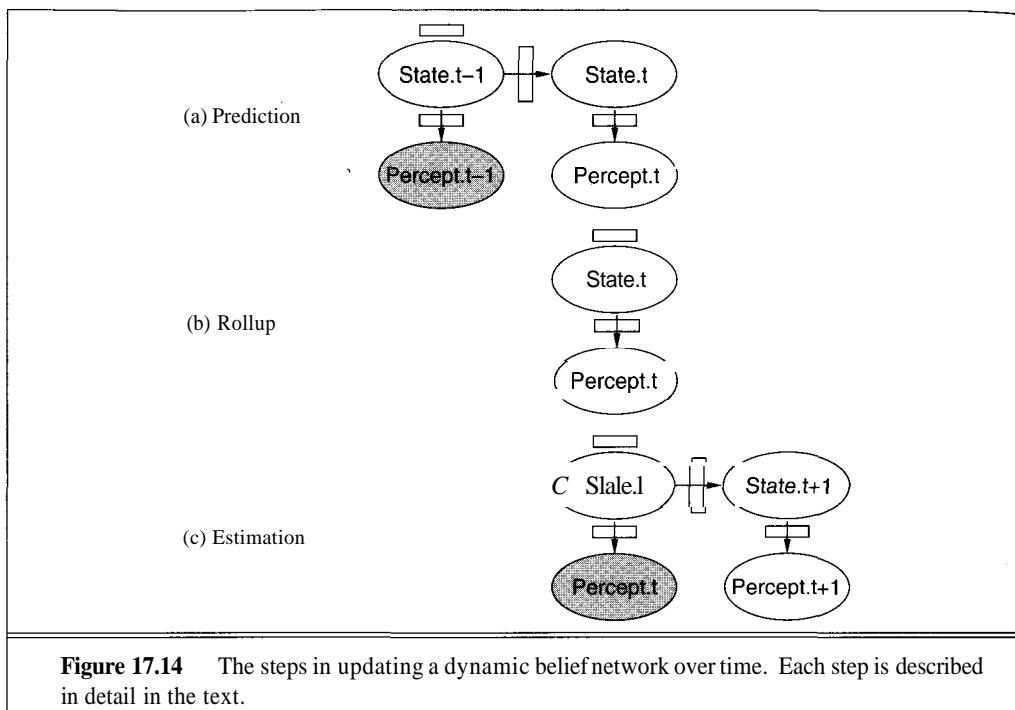


Figure 17.14 The steps in updating a dynamic belief network over time. Each step is described in detail in the text.

principal state variable *LanePosition*. The model of interest here is that for *SensorFailure*. The model is quite simple: basically, once a sensor has broken, it usually stays broken. What happens over time is that as the sensor continues to send nonsense signals, it becomes more and more likely that they are incorrect. This is especially true if there are other sensors through which the network can infer *LanePosition* indirectly. It will even work, however, just using the state evolution model for *LanePosition*, which will usually put limits on how much lateral motion we can expect for a vehicle.

17.6 DYNAMIC DECISION NETWORKS

DYNAMIC DECISION NETWORKS

All we need in order to convert dynamic belief networks into **dynamic decision networks** (DDNs) is to add utility nodes and decision nodes for actions. Figure 17.16 shows the generic structure of a DDN for a sequential decision problem where the terminal states are three steps ahead. The decision problem involves calculating the value of D , that maximizes the agent's expected utility over the remaining state sequence.⁵ In addition to the decision nodes for the

⁵ Usually, the final utility will be calculated as a sum of expected rewards $R_t + R_{t+1} + \dots$. We omit the reward nodes in order to simplify the diagram.

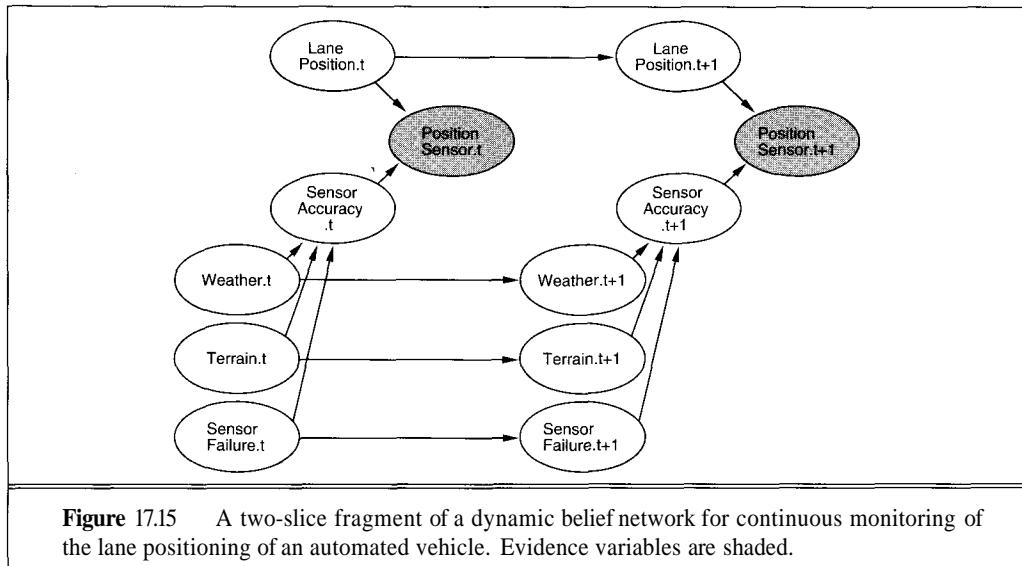


Figure 17.15 A two-slice fragment of a dynamic belief network for continuous monitoring of the lane positioning of an automated vehicle. Evidence variables are shaded.

current and future time steps, notice that the network also contains the previous decision, D_{t-1} , as an evidence node. It is treated as evidence because it has already happened.

The evaluation algorithm for DDNs is essentially the same as that for ordinary decision networks. In the worst case, the DDN calculates the expected utility of each decision sequence by fixing the decision nodes and applying probabilistic inference to calculate the final state. As in our discussion of sequential decision problems earlier in the chapter, we must also be careful to take into account the fact that, for each future decision, the agent does not currently know what information will be available at the time the future decision is made. That is, for decision D_{t+i} , the agent *will* have available percepts E_{t+1}, \dots, E_{t+i} ; but currently, it does not know what those percepts will be. For example, an autonomous vehicle might be contemplating a lane change at time $t + i$, but it will not know until then if there is another car blocking its path.

In our earlier discussion, we handled this by iteratively computing a **policy** that associates a decision with each *state*. With DDNs, we do not have this option because the states are represented implicitly by the set of state variables. Furthermore, in inaccessible environments, the agent will not know what state it is in anyway. What we must do instead is consider each possible instantiation of the future sensor variables as well as each possible instantiation of the future decision variables. The expected utility of each decision sequence is then the weighted sum of the utilities computed using each possible percept sequence, where the weight is the probability of the percept sequence given the decision sequence. Thus, the DDN provides approximate solutions for partially observable Markov decision problems, where the degree of approximation depends on the amount of lookahead.

The preceding paragraph boils down to this: in evaluating an action, one must consider not only its effect on the environment, but also its effect on the internal state of the agent via the percepts it generates (see also Section 16.6). In still plainer terms: such considerations allow the agent to see the value of (actively) looking before leaping, to hunt for lost keys, and so on.

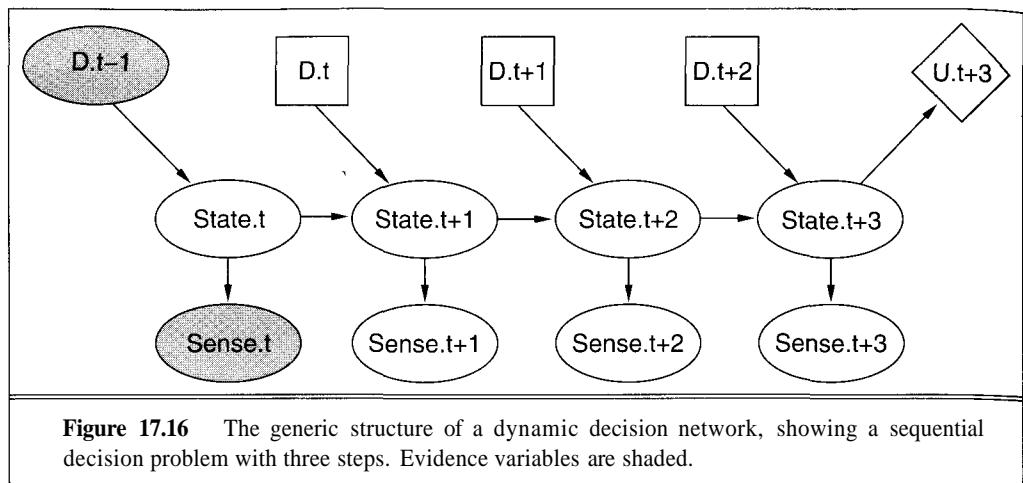


Figure 17.16 The generic structure of a dynamic decision network, showing a sequential decision problem with three steps. Evidence variables are shaded.

Just as we used a limited horizon in game playing and with value iteration and policy iteration, we can limit the extent of forward projection in the DDN in order to reduce complexity. This, combined with a heuristic estimate for the utility of the remaining steps, can provide a reasonable approximation to rational action. There are many other possible approximation techniques, such as using less detailed state variables for states in the distant future; using a greedy heuristic search through the space of decision sequences; assuming "most likely" values for future percept sequences rather than considering all possible values; and so on. There remain many possible techniques for DDN evaluation that are as yet unexplored.

Discussion

All in all, the DDN promises potential solutions to many of the problems that arise as AI systems are moved from static, accessible, and above all *simple* environments to dynamic, inaccessible, complex environments that are closer to the real world.

- They can handle uncertainty correctly, and sometimes efficiently.
- They deal with continuous streams of sensor input.
- They can handle unexpected events because they have no fixed "plan."
- They can handle noisy sensors and sensor failure.
- They can act in order to obtain relevant information.
- They can handle relatively large state spaces because they decompose the state into a set of state variables with sparse connections.
- They exhibit "graceful degradation" under time pressure and in complex environments, using various approximation techniques.

What is missing? The first, and probably the most important, defect of DDNs is that they retain the property of forward search through concrete states that is typical of the search algorithms studied in Part II. In Part IV, we explained how the ability to consider partially ordered, abstract plans using goal-directed search provided a massive increase in problem-solving power,

particularly when combined with plan libraries. At present, we do not really know how to extend these methods into the probabilistic domain. A second, related problem is the basically propositional nature of our probabilistic language. It is impossible, *within* the language of probability theory, to state properly beliefs such as "If any car hits a lamp post going over 30 mph, the occupants of the car will be injured with probability 0.6," because probability theory has no quantifiers ("any car") and no functions ("occupants of the car"). What this means in practice is that some of what goes on in DBNs and DDNs is that programs (rather than pure probabilistic inferences) are responsible for choosing which random variables to instantiate and for filling in their conditional probability tables. If we had an appropriate combination of first-order logic with probability, many of these difficulties could be addressed within a well-understood reasoning system. Work on such a language is one of the most important topics in knowledge representation research, and some progress has been made recently (Bacchus, 1990; Bacchus *et al.*, 1992).

Overall, the potential payoff of combining DDN-like techniques with planning methods is enormous. The technical and mathematical problems involved in getting it right are difficult, but it is an important area of current research.

17.7 SUMMARY

This chapter shows how to use knowledge about the world to make decisions even when the outcomes of an action are uncertain and the payoffs will not be reaped until several (or many) actions have passed. The main points are as follows:

- Sequential decision problems in uncertain environments can be solved by calculating a **policy** that associates an optimal decision with every state that the agent might reach.
- **Value iteration and policy iteration** are two methods for calculating optimal policies. Both are closely related to the general computational technique of dynamic programming.
- Slightly more complex methods are needed to handle the case where the length of the action sequence is unbounded. We briefly discussed the use of **system gain** and **discounting**.
- State-based methods for sequential decision problems do not scale well to large state spaces. Heuristic techniques using best-first search and a limited horizon seem to mitigate this to some extent, but suffer from local minima.
- Decomposing the state into a set of state variables provides a significant advantage. It also simplifies the handling of inaccessible environments.
- We derived a simple updating cycle for a decision-theoretic agent, using a set of **Markov assumptions**.
- We showed how **dynamic belief networks** can handle sensing and updating over time, and provide a direct implementation of the update cycle.
- We showed how **dynamic decision networks** can solve sequential decision problems, handling many (but not all) of the issues arising for agents in complex, uncertain domains.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

Richard Bellman (1957) initiated the modern approach to sequential decision problems, and proposed the dynamic programming approach in general and the value iteration algorithm in particular. A remarkable Ph.D. thesis by Ron Howard (1960) introduced policy iteration and the idea of system gain for solving infinite-horizon problems. Several additional results were introduced by Bellman and Dreyfus (1962). The analysis of discounting in terms of stationary preferences is due to Koopmans (1972). Bertsekas (1987) provides an authoritative modern text on dynamic programming, which has become one of the most widely used tools for search and optimization problems.

The observation that partially observable Markov decision problems can be transformed into a regular Markov decision problem using the belief states is due to Astrom (1965). The first complete algorithm for exact solution of partially-observable Markov decision problems (POMDPs) was proposed by Edward Sondik (1971) in his Ph.D. thesis. (A later journal paper by Smallwood and Sondik (1973) contains some errors but is more accessible.) Lovejoy (1991) surveys the state of the art in POMDPs. In AI, Cassandra *et al.* (1994) have investigated the application of POMDP algorithms to planning problems.

Several recent papers have attempted to combine dynamic programming algorithms such as policy iteration with planning and search models from AI (Dean *et al.*, 1993; Tash and Russell, 1994). This line of work involves approximating a Markov decision problem using a limited horizon and abstract states, in an effort to overcome the combinatorics of large state spaces. Heuristics based on the value of information can be used to select areas of the state space where a local expansion of the horizon will yield a significant improvement in decision quality. Agents using this approach can tailor their effort to handle time pressure, and generate some interesting behaviors such as using familiar "beaten paths" to find their way around the state space quickly without having to recompute optimal decisions at each point.

Many of the basic ideas for estimating the state of dynamical systems came from the mathematician C. F. Gauss (1809). The prediction-estimation cycle for monitoring environments under uncertainty was proposed by Kalman (Kalman, 1960), building on classified wartime research by Wiener (1942) and Kolmogorov (1941). Kalman filtering, which Kalman derived for linear systems with Gaussian noise, has since become an industry in itself (Gelb, 1974; Bar-Shalom and Fortmann, 1988). Leonard and Durrant-Whyte (1992) describe probabilistic sensor models in detail, with particular attention to the modelling of sonar sensors.

Dynamic belief networks (DBNs) can be viewed as a sparse encoding of a Markov process, and were first used in AI by Dean and Kanazawa (1989), Nicholson (1992), and Kjaerulff (1992). The last work includes a generic extension to the HUGIN belief net system to provide the necessary facilities for dynamic belief network generation and compilation. The development given in this chapter owes a good deal to the book by Dean and Wellman (1991), which provides extensive discussion of the use of DBNs and DDNs (dynamic decision networks) in mobile robots. Huang *et al.* (1994) describe an application of DBNs to the analysis of freeway traffic using computer vision. A notation and an evaluation algorithm for additive DDNs are provided by Tatman and Shachter (1990).

EXERCISES

17.1 For the stochastic version of the world shown in Figure 17.1, calculate which squares can be reached by the action sequence [*Up*, *Right*], and with what probabilities.

17.2 For a specific environment (which you can make up), construct a utility function on histories that is *not* separable. Explain how the concept of utility on states fails in this case.



17.3 Consider the stochastic version of the environment shown in Figure 17.1.

- Implement an environment simulator for this environment, such that the specific geography of the environment is easily altered.
- Create a SIMPLE-POLICY-AGENT that uses policy iteration, and measure its performance in the environment simulator from various starting states.
- Experiment with increasing the size of the environment. How does the execution time per action vary with the size of the environment?
- Analyze the policy iteration algorithm to find its worst-case complexity, assuming that value determination is done using a standard equation-solving method.
- Does value iteration terminate if the utility values are required to converge *exactly*?



17.4 For the environment shown in Figure 17.1, find all the threshold values for the cost of a step, such that the optimal policy changes when the threshold is crossed.

17.5 Prove that the calculations in the prediction and estimation phases of the basic decision cycle (Equations (17.8) and (17.9)) do in fact yield the correct value for $Bel(\mathbf{X}_t)$, given assumptions (17.5), (17.6), and (17.7).

17.6 In this exercise, we will consider part of the problem of building a robot that plays Ping-Pong.

One of the things it will have to do is find out where the ball is and estimate its trajectory. Let us suppose for a moment that we have a vision system that can return an estimated instantaneous (x, y, z) position for the ball. This position estimate for time t_i will be used as an evidence node O_i in a belief network that infers the desired information. The idea is that nodes X_j and V_i in the network represent the ball's instantaneous position and velocity. The trajectory of the ball is followed by having multiple copies of these nodes, one for each time step. In addition to these nodes, we also have nodes for the instantaneous friction force F_i acting on the ball due to air resistance, which depends only on the current velocity. The times t_1, t_2, t_3, \dots can be assumed to be separated by some small interval δt .

- Which of the belief networks in Figure 17.17 correctly (but not necessarily efficiently) represent the preceding information? (You may assume that second-order effects—proportional to δt^2 —are ignored.)
- Which is the best network?
- Which networks can be solved by local propagation algorithms?

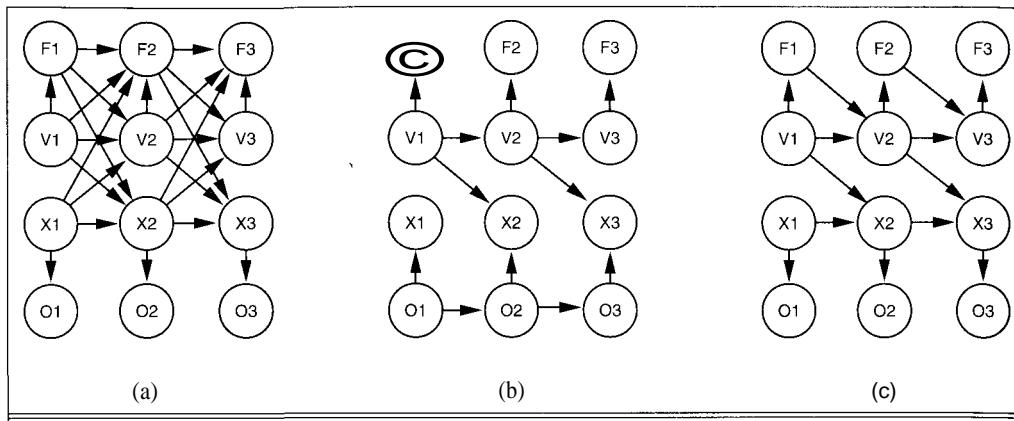


Figure 17.17 Some proposed networks for Ping-Pong tracking.

- d. The nodes in the networks shown in the figure refer to past and present times. Explain how the networks can be extended and used to predict the position at any future time.
 - e. Now consider the vision subsystem, whose job it is to provide observation evidence. Its input is an array of intensity values from the camera image. Assuming that the ball is white, whereas most of the room is darker, explain briefly how to calculate the position of the ball given the array.
 - f. Does it make more sense to calculate the position in coordinates relative to the robot or relative to the room? Why?

Part VI

LEARNING

So far we have assumed that all the "intelligence" in an agent has been built in by the agent's designer. The agent is then let loose in an environment, and does the best it can given the way it was programmed to act. But this is not necessarily the best approach—for the agent or the designer. Whenever the designer has incomplete knowledge of the environment that the agent will live in, learning is the only way that the agent can acquire what it needs to know. Learning thus provides **autonomy** in the sense defined in Chapter 1. It also provides a good way to build high-performance systems—by giving a learning system experience in the application domain.

The four chapters in this part cover the field of **machine learning**—the subfield of AI concerned with programs that learn from experience. Chapter 18 introduces the basic design for learning agents, and addresses the general problem of learning from examples. Chapter 19 discusses the process of learning in neural networks—collections of simple nonlinear processing elements—and in belief networks. In Chapter 20, we tackle the general problem of improving the behavior of an agent given some feedback as to its performance. Finally, Chapter 21 shows how learning can be improved by using prior knowledge.

18

LEARNING FROM OBSERVATIONS

In which we describe agents that can improve their behavior through diligent study of their own experiences.

The idea behind learning is that percepts should be used not only for acting, but also for improving the agent's ability to act in the future. Learning takes place as a result of the interaction between the agent and the world, and from observation by the agent of its own decision-making processes. Learning can range from trivial memorization of experience, as exhibited by the wumpus agent in Chapter 7, to the creation of entire scientific theories, as exhibited by Albert Einstein. This chapter starts with the design of general learning agents, and describes inductive learning—constructing a description of a function from a set of input/output examples. We then give several algorithms for inductive learning in logical agents and a theoretical analysis that explains why learning works.

18.1 A GENERAL MODEL OF LEARNING AGENTS

LEARNING ELEMENT
PERFORMANCE ELEMENT

A learning agent can be divided into four conceptual components, as shown in Figure 18.1. The most important distinction is between the **learning element**, which is responsible for making improvements, and the **performance element**, which is responsible for selecting external actions. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions. The learning element takes some knowledge about the learning element and some feedback on how the agent is doing, and determines how the performance element should be modified to (hopefully) do better in the future. The design of the learning element depends very much on the design of the performance element. When trying to design an agent that learns a certain capability, the first question is not "How am I going to get it to learn this?" but "What kind of performance element will my agent need to do this once it has learned how?" For example, the learning algorithms for producing rules for logical systems are quite different from the learning algorithms for producing belief networks. We will see, however, that the *principles* behind the learning algorithms are much the same.

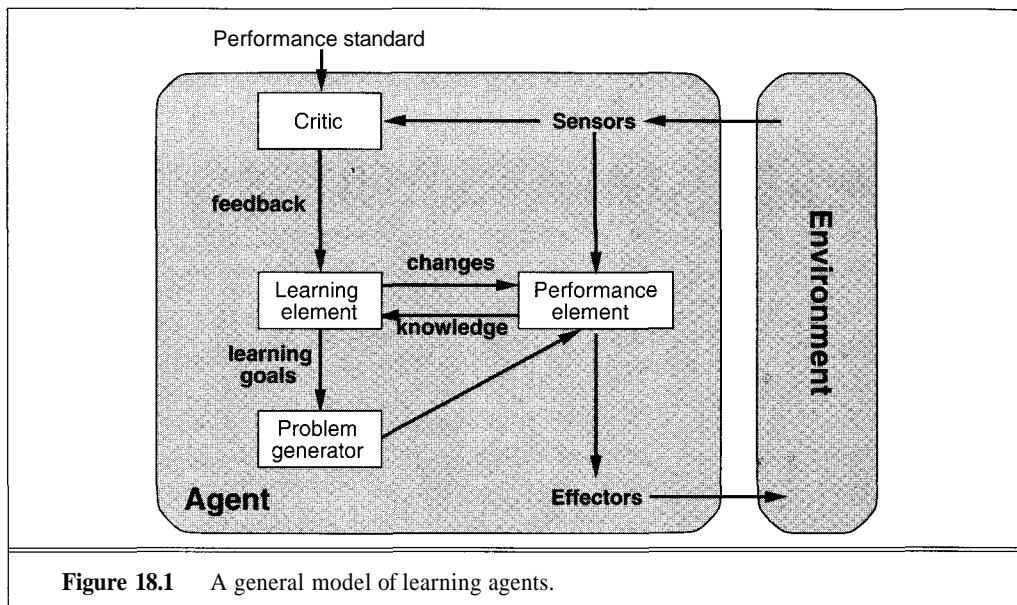


Figure 18.1 A general model of learning agents.

CRITIC

The **critic** is designed to tell the learning element how well the agent is doing. The critic employs a fixed standard of performance. This is necessary because the percepts themselves provide no indication of the agent's success. For example, a chess program may receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so. It is important that the performance standard is a fixed measure that is conceptually outside the agent; otherwise the agent could adjust its performance standards to meet its behavior. In humans, this form of irrationality is called "sour grapes" and is characterized by comments such as "Oh well, never mind, I didn't want that stupid Nobel prize anyway."

PROBLEM
GENERATOR

The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and informative experiences. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. But if the agent is willing to explore a little, and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run. The problem generator's job is to suggest these exploratory actions. This is what scientists do when they carry out experiments. Galileo did not think that dropping rocks from the top of a tower in Pisa was valuable in itself; he was not trying to break the rocks, nor to render pedestrians unconscious. His aim was to demonstrate a better theory of the motion of objects.

To make the overall design more concrete, let us return to the automated taxi example. The performance element consists of whatever collection of knowledge and procedures the taxi has for selecting its driving actions (turning, accelerating, braking, honking, and so on). The taxi goes out on the road and drives, using this performance element. The learning element formulates goals, for example, to learn better rules describing the effects of braking and accelerating, to learn the geography of the area, to learn how the taxi behaves on wet roads, and to learn what

causes annoyance to other drivers. The critic observes the world and passes information along to the learning element. For example, after the taxi makes a quick left turn across three lanes of traffic, the critic observes the shocking language used by other drivers, the learning element is able to formulate a rule saying this was a bad action, and the performance element is modified by installing the new rule. Occasionally, the problem generator kicks in with a suggestion: try taking 7th Avenue uptown this time, and see if it is faster than the normal route.

SPEEDUP LEARNING

The learning element is also responsible for improving the *efficiency* of the performance element. For example, when asked to make a trip to a new destination, the taxi might take a while to consult its map and plan the best route. But the next time a similar trip is requested, the planning process should be much faster. This is called **speedup learning**, and is dealt with in Chapter 21. In this chapter, we concentrate on the acquisition of knowledge.

Machine learning researchers have come up with a large variety of learning elements. To understand them, it will help to see how their design is affected by the context in which they will operate. The design of the learning element is affected by four major issues:

- Which *components* of the performance element are to be improved.
- What *representation* is used for those components.
- What *feedback* is available.
- What *prior information* is available.

Components of the performance element

We have seen that there are many ways to build the performance element of an agent. The components can include the following:

1. A direct mapping from conditions on the current state to actions.
2. A means to infer relevant properties of the world from the percept sequence.
3. Information about the way the world evolves.
4. Information about the results of possible actions the agent can take.
5. Utility information indicating the desirability of world states.
6. Action-value information indicating the desirability of particular actions in particular states.
7. Goals that describe classes of states whose achievement maximizes the agent's utility.

Each of the components can be learned, given the appropriate feedback. For example, if the agent does an action and then perceives the resulting state of the environment, this information can be used to learn a description of the results of actions (4). If the taxi exerts a certain braking pressure when driving on a wet road, then it will soon find out how much actual deceleration is achieved. Similarly, if the critic can use the performance standard to deduce utility values from the percepts, then the agent can learn a useful representation of its utility function (5). If the taxi receives no tips from passengers who have been thoroughly shaken up during the trip, it can learn a useful component of its overall utility function. In a sense, the performance standard can be seen as defining a set of *distinguished percepts* that will be interpreted as providing direct feedback on the quality of the agent's behavior. Hardwired performance standards such as pain and hunger in animals can be understood in this way.

Representation of the components

Any of these components can be represented using any of the representation schemes in this book. We have seen several examples: deterministic descriptions such as linear weighted polynomials for utility functions in game-playing programs and propositional and first-order logical sentences for all of the components in a logical agent; and probabilistic descriptions such as belief networks for the inferential components of a decision-theoretic agent. Effective learning algorithms have been devised for all of these. The details of the learning algorithm will be different for each representation, but the main idea remains the same.

Available feedback

For some components, such as the component for predicting the outcome of an action, the available feedback generally tells the agent what the correct outcome is. That is, the agent predicts that a certain action (braking) will have a certain outcome (stopping in 10 feet), and the environment immediately provides a percept that describes the actual correct outcome (stopping in 15 feet). Any situation in which both the inputs and outputs of a component can be perceived is called **supervised learning**. (Often, the outputs are provided by a friendly teacher.)

SUPERVISED
LEARNING

REINFORCEMENT
LEARNING
REINFORCEMENT

UNSUPERVISED
LEARNING

On the other hand, in learning the condition-action component, the agent receives some evaluation of its action (such as a hefty bill for rear-ending the car in front) but is not told the correct action (to brake more gently and much earlier). This is called **reinforcement learning**; the hefty bill is called a **reinforcement**.¹ The subject is covered in Chapter 20.²

Learning when there is no hint at all about the correct outputs is called **unsupervised learning**. An unsupervised learner can always learn relationships among its percepts using supervised learning methods—that is, it can learn to predict its future percepts given its previous percepts. It cannot learn what to *do* unless it already has a utility function.

Prior knowledge

The majority of learning research in AI, computer science, and psychology has studied the case in which the agent begins with no knowledge at all about what it is trying to learn. It only has access to the examples presented by its experience. Although this is an important special case, it is by no means the general case. Most human learning takes place in the context of a good deal of background knowledge. Some psychologists and linguists claim that even newborn babies exhibit knowledge of the world. Whatever the truth of this claim, there is no doubt that prior knowledge can help enormously in learning. A physicist examining a stack of bubble-chamber photographs may be able to induce a theory positing the existence of a new particle of a certain mass and charge; but an art critic examining the same stack might learn nothing more than that the "artist" must be some sort of abstract expressionist. In Chapter 21, we see several ways in which learning is helped by the use of existing knowledge.

¹ The terms **reward** and **punishment** are also used as synonyms for **reinforcement**.

² Drawing the line between supervised and reinforcement learning is somewhat arbitrary; reinforcement learning can also be thought of as supervised learning with a less informative feedback signal.

Bringing it all together



Each of the seven components of the performance element can be described mathematically as a **function**: for example, information about the way the world evolves can be described as a function from a world state (the current state) to a world state (the next state or states); a goal can be described as a function from a state to a Boolean value (0 or 1) indicating whether the state satisfies the goal. The key point is that *all learning can be seen as learning the representation of a function*. We can choose which component of the performance element to improve and how it is to be represented. The available feedback may be more or less useful, and we may or may not have any prior knowledge. The underlying problem remains the same.

18.2 INDUCTIVE LEARNING

EXAMPLE

PURE INDUCTIVE
INFERENCE
HYPOTHESIS

BIAS

INCREMENTAL
LEARNING

In supervised learning, the learning element is given the correct (or approximately correct) value of the function for particular inputs, and changes its representation of the function to try to match the information provided by the feedback. More formally, we say an **example** is a pair $(x, f(x))$, where x is the input and $f(x)$ is the output of the function applied to x . The task of **pure inductive inference** (or **induction**) is this: given a collection of examples of f , return a function h that approximates f . The function h is called a **hypothesis**.

Figure 18.2 shows an example of this from plane geometry. The examples in Figure 18.2(a) are (x, y) points in the plane, where $y = f(x)$, and the task is to find a function $h(x)$ that fits the points well. In Figure 18.2(b) we have a piecewise-linear h function, while in Figure 18.2(c) we have a more complicated h function. Both functions agree with the example points, but differ on the y values they assign to other x inputs. In (d) we have a function that apparently ignores one of the example points, but fits the others with a simple function. The true f is unknown, so there are many choices for h , but without further knowledge, we have no way to prefer (b), (c), or (d). Any preference for one hypothesis over another, beyond mere consistency with the examples, is called a **bias**. Because there are almost always a large number of possible consistent hypotheses, all learning algorithms exhibit some sort of bias. We will see many examples in this and subsequent chapters.

To get back to agents, suppose we have a reflex agent³ that is being taught by a teacher. Figure 18.3 shows that the REFLEX-LEARNING-ELEMENT updates a global variable, *examples*, that holds a list of (*percept, action*) pairs. The percept could be a chess board position, and the action could be the best move as determined by a helpful grandmaster. When the REFLEX-PERFORMANCE-ELEMENT is faced with a percept it has been told about, it chooses the corresponding action. Otherwise, it calls a learning algorithm INDUCE on the examples it has seen so far. INDUCE returns a hypothesis h which the agent uses to choose an action.

There are many variants on this simple scheme. For example, the agent could perform **incremental learning**: rather than applying the learning algorithm to the entire set of examples each time a new prediction is needed, the agent could just try to update its old hypothesis whenever

³ Recall that reflex agents map directly from percepts to actions.

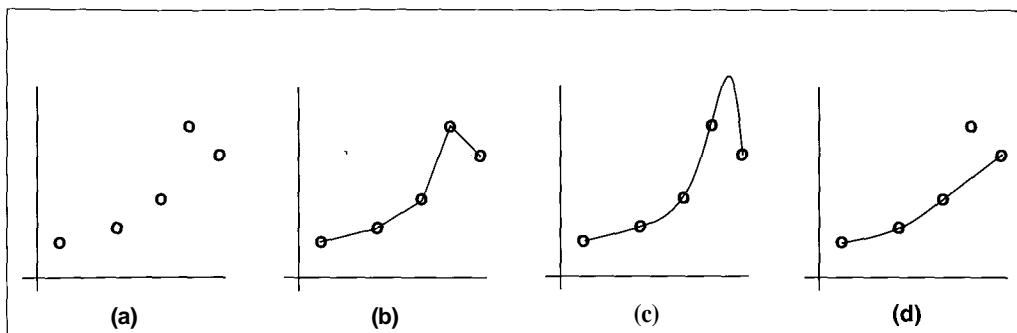


Figure 18.2 In (a) we have some example *(input, output)* pairs. In (b), (c), and (d) we have three hypotheses for functions from which these examples could be drawn.

```
global examples — { }
```

```
function REFLEX-PERFORMANCE-ELEMENT(percept) returns an action
```

```
  if (percept, a) in examples then return a
  else
    h ← INDUCE(examples)
    return h (percept)
```

```
procedure REFLEX-LEARNING-ELEMENT(percept, action)
```

```
  inputs: percept, feedback percept
          action, feedback action
```

```
  examples ← examples U {(percept,action)}  
}
```

Figure 18.3 Skeleton for a simple reflex learning agent. The learning element just stores each example percept/action pair. The performance element either does whatever was done last time for a given percept, or it induces an action from similar percepts. The set of *examples* is a global variable that is shared by the learning and performance elements.

a new example arrives. Also, the agent might receive some feedback concerning the quality of the actions it chooses. These variants, and many others, are examined in this part.

REFLEX-PERFORMANCE-ELEMENT makes no commitment to the way in which the hypothesis is represented. Because of its expressiveness and well-understood semantics, logic has been intensively studied as the target language for learning algorithms. In this chapter, we discuss two approaches to learning logical sentences: **decision tree** methods, which use a restricted representation of logical sentences specifically designed for learning, and the **version-space** approach, which is more general but often rather inefficient. In Chapter 19, we discuss **neural networks**, which are a general representation for nonlinear, numerical functions. The linear weighted polynomials used for game-playing evaluation functions are a special case of neural networks. The

design of learning algorithms for **belief networks** is also a very active area of research, and a brief sketch is provided in Section 19.6.

The choice of *representation* for the desired function is probably the most important issue facing the designer of a learning agent. As well as affecting the nature of the learning algorithm, it can affect whether the problem is feasible at all. As with reasoning, in learning there is a fundamental trade-off between *expressiveness*—is the desired function representable in the representation language—and *efficiency*—is the learning problem going to be tractable for a given choice of representation language. If one chooses to learn sentences in a nice, expressive language such as first-order logic, then one will probably have to pay a heavy penalty in terms of both computation time and the number of examples required to learn a good set of sentences.

By "a good set of sentences," we mean a set that not only correctly reflects the experiences the agent has already had, but also one that correctly predicts its future experiences. Therein lies one of the most vexing philosophical problems of all time. How can one possibly know that one's learning algorithm has produced a theory that will correctly predict the future? And if one does not, then how can one say that the algorithm is any good? Certainly, if one cannot say for sure that an algorithm is any good, then one cannot hope to design good learning algorithms! In Section 18.6, we discuss a mathematical approach to the study of induction algorithms that provides tentative answers to these questions, and also sheds considerable light on the complexity of learning different kinds of function representations.

18.3 LEARNING DECISION TREES

Decision tree induction is one of the simplest and yet most successful forms of learning algorithm. It serves as a good introduction to the area of inductive learning, and is easy to implement. We first describe the performance element, and then show how to learn it. Along the way, we will introduce many of the ideas and terms that appear in all areas of inductive learning.

Decision trees as performance elements

DECISION TREE

A **decision tree** takes as input an object or situation described by a set of properties, and outputs a yes/no "decision." Decision trees therefore represent Boolean functions. Functions with a larger range of outputs can also be represented, but for simplicity we will usually stick to the Boolean case. Each internal node in the tree corresponds to a test of the value of one of the properties, and the branches from the node are labelled with the possible values of the test. Each leaf node in the tree specifies the Boolean value to be returned if that leaf is reached.

GOAL PREDICATE

As an example, consider the problem of whether to wait for a table at a restaurant. The aim here is to learn a definition for the **goal predicate**⁴ *WillWait*, where the definition is expressed as a

⁴ The term **goal concept** is often used. Unfortunately, the word "concept" has been used in so many different ways in machine learning that we think it best to avoid it for a few years.

decision tree.⁵ In setting this up as a learning problem, we first have to decide what properties or *attributes* are available to describe examples in the domain.⁶ Suppose we decide on the following list of attributes:

1. *Alternate*: whether there is a suitable alternative restaurant nearby.
2. *Bar*: whether the restaurant has a comfortable bar area to wait in.
3. *Fri/Sat*: true on Fridays and Saturdays.
4. *Hungry*: whether we are hungry.
5. *Patrons*: how many people are in the restaurant (values are *None*, *Some*, and *Full*).
6. *Price*: the restaurant's price range (\$, \$\$, \$\$\$).
7. *Raining*: whether it is raining outside.
8. *Reservation*: whether we made a reservation.
9. *Type*: the kind of restaurant (French, Italian, Thai, or Burger).
10. *WaitEstimate*: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).

The decision tree usually used by the first author for this domain is shown in Figure 18.4. Notice that the tree does not use the *Price* and *Type* attributes, considering these to be irrelevant given the data it has seen. Logically, the tree can be expressed as a conjunction of individual implications corresponding to the paths through the tree ending in *Yes* nodes. For example, the path for a restaurant full of patrons, with an estimated wait of 10-30 minutes when the agent is not hungry is expressed by the logical sentence

$$\forall r \ Patrons(r, Full) \wedge WaitEstimate(r, 0-10) \wedge \neg Hungry(r, N) \Rightarrow WillWait(r)$$

Expressiveness of decision trees

If decision trees correspond to sets of implication sentences, a natural question is whether they can represent any set. The answer is no, because decision trees are implicitly limited to talking about a single object. That is, the decision tree language is essentially propositional, with each attribute test being a proposition. We cannot use decision trees to represent tests that refer to two or more different objects, for example,

$$\exists r_2 \ Nearby(r_2, r) \wedge Price(r, p) \wedge Price(r_2, p_2) \wedge Cheaper(p_2, p)$$

(is there a cheaper restaurant nearby). Obviously, we could add another Boolean attribute with the name *CheaperRestaurantNearby*, but it is intractable to add *all* such attributes.

Decision trees are fully expressive within the class of propositional languages, that is, any Boolean function can be written as a decision tree. This can be done trivially by having each row in the truth table for the function correspond to a path in the tree. This would not necessarily be a good way to represent the function, because the truth table is exponentially large in the number of attributes. Clearly, decision trees can represent many functions with much smaller trees.

⁵ Meanwhile, the automated taxi is learning whether to wait for the passengers in case they give up waiting for a table and want to go on to another restaurant.

⁶ One might ask why this isn't the job of the learning program. In fact, it is, but we will not be able to explain how it is done until Chapter 21.

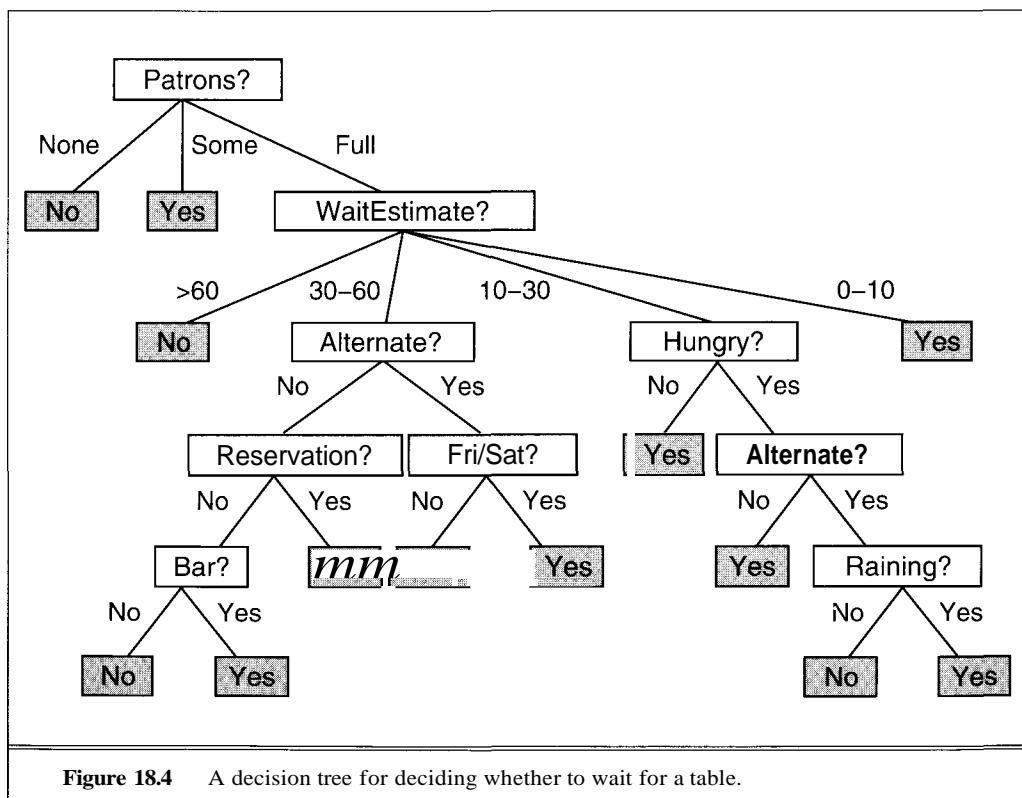
PARITY FUNCTION

MAJORITY FUNCTION

For some kinds of functions, however, this is a real problem. For example, if the function is the **parity function**, which returns 1 if and only if an even number of inputs are 1, then an exponentially large decision tree will be needed. It is also difficult to use a decision tree to represent a **majority function**, which returns 1 if more than half of its inputs are 1.

In other words, decision trees are good for some kinds of functions, and bad for others. Is there any kind of representation that is efficient for all kinds of functions? Unfortunately, the answer is no. We can show this in a very general way. Consider the set of all Boolean functions on n attributes. How many different functions are in this set? This is just the number of different truth tables that we can write down, because the function is defined by its truth table. The truth table has 2^n rows, because each input case is described by n attributes. We can consider the "answer" column of the table as a 2^n bit number that defines the function. No matter what representation we use for functions, some of the functions (almost all of them, in fact) are going to require at least this many bits to represent.

If it takes 2^n bits to define the function, this means that there are 2^{2^n} different functions on n attributes. This is a scary number. For example, with just six Boolean attributes, there are about 2×10^{19} different functions to choose from. We will need some ingenious algorithms to find consistent hypotheses in such a large space.



CLASSIFICATION

TRAINING SET

Inducing decision trees from examples

An **example** is described by the values of the attributes and the value of the goal predicate. We call the value of the goal predicate the **classification** of the example. If the goal predicate is true for some example, we call it a **positive** example; otherwise we call it a **negative** example. A set of examples X_1, \dots, X_{12} for the restaurant domain is shown in Figure 18.5. The positive examples are ones where the goal *WillWait* is true (X_1, X_3, \dots) and negative examples are ones where it is false (X_2, X_5, \dots). The complete set of examples is called the **training set**.

Example	Attributes										Goal <i>WillWait</i>
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	
X_1	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	Yes
X_2	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	No
X_3	No	Yes	No	No	Some	\$	No	No	Burger	0-10	Yes
X_4	Yes	No	Yes	Yes	Full	\$	No	No	Thai	10-30	Yes
X_5	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No
X_6	No	Yes	No	Yes	Some	\$\$	Yes	Yes	Italian	0-10	Yes
X_7	No	Yes	No	No	None	\$	Yes	No	Burger	0-10	No
X_8	No	No	No	Yes	Some	\$\$	Yes	Yes	Thai	0-10	Yes
X_9	No	Yes	Yes	No	Full	\$	Yes	No	Burger	>60	No
X_{10}	Yes	Yes	Yes	Yes	Full	\$\$\$	No	Yes	Italian	10-30	No
X_{11}	No	No	No	No	None	\$	No	No	Thai	0-10	No
X_{12}	Yes	Yes	Yes	Yes	Full	\$	No	No	Burger	30-60	Yes

Figure 18.5 Examples for the restaurant domain.

The problem of finding a decision tree that agrees with the training set might seem difficult, but in fact there is a trivial solution. We could simply construct a decision tree that has one path to a leaf for each example, where the path tests each attribute in turn and follows the value for the example, and the leaf has the classification of the example. When given the same example again,⁷ the decision tree will come up with the right classification. Unfortunately, it will not have much to say about any other cases!

The problem with this trivial tree is that it just memorizes the observations. It does not extract any pattern from the examples and so we cannot expect it to be able to extrapolate to examples it has not seen.

Extracting a pattern means being able to describe a large number of cases in a concise way. Rather than just trying to find a decision tree that agrees with the examples, we should try to find a concise one, too. This is an example of a general principle of inductive learning often called **Ockham's razor**:⁸ *The most likely hypothesis is the simplest one that is consistent with all observations.* Some people interpret this as meaning "the world is inherently simple." Even if the world is complex, however, Ockham's razor still makes sense. There are far fewer simple

⁷ The same example or an example with the same description—this distinction is very important and we will return to it in Chapter 21.

⁸ Sometimes spelled "Occam," although the origin of this corruption is obscure.

hypotheses than complex ones, so that there is only a small chance that *any* simple hypothesis that is wildly incorrect will be consistent with all observations. Hence, other things being equal, a simple hypothesis that is consistent with the observations is more likely to be correct than a complex one. We discuss hypothesis quality further in Section 18.6.

Unfortunately, finding the *smallest* decision tree is an intractable problem, but with some simple heuristics, we can do a good job of finding a smallish one. The basic idea behind the DECISION-TREE-LEARNING algorithm is to test the most important attribute first. By "most important," we mean the one that makes the most difference to the classification of an example. This way, we hope to get to the correct classification with a small number of tests, meaning that all paths in the tree will be short and the tree as a whole will be small.

Figure 18.6 shows how the algorithm gets started. We are given 12 training examples, which we classify into positive and negative sets. We then decide which attribute to use as the first test in the tree. Figure 18.6(a) shows that *Patrons* is a fairly important attribute, because if the value is *None* or *Some*, then we are left with example sets for which we can answer definitively (*No* and *Yes*, respectively). (If the value is *Full*, we will need additional tests.) In Figure 18.6(b) we see that *Type* is a poor attribute, because it leaves us with four possible outcomes, each of which has the same number of positive and negative answers. We consider all possible attributes in this way, and choose the most important one as the root test. We leave the details of how importance is measured for Section 18.4, because it does not affect the basic algorithm. For now, assume the most important attribute is *Patrons*.

After the first attribute test splits up the examples, each outcome is a new decision tree learning problem in itself, with fewer examples and one fewer attribute. There are four cases to consider for these recursive problems:

1. If there are some positive and some negative examples, then choose the best attribute to split them. Figure 18.6(c) shows *Hungry* being used to split the remaining examples.
2. If all the remaining examples are positive (or all negative), then we are done: we can answer *Yes* or *No*. Figure 18.6(c) shows examples of this in the *None* and *Some* cases.
3. If there are no examples left, it means that no such example has been observed, and we return a default value calculated from the majority classification at the node's parent.
4. If there are no attributes left, but both positive and negative examples, we have a problem. It means that these examples have exactly the same description, but different classifications. This happens when some of the data are incorrect; we say there is **noise** in the data. It also happens when the attributes do not give enough information to fully describe the situation, or when the domain is truly nondeterministic. One simple way out of the problem is to use a majority vote.

NOISE

We continue to apply the DECISION-TREE-LEARNING algorithm (Figure 18.7) until we get the tree shown in Figure 18.8. The tree is distinctly different from the original tree shown in Figure 18.4, despite the fact that the data were actually generated from an agent using the original tree.

One might conclude that the learning algorithm is not doing a very good job of learning the correct function. This would be the wrong conclusion to draw. The learning algorithm looks at the *examples*, not at the correct function, and in fact, its hypothesis (see Figure 18.8) not only agrees with all the examples, but is considerably simpler than the original tree. The learning algorithm has no reason to include tests for *Raining* and *Reservation*, because it can classify all

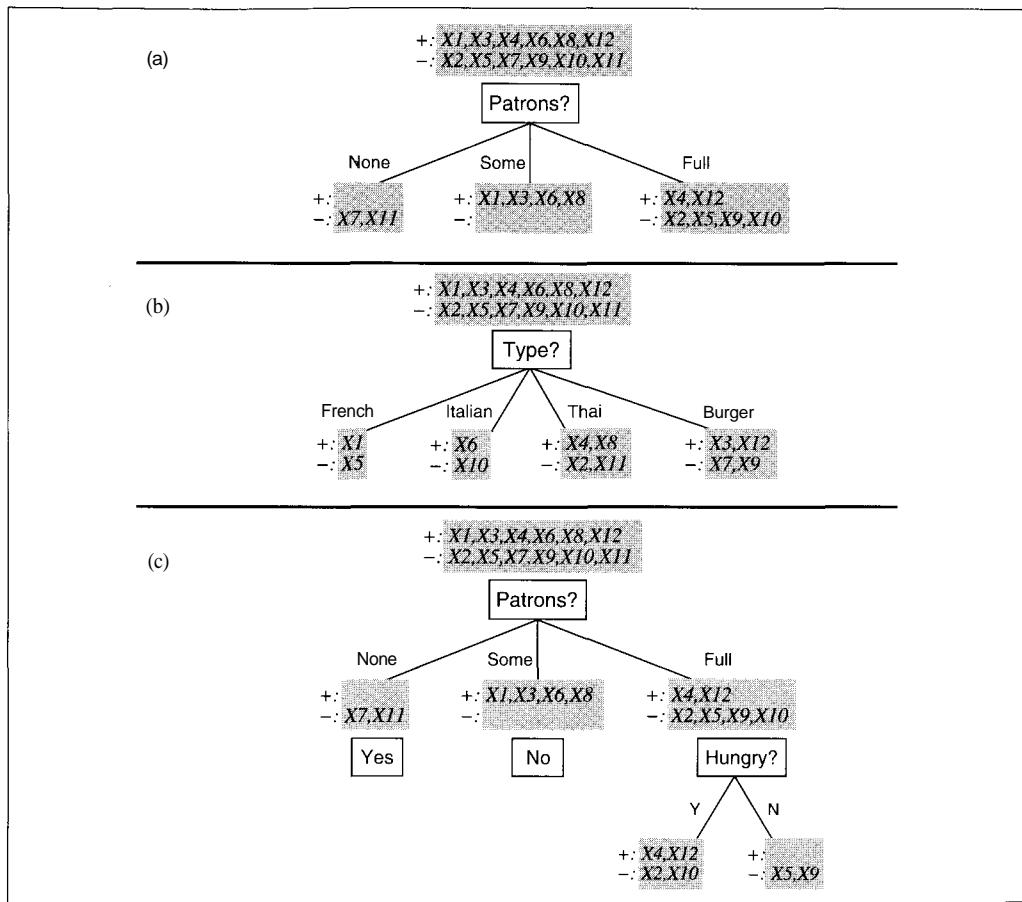


Figure 18.6 Splitting the examples by testing on attributes. In (a), we see that *Patrons* is a good attribute to test first; in (b), we see that *Type* is a poor one; and in (c), we see that *Hungry* is a fairly good second test, given that *Patrons* is the first test.

the examples without them. It has also detected an interesting regularity in the data (namely, that the first author will wait for Thai food on weekends) that was not even suspected. Many hours have been wasted by machine learning researchers trying to debug their learning algorithms when in fact the algorithm was behaving properly all along.

Of course, if we were to gather more examples, we might induce a tree more similar to the original. The tree in Figure 18.8 is bound to make a mistake; for example, it has never seen a case where the wait is 0-10 minutes but the restaurant is full. For a case where *Hungry* is false, the tree says not to wait, but the author would certainly wait. This raises an obvious question: if the algorithm induces a consistent but incorrect tree from the examples, how incorrect will the tree be? The next section shows how to analyze this experimentally.

```

function DECISION-TREE-LEARNING(examples, attributes, default) returns a decision tree
  inputs: examples, set of examples
           attributes, set of attributes
           default, default value for the goal predicate

  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MAJORITY-VALUE(examples)
  else
    best  $\leftarrow$  CHOOSE-ATTRIBUTE(attributes, examples)
    tree  $\leftarrow$  a new decision tree with root test best
    for each value vi of best do
      examplesi  $\leftarrow$  {elements of examples with best = vi}
      subtree  $\leftarrow$  DECISION-TREE-LEARNING(examplesi, attributes — best,
                                         MAJORITY- VALUE(examples))
      add a branch to tree with label vi and subtree subtree
    end
  return tree

```

Figure 18.7 The decision tree learning algorithm.

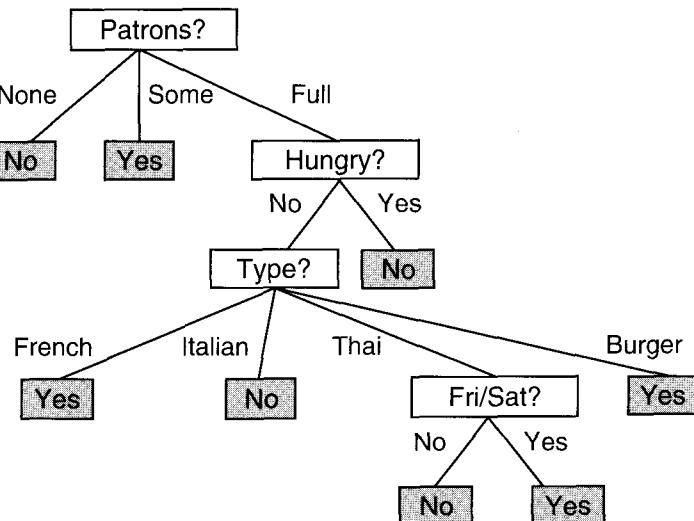


Figure 18.8 The decision tree induced from the 12-example training set.

Assessing the performance of the learning algorithm

A learning algorithm is good if it produces hypotheses that do a good job of predicting the classifications of unseen examples. In Section 18.6, we will see how prediction quality can be estimated in advance. For now, we will look at a methodology for assessing prediction quality after the fact.

TEST SET

Obviously, a prediction is good if it turns out to be true, so we can assess the quality of a hypothesis by checking its predictions against the correct classification once we know it. We do this on a set of examples known as the **test set**. If we train on all our available examples, then we will have to go out and get some more to test on, so often it is more convenient to adopt the following methodology:

1. Collect a large set of examples.
2. Divide it into two disjoint sets: the **training set** and the **test set**.
3. Use the learning algorithm with the training set as examples to generate a hypothesis H .
4. Measure the percentage of examples in the test set that are correctly classified by H .
5. Repeat steps 1 to 4 for different sizes of training sets and different randomly selected training sets of each size.

LEARNING CURVE

The result of this is a set of data that can be processed to give the average prediction quality as a function of the size of the training set. This can be plotted on a graph, giving what is called the **learning curve** for the algorithm on the particular domain. The learning curve for DECISION-TREE-LEARNING with the restaurant examples is shown in Figure 18.9. Notice that as the training set grows, the prediction quality increases. (For this reason, such curves are also called **happy graphs**.) This is a good sign that there is indeed some pattern in the data and the learning algorithm is picking it up.

The key idea of the methodology is to keep the training and test data separate, for the same reason that the results of an exam would not be a good measure of quality if the students saw the test beforehand. The methodology of randomly dividing up the examples into training and test sets is fair when each run is independent of the others—in that case, no run can "cheat" and tell the other runs what the right answers are. But there is the problem that you, as the designer of the learning algorithm, can cheat. If you run some examples, notice a pattern, and change either the learning or the performance element, then the runs are no longer independent, and you have effectively passed on information about the test set. In theory, every time you make a change to the algorithm, you should get a new set of examples to work from. In practice, this is too difficult, so people continue to run experiments on tainted sets of examples.

Practical uses of decision tree learning

Decision trees provide a simple representation for propositional knowledge that can be used for decision making and classification of objects. Although decision tree learning cannot generate interesting scientific theories because of its representational restrictions, it has been used in a wide variety of applications. Here we describe just two.

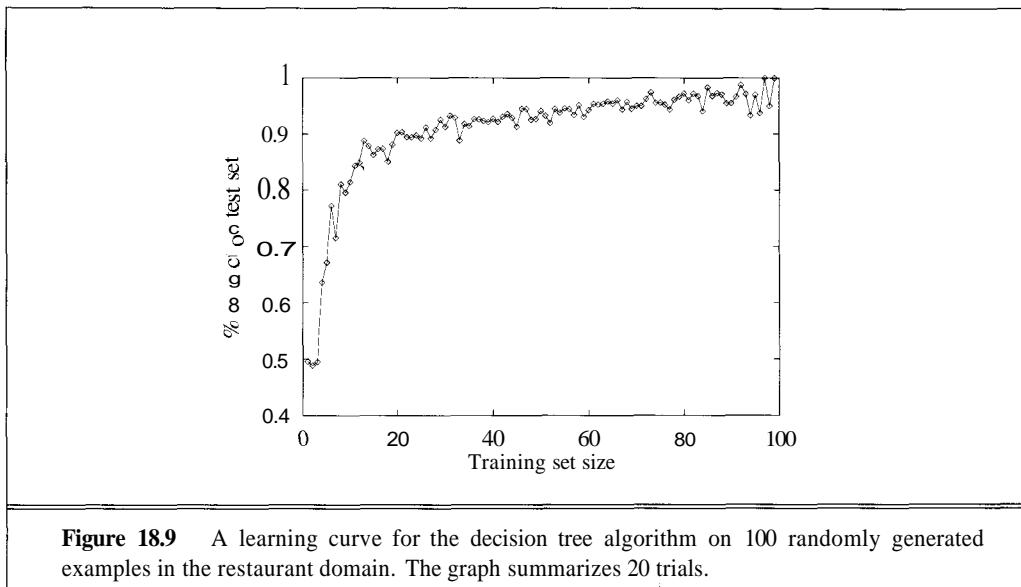


Figure 18.9 A learning curve for the decision tree algorithm on 100 randomly generated examples in the restaurant domain. The graph summarizes 20 trials.

Designing oil platform equipment

In 1986, BP deployed an expert system called GASOIL for designing gas-oil separation systems for offshore oil platforms. Gas-oil separation is done at the wellhead by a very large, complex, and expensive separation system, whose design depends on a number of attributes including the relative proportions of gas, oil, and water, and the flow rate, pressure, density, viscosity, temperature, and susceptibility to waxing. At the time, GASOIL was the largest commercial expert system in the world, containing approximately 2500 rules. Building such a system by hand would have taken roughly 10 person-years. Using decision-tree learning methods applied to a database of existing designs, the system was developed in 100 person-days (Michie, 1986). It is said to outperform human experts and to have saved BP many millions of dollars.

Learning to fly

There are two ways to design an automatic controller for a complex system. One can construct a precise model of the dynamics of the system, and use one of a variety of formal methods (including AI planning methods) to design a controller that has certain guaranteed properties. Alternatively, one can simply learn the correct mapping from the state of the system to the correct action. For very complex systems, such as aircraft and electorates, developing a detailed model may be infeasible, which leaves the second alternative. Sammut *et al.* (1992) adopted this alternative for the task of learning to fly a Cessna on a flight simulator. The data was generated by watching three skilled human pilots performing an assigned flight plan 30 times each. Each time the pilot took an action by setting one of the control variables such as thrust or flaps, a training example was created. In all, 90,000 examples were obtained, each described by 20 state variables and labelled by the action taken. From these examples, a decision tree was extracted

using the C4.5 system (Quinlan, 1993). The decision tree was then converted into C code and inserted into the flight simulator's control loop so that it could fly the plane itself.

The results are surprising: not only does the program learn to fly, it learns to fly somewhat *better* than its teachers. This is because the generalization process cleans up the occasional mistakes made by humans. Such results suggest that machine learning techniques may yield controllers that are more robust than conventional, manually programmed autopilots. For difficult tasks such as flying helicopters carrying heavy loads in high winds, no autopilots are available and very few humans are competent. Such tasks are potentially suitable for systems based on automated learning.

18.4 USING INFORMATION THEORY

This section looks at a mathematical model for choosing the best attribute and at methods for dealing with noise in the data. It can be skipped by those who are not interested in these details.

The scheme used in decision tree learning for selecting attributes is designed to minimize the depth of the final tree. The idea is to pick the attribute that goes as far as possible toward providing an exact classification of the examples. A perfect attribute divides the examples into sets that are all positive or all negative. The *Patrons* attribute is not perfect, but it is fairly good. A really useless attribute such as *Type* leaves the example sets with roughly the same proportion of positive and negative examples as the original set.

All we need, then, is a formal measure of "fairly good" and "really useless" and we can implement the CHOOSE-ATTRIBUTE function of Figure 18.7. The measure should have its maximum value when the attribute is perfect and its minimum value when the attribute is of no use at all. One suitable measure is the expected amount of **information** provided by the attribute, where we use the term in the mathematical sense first defined in (Shannon and Weaver, 1949). To understand the notion of information, think about it as providing the answer to a question, for example, whether a coin will come up heads. If one already has a good guess about the answer, then the actual answer is less informative. Suppose you are going to bet \$1 on the flip of a coin, and you believe that the coin is rigged so that it will come up heads with probability 0.99. You will bet heads (obviously), and have expected value \$0.98 for the bet. That means you would only be willing to pay less than \$0.02 for advance information about the actual outcome of the flip. If the coin were fair, your expected value would be zero, and you would be willing to pay up to \$1.00 for advance information—the less you know, the more valuable the information.

Information theory uses this same intuition, but instead of measuring the value of information in dollars, it measures information content in **bits**. One bit of information is enough to answer a yes/no question about which one has no idea, such as the flip of a fair coin. In general, if the possible answers v_i have probabilities $P(v_i)$, then the information content / of the actual answer is given by

$$I(P(v_1), \dots, P(v_n)) = \sum_{i=1}^n -P(v_i) \log_2 P(v_i)$$

This is just the average information content of the various events (the $-\log_2 P$ terms) weighted by the probabilities of the events. To check this equation, for the tossing of a fair coin we get

$$I\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1 \text{ bit}$$

If the coin is loaded to give 99% heads we get $(1/100, 99/100) = 0.08$ bits, and as the probability of heads goes to 1, the information of the actual answer goes to 0.

For decision tree learning, the question that needs answering is: for a given example, what is the correct classification? A correct decision tree will answer this question. An estimate of the probabilities of the possible answers before any of the attributes have been tested is given by the proportions of positive and negative examples in the training set. Suppose the training set contains p positive examples and n negative examples. Then an estimate of the information contained in a correct answer is

$$\rightarrow \left(\frac{P}{p+n}, \frac{n}{p+n} \right) = -\frac{P}{p+n} \log_2 \frac{P}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

For the restaurant training set shown in Figure 18.5, we have $p = n = 6$, so we need 1 bit of information.

Now a test on a single attribute A will not usually tell us this much information, but it will give us some of it. We can measure exactly how much by looking at how much information we still need *after* the attribute test. Any attribute A divides the training set E into subsets E_1, \dots, E_v according to their values for A , where A can have v distinct values. Each subset E_i has p_i positive examples and n_i negative examples, so if we go along that branch we will need an additional $I(p_i/(p_i + n_i), n_i/(p_i + n_i))$ bits of information to answer the question. A random example has the i th value for the attribute with probability $(p_i + n_i)/(p + n)$, so on average, after testing attribute A , we will need

$$\text{Remainder}(A) = \sum_{i=1}^v \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

INFORMATION GAIN

bits of information to classify the example. The **information gain** from the attribute test is defined as the difference between the original information requirement and the new requirement:

$$\text{Gain}(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{Remainder}(A)$$

and the heuristic used in the CHOOSE-ATTRIBUTE function is just to choose the attribute with the largest gain.

Looking at the attributes *Patrons* and *Type* and their classifying power, as shown in Figure 18.6, we have

$$\text{Gain}(\text{Patrons}) = 1 - \left[\frac{2}{12} I(0, 1) + \frac{4}{12} I(1, 0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] \approx 0.541 \text{ bits}$$

$$\text{Gain}(\text{Type}) = 1 - \left[\frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}$$

In fact, *Patrons* has the highest gain of any of the attributes and would be chosen by the decision tree learning algorithm as the root.

Noise and overfitting

We saw earlier that if there are two or more examples with the same descriptions (in terms of the attributes) but different classifications, then the DECISION-TREE-LEARNING algorithm must fail to find a decision tree consistent with all the examples. The solution we mentioned before is to have each leaf node report either the majority classification for its set of examples or report the estimated probabilities of each classification using the relative frequencies. The former is appropriate for an agent that requires the decision tree to represent a strict logical function, whereas the latter can be used by a decision-theoretic agent.

Unfortunately, this is far from the whole story. It is quite possible, and in fact likely, that even when vital information is missing, the decision tree learning algorithm will find a decision tree that is consistent with all the examples. This is because the algorithm can use the *irrelevant* attributes, if any, to make spurious distinctions among the examples.

Consider the problem of trying to predict the roll of a die. Suppose that experiments are carried out during an extended period of time with various dice, and that the attributes describing each training example are as follows:

1. *Day*: the day on which the die was rolled (Mon, Tue, Wed, Thu).
2. *Month*: the month in which the die was rolled (Jan or Feb).
3. *Color*: the color of the die (Red or Blue).

As long as there are no two examples with identical descriptions, DECISION-TREE-LEARNING will find an exact hypothesis. This will, however, be totally spurious. What we would like is that DECISION-TREE-LEARNING return a single leaf node with probabilities close to 1/6 for each roll, once it has seen enough examples.

OVERTFITTING

Whenever there is a large set of possible hypotheses, one has to be careful not to use the resulting freedom to find meaningless "regularity" in the data. This problem is called overfitting. It is a very general phenomenon, and occurs even when the target function is not at all random. It afflicts every kind of learning algorithm, not just decision trees.

DECISION TREE PRUNING

A complete mathematical treatment of overfitting is beyond the scope of this book. Here we present a simple technique called decision tree pruning. Pruning works by preventing recursive splitting on attributes that are not clearly relevant, even when the data at that node in the tree is not uniformly classified. The question is, how do we detect an irrelevant attribute?

NULL HYPOTHESIS

Suppose we split a set of examples using an irrelevant attribute. Generally speaking, we would expect the resulting subsets to have roughly the same proportions of each class as the original set. In this case, the information gain will be close to zero.⁹ Thus, the information gain is a good clue to irrelevance. Now the question is, how large a gain should we require in order to split on a particular attribute?

This is exactly the sort of question addressed by classical tests for statistical significance. A significance test begins by assuming that there is no underlying pattern (the so-called **null hypothesis**). Then the actual data are analyzed to calculate the extent to which it deviates from a perfect absence of pattern. If the degree of deviation is statistically unlikely (usually taken to mean a 5% probability or less), then that is considered to be good evidence for the presence of a

⁹ In fact, the gain will be greater than zero unless the proportions are all exactly the same (see Exercise 18.9).

significant pattern in the data. The probabilities are calculated from standard distributions of the amount of deviation one would expect to see due to random sampling.

In this case, the null hypothesis is that the attribute is irrelevant, and hence the information gain for an infinitely large sample would be zero. We need to calculate the probability that, under the null hypothesis, a sample of size v would exhibit the observed deviation from the expected distribution of positive and negative examples. We can measure the deviation by comparing the actual numbers of positive and negative examples in each subset, p_i and n_i , to the expected numbers \hat{p}_i and \hat{n}_i assuming true irrelevance:

$$\hat{p}_i = p \times \frac{p_i + n_i}{p + n} \quad \hat{n}_i = n \times \frac{p_i + n_i}{p + n}$$

A convenient measure of the total deviation is given by

$$D = \sum_{i=1}^v \frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(n_i - \hat{n}_i)^2}{\hat{n}_i}$$

Under the null hypothesis, the value of D is distributed according to the χ^2 (chi-squared) distribution with $v - 1$ degrees of freedom. The probability that the attribute is really irrelevant can be calculated with the help of standard χ^2 tables, or with statistical software. Exercise 18.10 asks you to make the appropriate changes to DECISION-TREE-LEARNING to implement this form of pruning, which is known as χ^2 **pruning**.

χ^2 PRUNING

With pruning, noise can be tolerated—classification errors give a linear increase in prediction error, whereas errors in the descriptions of examples (i.e., the wrong output for a given input) have an asymptotic effect that gets worse as the tree shrinks down to smaller sets. Trees constructed with pruning perform significantly better than trees constructed without pruning when the data contain a large amount of noise. The pruned trees are often much more compact, and are therefore easier to understand.

CROSS-VALIDATION

Cross-validation is another technique that eliminates the dangers of overfitting. The basic idea of cross-validation is to try to estimate how well the current hypothesis will predict unseen data. This is done by setting aside some fraction of the known data, and using it to test the prediction performance of a hypothesis induced from the rest of the known data. This can be done repeatedly with different subsets of the data, with the results averaged. Cross-validation can be used in conjunction with any tree-construction method (including pruning) in order to select a tree with good prediction performance.

Broadening the applicability of decision trees

In order to extend decision tree induction to a wider variety of problems, a number of issues must be addressed. We will briefly mention each, suggesting that a full understanding is best obtained by doing the associated exercises:

- 0 **Missing data:** In many domains, not all the attribute values will be known for every example. The values may not have been recorded, or they may be too expensive to obtain. This gives rise to two problems. First, given a complete decision tree, how should one classify an object that is missing one of the test attributes? Second, how should one modify

the information gain formula when some examples have unknown values for the attribute? These questions are addressed in Exercise 18.11.

- ◊ **Multivalued attributes:** When an attribute has a large number of possible values, the information gain measure gives an inappropriate indication of the attribute's usefulness. Consider the extreme case where every example has a different value for the attribute—for instance, if we were to use an attribute *RestaurantName* in the restaurant domain. In such a case, each subset of examples is a singleton and therefore has a unique classification, so the information gain measure would have its highest value for this attribute. However, the attribute may be irrelevant or useless. One possible solution is to use the **gain ratio**, as described in Exercise 18.12.
- ◊ **Continuous-valued attributes:** Attributes such as *Height* and *Weight* have a large or infinite set of possible values. They are therefore not well-suited for decision-tree learning in raw form. An obvious way to deal with this problem is to **discretize** the attribute. For example, the *Price* attribute for restaurants was discretized into \$, \$\$, and \$\$\$ values. Normally, such discrete ranges would be defined by hand. A better approach is to preprocess the raw attribute values during the tree-growing process in order to find out which ranges give the most useful information for classification purposes.

A decision-tree learning system for real-world applications must be able to handle all of these problems. Handling continuous-valued variables is especially important, because both physical and financial processes provide numerical data. Several commercial packages have been built that meet these criteria, and they have been used to develop several hundred fielded systems.

18.5 LEARNING GENERAL LOGICAL DESCRIPTIONS

In this section, we examine ways in which more general kinds of logical representations can be learned. In the process, we will construct a general framework for understanding learning algorithms, centered around the idea that inductive learning can be viewed as a process of searching for a good hypothesis in a large space—the **hypothesis space**—defined by the representation language chosen for the task. We will also explain what is going on in logical terms: the logical connections among examples, hypotheses, and the goal. Although this may seem like a lot of extra work at first, it turns out to clarify many of the issues in learning. It enables us to go well beyond the simple capabilities of the decision tree learning algorithm, and allows us to use the full power of logical inference in the service of learning.

Hypotheses

The situation is usually this: we start out with a goal predicate, which we will generically call Q . (For example, in the restaurant domain, Q will be *WillWait*.) Q will be a unary predicate, and we are trying to find an equivalent logical expression that we can use to classify examples correctly. Each hypothesis proposes such an expression, which we call a **candidate definition** of the goal

GAIN RATIO

HYPOTHESIS SPACE

CANDIDATE DEFINITION

predicate. Using C_i to denote the candidate definition, each hypothesis H_i is a sentence of the form $\forall x Q(x) \Leftrightarrow C_i(x)$. For example, the decision tree shown in Figure 18.8 expresses the following logical definition (which we will call H_r for future reference):

$$\begin{aligned} \forall r \text{ } WillWait(r) &\Leftrightarrow Patrons(r, Some) \\ &\vee Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, French) \\ &\vee Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, Thai) \wedge Fri/Sat(r) \\ &\vee Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, Burger) \end{aligned}$$

The hypothesis space is then the set of all hypotheses that the learning algorithm is designed to entertain. For example, the DECISION-TREE-LEARNING algorithm can entertain any decision tree hypothesis defined in terms of the attributes provided; its hypothesis space therefore consists of all these decision trees. We will generically use the letter H to denote the hypothesis space $\{H_1, \dots, H_n\}$. Presumably, the learning algorithm believes that one of the hypotheses is correct; that is, it believes the sentence

$$H_1 \vee H_2 \vee H_3 \vee \dots \vee H_n \quad (18.1)$$

EXTENSION

Each hypothesis predicts that a certain set of examples—namely, those that satisfy its candidate definition—will be examples of the goal predicate. This set is called the **extension** of the predicate. Two hypotheses with different extensions are therefore logically inconsistent with each other, because they disagree on their predictions for at least one example. If they have the same extension, they are logically equivalent.

Examples

Logically speaking, an example is an object to which the goal concept may or may not apply, and that has some logical description. Let us generically call the i th example X_i . Its description will be the sentence $D_i(X_i)$, where D_i can be any logical expression taking a single argument. The classification will be given by a sentence $Q(X_i)$ if the example is positive, and $\neg Q(X_i)$ if the example is negative. For instance, the first example from Figure 18.5 is described by the sentences

$$Alternate(X_1) \wedge \neg Bar(X_1) \wedge \neg Fri/Sat(X_1) \wedge Hungry(X_1) \wedge \dots$$

and the classification

$$WillWait(X_1)$$

The complete training set is then just the conjunction of all these sentences. A hypothesis agrees with all the examples if and only if it is logically consistent with the training set; ideally, we would like to find such a hypothesis.

Let us examine this notion of consistency more carefully. Obviously, if hypothesis H_i is consistent with the entire training set, it has to be consistent with each example. What would it mean for it to be inconsistent with an example? This can happen in one of two ways:

FALSE NEGATIVE

- An example can be a **false negative** for the hypothesis, if the hypothesis says it should be negative but in fact it is positive. For instance, the new example X_{13} described by

$$Patrons(X_{13}, Full) \wedge Wait(X_{13}, 0-10) \wedge \neg Hungry(X_{13}) \wedge \dots \wedge WillWait(X_{13})$$

would be a false negative for the hypothesis H_r given earlier. From H_r and the example description, we can deduce both $\text{WillWait}(X_{13})$, which is what the example says, and $\neg\text{WillWait}(X_{13})$, which is what the hypothesis predicts. The hypothesis and the example are therefore logically inconsistent.

FALSE POSITIVE

- An example can be a **false positive** for the hypothesis, if the hypothesis says it should be positive but in fact it is negative.¹⁰

If an example is a false positive or false negative for a hypothesis, then the example and the hypothesis are logically inconsistent with each other. Assuming that the example is a correct observation of fact, then the hypothesis can be ruled out. Logically, this is exactly analogous to the resolution rule of inference (see Chapter 9), where the disjunction of hypotheses corresponds to a clause and the example corresponds to a literal that resolves against one of the literals in the clause. An ordinary logical inference system therefore could, in principle, learn from the example by eliminating one or more hypotheses. Suppose, for example, that the example is denoted by the sentence I_1 , and the hypothesis space is $H_1 \vee H_2 \vee H_3 \vee H_4$. Then if I_1 is inconsistent with H_2 and H_3 , the logical inference system can deduce the new hypothesis space $H_1 \vee H_4$.

We therefore can characterize inductive learning in a logical setting as a process of gradually eliminating hypotheses that are inconsistent with the examples, narrowing down the possibilities. Because the hypothesis space is usually vast (or even infinite in the case of first-order logic), we do not recommend trying to build a learning system using resolution-based theorem proving and a complete enumeration of the hypothesis space. Instead, we will describe two approaches that find logically consistent hypotheses with much less effort.

Current-best-hypothesis search

CURRENT-BEST HYPOTHESIS

The idea behind **current-best-hypothesis** search is to maintain a single hypothesis, and to adjust it as new examples arrive in order to maintain consistency. The basic algorithm was described by John Stuart Mill (1843), and may well have appeared even earlier.

Suppose we have some hypothesis such as H_r , of which we have grown quite fond. As long as each new example is consistent, we need do nothing. Then along comes a false negative example, X_{13} . What do we do?

GENERALIZATION

Figure 18.10(a) shows H_r schematically as a region: everything inside the rectangle is part of the extension of H_r . The examples that have actually been seen so far are shown as “+” or “-”, and we see that H_r correctly categorizes all the examples as positive or negative examples of WillWait . In Figure 18.10(b), a new example (circled) is a false negative: the hypothesis says it should be negative but it is actually positive. The extension of the hypothesis must be increased to include it. This is called **generalization**; one possible generalization is shown in Figure 18.10(c). Then in Figure 18.10(d), we see a false positive: the hypothesis says the new example (circled) should be positive, but it actually is negative. The extension of the hypothesis must be decreased to exclude the example. This is called **specialization**; in Figure 18.10(e) we see one possible specialization of the hypothesis.

SPECIALIZATION

¹⁰ The terms "false positive" and "false negative" were first used in medicine to describe erroneous results from laboratory tests. A result is a false positive if it indicates that the patient has the disease when in fact no disease is present.

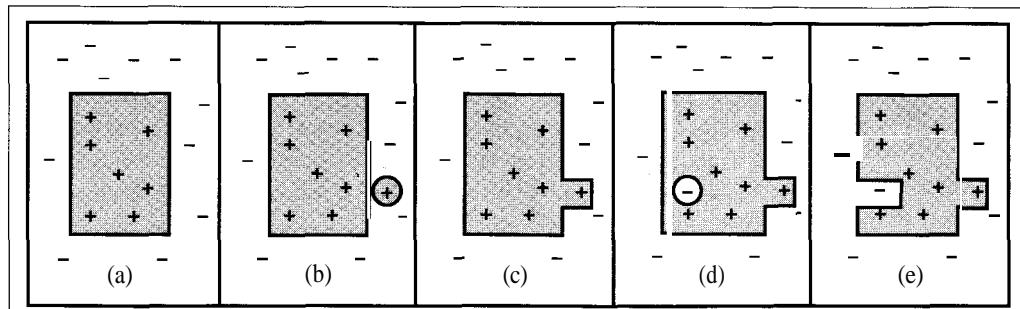


Figure 18.10 (a) A consistent hypothesis. (b) A false negative. (c) The hypothesis is generalized. (d) A false positive. (e) The hypothesis is specialized.

We can now specify the CURRENT-BEST-LEARNING algorithm, shown in Figure 18.11. Notice that each time we consider generalizing or specializing the hypothesis, we must check for consistency with the other examples, because there is no guarantee that an arbitrary increase/decrease in the extension will avoid including/excluding any other negative/positive examples.

```

function CURRENT-BEST-LEARNING(examples)returns a hypothesis
    H — any hypothesis consistent with the first example in examples
    for each remaining example in examples do
        if e is false positive for H then
            H — choose a specialization of H consistent with examples
        else if e is false negative for H then
            H — choose a generalization of H consistent with examples
        if no consistent specialization/generalization can be found then fail
    end
    return H

```

Figure 18.11 The current-best-hypothesis learning algorithm. It searches for a consistent hypothesis and backtracks when no consistent specialization/generalization can be found.

We have defined generalization and specialization as operations that change the *extension* of a hypothesis. Now we need to determine exactly how they can be implemented as syntactic operations that change the candidate definition associated with the hypothesis, so that a program can carry them out. This is done by first noting that generalization and specialization are also *logical* relationships between hypotheses. If hypothesis H_1 , with definition C_1 , is a generalization of hypothesis H_2 with definition C_2 , then we must have

$$\forall x \ C_2(x) \Rightarrow C_1(x)$$

Therefore in order to construct a generalization of H_2 , we simply need to find a definition C_1 that is logically implied by C_2 . This is easily done. For example, if $C_2(x)$ is *Alternate(x)* A

DROPPING
CONDITIONS

$\text{Patrons}(x, \text{Some})$, then one possible generalization is given by $C_1(x) = \text{Patrons}(x, \text{Some})$. This is called **dropping conditions**. Intuitively, it generates a weaker definition and therefore allows a larger set of positive examples. There are a number of other generalization operations, depending on the language being operated on. Similarly, we can specialize a hypothesis by adding extra conditions to its candidate definition or by removing disjuncts from a disjunctive definition. Let us see how this works on the restaurant example, using the data in Figure 18.5.

- The first example X_1 is positive. $\text{Alternate}(X_1)$ is true, so let us assume an initial hypothesis

$$H_1 : \forall x \text{ WillWait}(x) \Leftrightarrow \text{Alternate}(x)$$

- The second example X_2 is negative. H_1 predicts it to be positive, so it is a false positive. Therefore, we need to specialize H_1 . This can be done by adding an extra condition that will rule out X_2 . One possibility is

$$H_2 : \forall x \text{ WillWait}(x) \Leftrightarrow \text{Alternate}(x) \text{ A } \text{Patrons}(x, \text{Some})$$

- The third example X_3 is positive. H_2 predicts it to be negative, so it is a false negative. Therefore, we need to generalize H_2 . This can be done by dropping the *Alternate* condition, yielding

$$H_3 : \forall x \text{ WillWait}(x) \Leftrightarrow \text{Patrons}(x, \text{Some})$$

- The fourth example X_4 is positive. H_3 predicts it to be negative, so it is a false negative. We therefore need to generalize H_3 . We cannot drop the *Patrons* condition, because that would yield an all-inclusive hypothesis that would be inconsistent with X_2 . One possibility is to add a disjunct:

$$H_4 : \forall x \text{ WillWait}(x) \Leftrightarrow \text{Patrons}(x, \text{Some}) \vee (\text{Patrons}(x, \text{Full}) \text{ A } \text{Fri/Sat}(x))$$

Already, the hypothesis is starting to look reasonable. Obviously, there are other possibilities consistent with the first four examples; here are two of them:

$$H'_4 : \forall x \text{ WillWait}(x) \Leftrightarrow \neg \text{WaitEstimate}(x, 30-60)$$

$$H''_4 : \forall x \text{ WillWait}(x) \Leftrightarrow \begin{aligned} &\text{Patrons}(x, \text{Some}) \\ &\vee (\text{Patrons}(x, \text{Full}) \text{ A } \text{WaitEstimate}(x, 10-30)) \end{aligned}$$

The CURRENT-BEST-LEARNING algorithm is described nondeterministically, because at any point, there may be several possible specializations or generalizations that can be applied. The choices that are made will not necessarily lead to the simplest hypothesis, and may lead to an unrecoverable situation where no simple modification of the hypothesis is consistent with all of the data. In such cases, the program must backtrack to a previous choice point.

The CURRENT-BEST-LEARNING algorithm and its variants have been used in many machine learning systems, starting with Patrick Winston's (1970) "arch-learning" program. With a large number of instances and a large space, however, some difficulties arise:

- Checking all the previous instances over again for each modification is very expensive.
- It is difficult to find good search heuristics, and backtracking all over the place can take forever. As we saw earlier, hypothesis space can be a doubly exponentially large place.

Least-commitment search

Backtracking arises because the current-best-hypothesis approach has to *choose* a particular hypothesis as its best guess even though it does not have enough data yet to be sure of the choice. What we can do instead is to keep around all and only those hypotheses that are consistent with all the data so far. Each new instance will either have no effect or will get rid of some of the hypotheses. Recall that the original hypothesis space can be viewed as a disjunctive sentence

$$H_1 \vee H_2 \vee H_3 \dots \vee H_n$$

As various hypotheses are found to be inconsistent with the examples, this disjunction shrinks, retaining only those hypotheses not ruled out. Assuming that the original hypothesis space does in fact contain the right answer, the reduced disjunction must still contain the right answer because only incorrect hypotheses have been removed. The set of hypotheses remaining is called the **version space**, and the learning algorithm (sketched in Figure 18.12) is called the version space learning algorithm (also the **candidate elimination** algorithm).

VERSION SPACE
CANDIDATE
ELIMINATION

```
function VERSION-SPACE-LEARNING(examples) returns a version space
  local variables: V, the version space: the set of all hypotheses
    V — the set of all hypotheses
    for each example e in examples do
      if V is not empty then V ← VERSION-SPACE-UPDATE(V, e)
    end
    return V

function VERSION-SPACE-UPDATE(V, e) returns an updated version space
  V ← {h ∈ V : h is consistent with e}
```

Figure 18.12 The version space learning algorithm. It finds a subset of *V* that is consistent with the *examples*.

One important property of this approach is that it is *incremental*: one never has to go back and reexamine the old examples. All remaining hypotheses are guaranteed to be consistent with them anyway. It is also a **least-commitment** algorithm because it makes no arbitrary choices (cf. the partial-order planning algorithm in Chapter 11). But there is an obvious problem. We already said that the hypothesis space is enormous, so how can we possibly write down this enormous disjunction?

The following simple analogy is very helpful. How do you represent all the real numbers between 1 and 2? After all, there is an infinite number of them! The answer is to use an interval representation that just specifies the boundaries of the set: [1,2]. It works because we have an *ordering* on the real numbers.

We also have an ordering on the hypothesis space, namely, generalization/specialization. This is a partial ordering, which means that each boundary will not be a point but rather a set of hypotheses called a **boundary set**. The great thing is that we can represent the entire version

BOUNDARY SET

G-SET
S-SET

space using just two boundary sets: a most general boundary (the G-set) and a most specific boundary (the S-set). *Everything in between is guaranteed to be consistent with the examples.* Before we prove this, let us recap:

- The current version space is the set of hypotheses consistent with all the examples so far. It is represented by the S-set and G-set, each of which is a set of hypotheses.
- Every member of the S-set is consistent with all observations so far, and there are no consistent hypotheses that are more specific.
- Every member of the G-set is consistent with all observations so far, and there are no consistent hypotheses that are more general.

We want the initial version space (before any examples have been seen) to represent all possible hypotheses. We do this by setting the G-set to contain just *True* (the hypothesis that contains everything), and the S-set to contain just *False* (the hypothesis whose extension is empty).

Figure 18.13 shows the general structure of the boundary set representation of the version space. In order to show that the representation is sufficient, we need the following two properties:

1. Every consistent hypothesis (other than those in the boundary sets) is more specific than some member of the G-set, and more general than some member of the S-set. (That is, there are no "stragglers" left outside.) This follows directly from the definitions of *S* and *G*. If there were a straggler *h*, then it would have to be no more specific than any member of *G*, in which case it belongs in *G*; or no more general than any member of *S*, in which case it belongs in *S*.
2. Every hypothesis more specific than some member of the G-set and more general than some member of the S-set is a consistent hypothesis. (That is, there are no "holes" between the boundaries.) Any *h* between *S* and *G* must reject all the negative examples rejected by each member of *G* (because it is more specific), and must accept all the positive examples accepted by any member of *S* (because it is more general). Thus, *h* must agree with all the examples, and therefore cannot be inconsistent. Figure 18.14 shows the situation: there are no known examples outside *S* but inside *G*, so any hypothesis in the gap must be consistent.

We have therefore shown that if *S* and *G* are maintained according to their definitions, then they provide a satisfactory representation of the version space. The only remaining problem is how to update *S* and *G* for a new example (the job of the VERSION-SPACE-UPDATE function). This may appear rather complicated at first, but from the definitions and with the help of Figure 18.13, it is not too hard to reconstruct the algorithm.

We need to worry about the members *S_i* and *G_i* of the S- and G-sets. For each one, the new instance may be a false positive or a false negative.

1. False positive for *S_i*: This means *S_i* is too general, but there are no consistent specializations of *S_i* (by definition), so we throw it out of the S-set.
2. False negative for *S_i*: This means *S_i* is too specific, so we replace it by all its immediate generalizations.
3. False positive for *G_i*: This means *G_i* is too general, so we replace it by all its immediate specializations.
4. False negative for *G_i*: This means *G_i* is too specific, but there are no consistent generalizations of *G_i* (by definition) so we throw it out of the G-set.

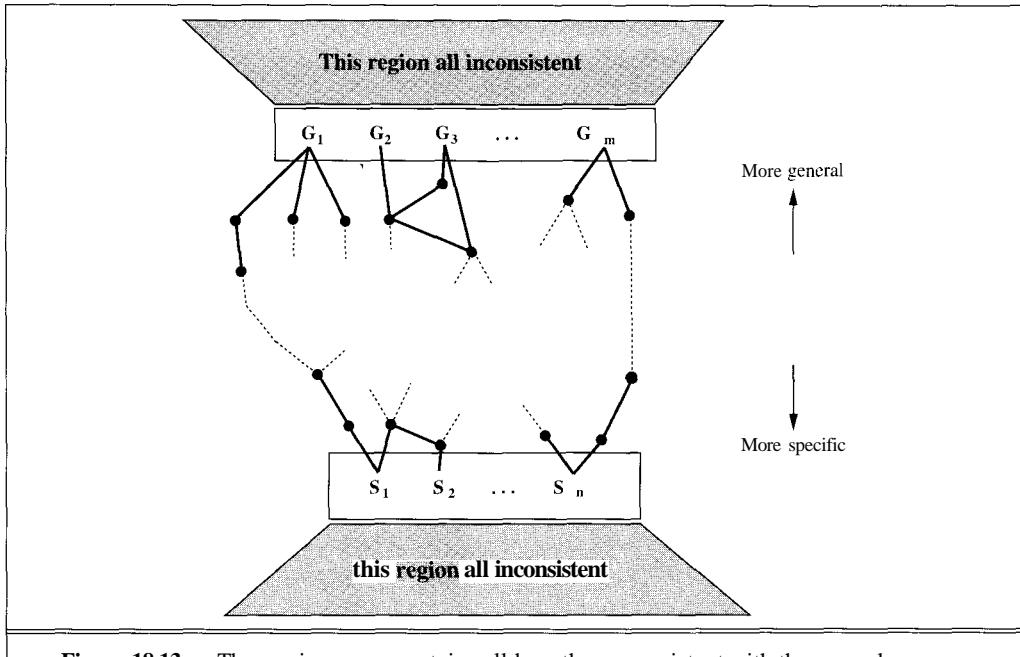


Figure 18.13 The version space contains all hypotheses consistent with the examples.

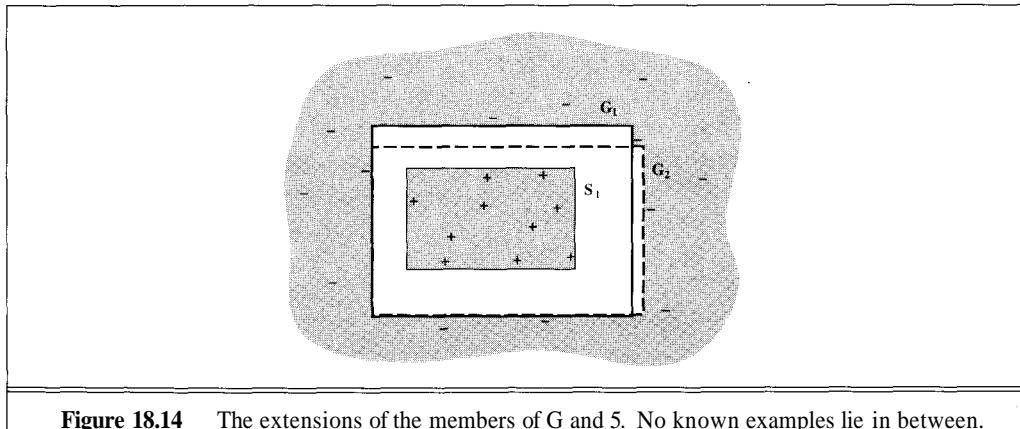


Figure 18.14 The extensions of the members of G and S . No known examples lie in between.

We continue these operations for each new instance until one of three things happens:

1. We have exactly one concept left in the version space, in which case we return it as the unique hypothesis.
2. The version space *collapses*—either S or G becomes empty, indicating that there are no consistent hypotheses for the training set. This is the same case as the failure of the simple version of the decision tree algorithm.

3. We run out of examples with several hypotheses remaining in the version space. This means the version space represents a disjunction of hypotheses. For any new example, if all the disjuncts agree, then we can return their classification of the example. If they disagree, one possibility is to take the majority vote.

We leave as an exercise the application of the VERSION-SPACE-LEARNING algorithm to the restaurant data.

Discussion

There are two principal drawbacks to the version-space approach:

- If the domain contains noise or insufficient attributes for exact classification, the version space will always collapse.
- If we allow unlimited disjunction in the hypothesis space, the S-set will always contain a single most-specific hypothesis, namely, the disjunction of the descriptions of the positive examples seen to date. Similarly, the G-set will contain just the negation of the disjunction of the descriptions of the negative examples.

GENERALIZATION
HIERARCHY

To date, no completely successful solution has been found for the problem of noise. The problem of disjunction can be addressed by allowing limited forms of disjunction or by including a **generalization hierarchy** of more general predicates. For example, instead of using the disjunction $\text{WaitEstimate}(x, 30\text{-}60) \vee \text{WaitEstimate}(x, >60)$, we might use the single literal $\text{LongWait}(x)$. The set of generalization and specialization operations can be easily extended to handle this.

The pure version space algorithm was first applied in the Meta-DENDRAL system, which was designed to learn rules for predicting how molecules would break into pieces in a mass spectrometer (Buchanan and Mitchell, 1978). Meta-DENDRAL was able to generate rules that were sufficiently novel to warrant publication in a journal of analytical chemistry—the first real scientific knowledge generated by a computer program. It was also used in the elegant LEX system (Mitchell *et al.*, 1983), which was able to learn to solve symbolic integration problems by studying its own successes and failures. Although version space methods are probably not practical in most real-world learning problems, mainly because of noise, they provide a good deal of insight into the logical structure of hypothesis space.

18.6 WHY LEARNING WORKS: COMPUTATIONAL LEARNING THEORY

Learning means behaving better as a result of experience. We have shown several algorithms for inductive learning, and explained how they fit into an agent. The main unanswered question was posed in Section 18.2: how can one possibly know that one's learning algorithm has produced a theory that will correctly predict the future? In terms of the definition of inductive learning, how do we know that the hypothesis h is close to the target function f if we don't know what f is?

These questions have been pondered for several centuries, but unless we find some answers, machine learning will, at best, be puzzled by its own success. Fortunately, within the last decade,

COMPUTATIONAL
LEARNING THEORYPROBABLY
APPROXIMATELY
CORRECT
PAC-LEARNING

STATIONARITY

ERROR

 ϵ -BALL

answers have begun to emerge. We will focus on the answers provided by **computational learning theory**, a field at the intersection of AI and theoretical computer science.

The underlying principle is the following: any hypothesis that is seriously wrong will almost certainly be "found out" with high probability after a small number of examples, because it will make an incorrect prediction. Thus, any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong—that is, it must be **Probably Approximately Correct**. **PAC-learning** is the subfield of computational learning theory that is devoted to this idea.

There are some subtleties in the preceding argument. The main question is the connection between the training and the test examples—afterall, we want the hypothesis to be approximately correct on the test set, not just on the training set. The key assumption, introduced by Valiant, is that the training and test sets are drawn randomly from the same population of examples using the *same probability distribution*. This is called the **stationarity** assumption. It is much more precise than the usual proposals for justifying induction, which mutter something about "the future being like the past." Without the stationarity assumption, the theory can make no claims at all about the future because there would be no necessary connection between future and past. The stationarity assumption amounts to supposing that the process that selects examples is not malevolent. Obviously, if the training set consisted only of weird examples—two-headed dogs, for instance—then the learning algorithm cannot help but make unsuccessful generalizations about how to recognize dogs.

How many examples are needed?

In order to put these insights into practice, we will need some notation:

- Let X be the set of all possible examples.
- Let D be the distribution from which examples are drawn.
- Let H be the set of possible hypotheses.
- Let m be the number of examples in the training set.

Initially, we will assume that the true function f is a member of H . Now we can define the **error** of a hypothesis h with respect to the true function f given a distribution D over the examples as the probability that h is different from f on an example:

$$\text{error}(h) = P(h(x) \neq f(x) | \text{x drawn from } D)$$

This is the same quantity being measured experimentally by the learning curves shown earlier.

A hypothesis h is called **approximately correct** if $\text{error}(h) < \epsilon$, where ϵ is a small constant. The plan of attack is to show that after seeing m examples, with high probability, all consistent hypotheses will be approximately correct. One can think of an approximately correct hypothesis as being "close" to the true function in hypothesis space—it lies inside what is called the ϵ -ball around the true function f . Figure 18.15 shows the set of all hypotheses H , divided into the ϵ -ball around f and the remainder, which we call H_{bad} .

We can calculate the probability that a "seriously wrong" hypothesis $h_b \in H_{\text{bad}}$ is consistent with the first m examples as follows. We know that $\text{error}(h_b) > \epsilon$. Thus, the probability that it

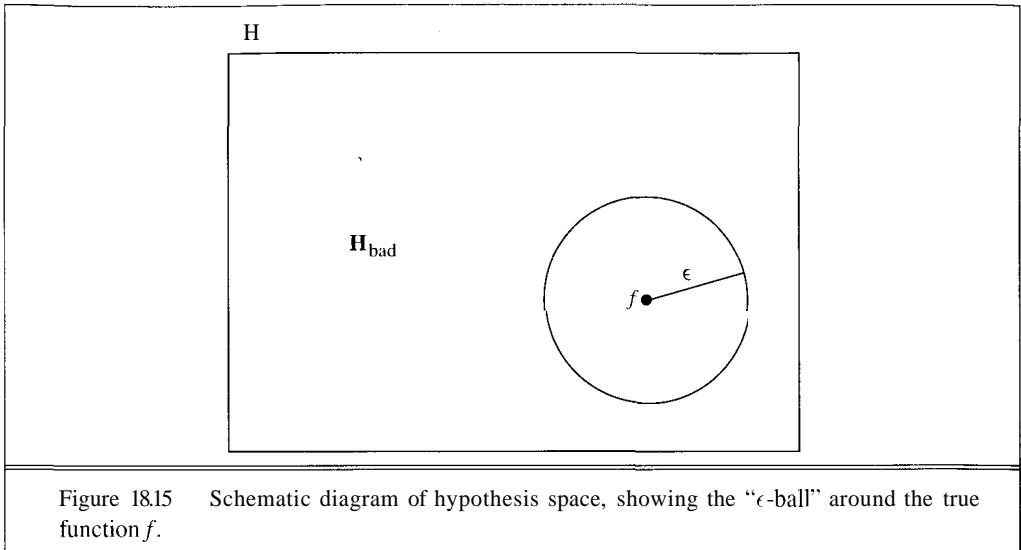


Figure 18.15 Schematic diagram of hypothesis space, showing the “ ϵ -ball” around the true function f .

agrees with any given example is $< (1 - \epsilon)$. The bound for m examples is

$$P(h_b \text{ agrees with } m \text{ examples}) < (1 - \epsilon)^m$$

For H_{bad} to contain a consistent hypothesis, at least one of the hypotheses in H_{bad} must be consistent. The probability of this occurring is bounded by the sum of the individual probabilities:

$$\begin{aligned} P(H_{\text{bad}} \text{ contains a consistent hypothesis}) &< |H_{\text{bad}}|(1 - \epsilon)^m \\ &\leq |H|(1 - \epsilon)^m \end{aligned}$$

We would like to reduce the probability of this event below some small number δ :

$$|H|(1 - \epsilon)^m \leq \delta$$

We can achieve this if we allow the algorithm to see

$$m > \frac{1}{\epsilon} \left(\ln \frac{1}{\delta} + \ln |H| \right) \quad (18.2)$$

examples. Thus, if a learning algorithm returns a hypothesis that is consistent with this many examples, then with probability at least $1 - \delta$, it has error at most ϵ . In other words, it is probably approximately correct. The number of required examples, as a function of ϵ and δ , is called the **sample complexity** of the hypothesis space.

It appears, then, that the key question is the size of the hypothesis space. As we saw earlier, if H is the set of all Boolean functions on n attributes, then $|H| = 2^n$. Thus, the sample complexity of the space grows as 2^n . Because the number of possible examples is also 2^n , this says that any learning algorithm for the space of all Boolean functions will do no better than a lookup table, if it merely returns a hypothesis that is consistent with all known examples. Another way to see this is to observe that for any unseen example, the hypothesis space will contain as many consistent hypotheses predicting a positive outcome as predict a negative outcome.

The dilemma we face, then, is that unless we restrict the space of functions the algorithm can consider, it will not be able to learn; but if we do restrict the space, we may eliminate the true function altogether. There are two ways to "escape" this dilemma. The first way is to insist that the algorithm returns not just any consistent hypothesis, but preferably the simplest one. The theoretical analysis of such algorithms is beyond the scope of this book, but in most cases, finding the simplest hypothesis is intractable. The second escape, which we pursue here, is to focus on learnable subsets of the entire set of Boolean functions. The idea is that in most cases we do not need the full expressive power of Boolean functions, and can get by with more restricted languages. We now examine one such restricted language in more detail.

Learning decision lists

DECISION LIST

A **decision list** is a logical expression of a restricted form. It consists of a series of tests, each of which is a conjunction of literals. If a test succeeds when applied to an example description, the decision list specifies the value to be returned. If the test fails, processing continues with the next test in the list.¹¹ Decision lists resemble decision trees, but their overall structure is simpler, whereas the individual tests are more complex. Figure 18.16 shows a decision list that represents the hypothesis H_4 obtained by the earlier CURRENT-BEST-LEARNING algorithm:

$$\forall x \text{ WillWait}(x) \Leftrightarrow \text{Patrons}(x, \text{Some}) \vee (\text{Patrons}(x, \text{Full}) \wedge \text{Fri/Sat}(x))$$

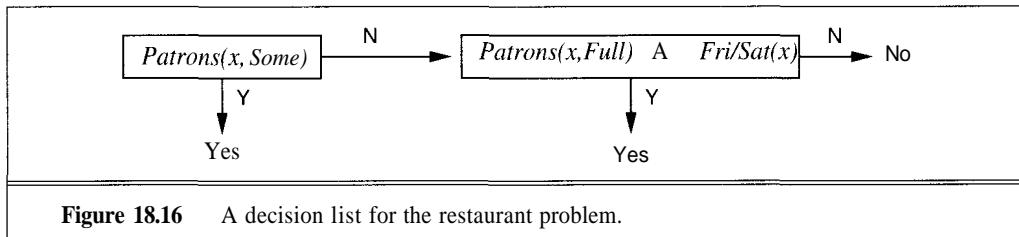


Figure 18.16 A decision list for the restaurant problem.

k -DL k -DT

If we allow tests of arbitrary size, then decision lists can represent any Boolean function (Exercise 18.13). On the other hand, if we restrict the size of each test to at most k literals, then it is possible for the learning algorithm to generalize successfully from a small number of examples. We call this language k -DL. The example in Figure 18.16 is in 2-DL. It is easy to show (Exercise 18.13) that k -DL includes as a subset the language k -DT, the set of all decision trees of depth at most k . It is important to remember that the particular language referred to by k -DL depends on the attributes used to describe the examples. We will use the notation k -DL(n) to denote a k -DL language using n Boolean attributes.

The first task is to show that k -DL is learnable—that is, any function in k -DL can be accurately approximated after seeing a reasonable number of examples. To do this, we need to calculate the number of hypotheses in the language. Let the language of tests—conjunctions of at most k literals using n attributes—be $\text{Conj}(n, k)$. Because a decision list is constructed of tests, and each test can be attached to either a *Yes* or a *No* outcome or can be absent from the decision list, there

¹¹ A decision list is therefore identical in structure to a COND statement in Lisp.

are at most $3^{|Conj(n,k)|}$ distinct sets of component tests. Each of these sets of tests can be in any order, so

$$|k\text{-DL}(n)| \leq 3^{|Conj(n,k)|} |Conj(n, k)|!$$

The number of conjunctions of *If* literals from n attributes is given by

$$|Conj(n, k)| = \sum_{i=0}^k \binom{2n}{i} = O(n^k)$$

Hence, after some work, we obtain

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}$$

We can plug this into Equation (18.2) to show that the number of examples needed for PAC-learning a k -DL function is polynomial in n :

$$m > \frac{1}{\epsilon} \left(\ln \frac{1}{\delta} + O(n^k \log_2(n^k)) \right)$$

Therefore, any algorithm that returns a consistent decision list will PAC-learn a k -DL function in a reasonable number of examples, for small k . The next task is to find an efficient algorithm that returns a consistent decision list. We will use a greedy algorithm called DECISION-LIST-LEARNING that repeatedly finds a test that agrees exactly with some subset of the training set. Once it finds such a test, it adds it to the decision list under construction and removes the corresponding examples. It then constructs the remainder of the decision list using just the remaining examples. This is repeated until there are no examples left. The algorithm is shown in Figure 18.17.

This algorithm does not specify the method for selecting the next test to add to the decision list. Although the formal results given earlier do not depend on the selection method, it would seem reasonable to prefer small tests that match large sets of uniformly classified examples, so that the overall decision list will be as compact as possible. The simplest strategy is to find the smallest test t that matches any uniformly classified subset, regardless of the size of the subset. Even this approach works quite well. The results for the restaurant data are shown in Figure 18.18, and suggest that learning decision lists is an effective way to make predictions.

```

function DECISION-LIST-LEARNING(examples) returns a decision list, No or failure
  if examples is empty then return the value No
  t — a test that matches a nonempty subset examplest of examples
    such that the members of examplest are all positive or all negative
  if there is no such t then return failure
  if the examples in examplest are positive then o  $\leftarrow$  Yes
  else o — No
  return a decision list with initial test t and outcome o
        and remaining elements given by DECISION-LIST-LEARNING(examples - examplest)

```

Figure 18.17 An algorithm for learning decision lists.

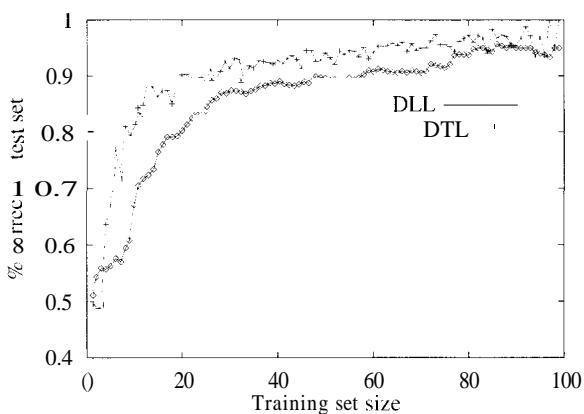


Figure 18.18 Graph showing the predictive performance of the DECISION-LIST-LEARNING algorithm on the restaurant data, as a function of the number of examples seen. The curve for DECISION-TREE-LEARNING is shown for comparison.

Discussion

IDENTIFICATION IN
THE LIMIT

Computational learning theory has generated a new way of looking at the problem of learning. In the early 1960s, the theory of learning focussed on the problem of **identification in the limit**. An identification algorithm must return a hypothesis that exactly matches the true function. The standard approach combines the current-best-hypothesis and version space methods: the current best hypothesis is the first consistent hypothesis in some fixed simplicity ordering of the hypothesis space. As examples arrive, the learner abandons simpler hypotheses as they become inconsistent. Once it reaches the true function, it will never abandon it. Unfortunately, in many hypothesis spaces, the number of examples and the computation time required to reach the true function is enormous. Computational learning theory does not insist that the learning agent find the "one true law" governing its environment, but instead that it find a hypothesis with a certain degree of predictive accuracy. It also brings sharply into focus the trade-off between the expressiveness of the hypothesis language and the complexity of learning.

The results we have shown are worst-case complexity results, and do not necessarily reflect the average-case sample complexity as measured by the learning curves we have shown. An average-case analysis must also make assumptions as to the distribution of examples and the distribution of true functions that the algorithm will have to learn. As these issues become better understood, computational learning theory is providing valuable guidance to machine learning researchers who are interested in predicting or modifying the learning ability of their algorithms. Besides decision lists, results have been obtained for almost all known subclasses of Boolean functions, for neural networks (see Chapter 19) and for sets of first-order logical sentences (see Chapter 21). The results show that the pure inductive learning problem, where the agent begins with no prior knowledge about the target function, is generally very hard. As we

show in Chapter 21, the use of prior knowledge to guide inductive learning makes it possible to learn quite large sets of sentences from reasonable numbers of examples, even in a language as expressive as first-order logic.

18.7 SUMMARY

We have seen that all learning can be seen as learning a function, and in this chapter we concentrate on induction: learning a function from example input/output pairs. The main points were as follows:

- Learning in intelligent agents is essential for dealing with unknown environments (i.e., compensating for the designer's lack of omniscience about the agent's environment).
- Learning is also essential for building agents with a reasonable amount of effort (i.e., compensating for the designer's laziness, or lack of time).
- Learning agents can be divided conceptually into a **performance element**, which is responsible for selecting actions, and a **learning element**, which is responsible for modifying the performance element.
- Learning takes many forms, depending on the nature of the performance element, the available feedback, and the available knowledge.
- Learning any particular component of the performance element can be cast as a problem of learning an accurate representation of a **function**.
- Learning a function from examples of its inputs and outputs is called **inductive learning**.
- The difficulty of learning depends on the chosen representation. Functions can be represented by logical sentences, polynomials, belief networks, neural networks, and others.
- Decision trees are an efficient method for learning deterministic Boolean functions.
- Ockham's razor suggests choosing the simplest hypothesis that matches the observed examples. The information gain heuristic allows us to find a simple decision tree.
- The performance of inductive learning algorithms is measured by their learning curve, which shows the prediction accuracy as a function of the number of observed examples.
- We presented two general approaches for learning logical theories. The current-best-hypothesis approach maintains and adjusts a single hypothesis, whereas the version space approach maintains a representation of all consistent hypotheses. Both are vulnerable to noise in the training set.
- Computational learning theory analyses the sample complexity and computational complexity of inductive learning. There is a trade-off between the expressiveness of the hypothesis language and the ease of learning.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The general architecture for learning systems portrayed in Figure 18.1 is classic in the machine learning literature (Buchanan *et al.*, 1978). The architecture itself, as embodied in programs, goes back at least as far as Arthur Samuel's (1959; 1967) program for playing checkers, which improved its performance through learning. It is described further in Chapter 20.

Claude Shannon was the inventor of information theory (Shannon and Weaver, 1949), which revolutionized the study of communication as well as contributing to the success of decision-tree learning. He also contributed one of the earliest examples of machine learning, a mechanical mouse named Theseus that learned how to travel efficiently through a maze by trial and error. EPAM, the "Elementary Perceiver And Memorizer" (Feigenbaum, 1961), was one of the earliest systems to use decision trees (or **discrimination nets**). EPAM was intended as a cognitive-simulation model of human concept learning. CLS (Hunt *et al.*, 1966) used a heuristic lookahead method to construct decision trees. ID3 (Quinlan, 1979) added the crucial idea of using information content to provide the heuristic function. Quinlan (1986) experimentally investigates the effect of noise on learning and describes a modification of ID3 designed to handle noise. A copy of C4.5, an industrial-strength version of ID3, can be found in Quinlan (1993).

William of Ockham (c. 1285–1349), the most influential philosopher of his century and a major contributor to medieval epistemology, logic, and metaphysics, is credited with a statement called "Ockham's Razor"—in Latin, *Entia non sunt multiplicanda praeter necessitatem*, and in English, "Entities are not to be multiplied without necessity." Unfortunately, this laudable piece of advice is nowhere to be found in his writings in precisely these words.

The current-best-hypothesis approach has been a mainstay of the philosophy of science, as a model of scientific theory formation, for centuries. In AI, the approach is most closely associated with the work of Patrick Winston, whose Ph.D. thesis (Winston, 1970) addressed the problem of learning descriptions of complex objects. Tom Mitchell (1977; 1982) invented version spaces and the learning algorithm that uses them. They were initially used in the Meta-DENDRAL expert system for chemistry (Buchanan and Mitchell, 1978), and later in Mitchell's (1983) LEX system, which learns to solve calculus problems. The importance of bias in inductive learning was emphasized by Mitchell (1980), who showed that an unbiased algorithm is incapable of learning because its hypothesis space always contains equal numbers of consistent hypotheses that predict Q and $\neg Q$ for any example and for any goal predicate Q .

Historically, the earliest research on the theoretical analysis of learning focused on the problem of "identification in the limit" (Gold, 1967), which involves showing that a learning algorithm will eventually converge on the correct hypothesis once it has seen enough examples. This approach was motivated in part by models of scientific discovery from the philosophy of science (Popper, 1962), but has been applied mainly to the problem of learning grammars from example sentences. Osherson, Stob, and Weinstein (1986) provide a modern and rigorous treatment of the field.

Whereas the identification-in-the-limit approach concentrates on eventual convergence, the study of **Kolmogorov complexity or algorithmic complexity**, developed independently by Solomonoff (1964) and Kolmogorov (1965), attempts to provide a formal definition for the notion of simplicity used in Ockham's razor. To escape the problem that simplicity depends on the way

MINIMUM
DESCRIPTION
LENGTH

UNIFORM
CONVERGENCE
THEORY

OCKHAM
ALGORITHM

in which information is represented, it is proposed that simplicity be measured by the length of the shortest program for a universal Turing machine that correctly reproduces the observed data. Although there are many possible universal Turing machines, and hence many possible "shortest" programs, these programs differ in length by at most a constant that is independent of the amount of data. This beautiful insight, which essentially shows that *any* initial representation bias will eventually be overcome by the data itself, is marred only by the undecidability of computing the length of the shortest program. Approximate measures such as the **minimum description length** or MDL (Rissanen, 1984) can be used instead, and have produced excellent results in practice. The recent text by Li and Vitanyi (1993) is the best source for Kolmogorov complexity.

Computational learning theory in the modern sense, that is, the theory of PAC-learning, was inaugurated by Leslie Valiant (1984), but also has roots in the subfield of statistics called **uniform convergence theory** (Vapnik and Chervonenkis, 1971). Valiant's work brought computational complexity issues into the picture, and emphasized the idea that the learner need find only *approximately* correct hypotheses. With Michael Kearns (1990), Valiant showed that several concept classes cannot be PAC-learned tractably even though sufficient information is available in the examples. Some positive results have been obtained for classes such as decision lists (Rivest, 1987), although most PAC-learning results have been negative.

PAC-learning theory and uniform convergence theory were unified by the "four Germans" (none of whom are actually German): Blumer, Ehrenfeucht, Haussler, and Warmuth (1989). PAC learning is also related to other theoretical approaches through the notion of an **Ockham algorithm**. Such algorithms are capable of finding a consistent hypothesis that achieves a "significant" compression of the data it represents. The four Germans showed that if an Ockham algorithm exists for a given class of concepts, then the class is PAC-learnable (Blumer *et al.*, 1990). Board and Pitt (1992) showed that the implication also goes the other way: if a concept class is PAC-learnable, then there must exist an Ockham algorithm for it.

A large number of important papers on machine learning have been collected in *Readings in Machine Learning* (Shavlik and Dietterich, 1990). The two volumes *Machine Learning 1* (Michalski *et al.*, 1983) and *Machine Learning 2* (Michalski *et al.*, 1986) also contain many important papers as well as huge bibliographies. Weiss and Kulikowski (1991) provide a broad introduction to function-learning methods from machine learning, statistics, and neural networks. The STATLOG project (Michie *et al.*, 1994) is by far the most exhaustive investigation into the comparative performance of learning algorithms. Good current research in machine learning is published in the annual proceedings of the International Conference on Machine Learning, in the journal *Machine Learning*, and in mainstream AI journals. Work in computational learning theory appears in the annual ACM Workshop on Computational Learning Theory (COLT), in *Machine Learning*, and in several theoretical journals.

EXERCISES

18.1 Consider the problem faced by an infant learning to speak and understand a language. Explain how this process fits into the general learning model, identifying each of the components of the model as appropriate.

18.2 Repeat Exercise 18.1 for the case of learning to play tennis (or some other competitive sport with which you are familiar). Is this supervised learning or reinforcement learning?

18.3 Draw a decision tree for the problem of deciding whether or not to move forward at a road intersection given that the light has just turned green.

18.4 We never test the same attribute twice along one path in a decision tree. Why not?

18.5 A good "straw man" learning algorithm is as follows: create a table out of all the training examples. Determine which output occurs most often among the training examples; call it d . Then when given an input that is not in the table, just return d . For inputs that are in the table, return the output associated with it (or the most frequent output, if there is more than one). If the input does not appear in the table, then return d . Implement this algorithm and see how well it does in a sample domain. This should give you an idea of the baseline for the domain—the minimal performance that any algorithm should be able to obtain (although many published algorithms have managed to do worse).

18.6 Look back at Exercise 3.16, which asked you to predict from a sequence of numbers (such as [1,4,9,16]) the function underlying the sequence. What techniques from this chapter are applicable to this problem? How would they allow you to do better than the problem-solving approach of Exercise 3.16?

18.7 In the recursive construction of decision trees, it sometimes occurs that a mixed set of positive and negative examples remains at a leaf node, even after all the attributes have been used. Suppose that we have p positive examples and n negative examples.

- Show that the solution used by DECISION-TREE-LEARNING, which picks the majority classification, minimizes the absolute error over the set of examples at the leaf.
- Show that returning the **class probability** $p/(p + n)$ minimizes the sum of squared errors.

CLASS PROBABILITY

18.8 Suppose that a learning algorithm is trying to find a consistent hypothesis when the classifications of examples are actually being generated randomly. There are n Boolean attributes, and examples are drawn uniformly from the set of 2^n possible examples. Calculate the number of examples required before the probability of finding a contradiction in the data reaches 0.5.

18.9 Suppose that an attribute splits the set of examples E into subsets E_i , and that each subset has p_i positive examples and n_i negative examples. Show that unless the ratio $p_i/(p_i + n_i)$ is the same for all i , the attribute has strictly positive information gain.

18.10 Modify DECISION-TREE-LEARNING to include χ^2 -pruning. You may wish to consult Quinlan (1986) for details.



18.11 The standard DECISION-TREE-LEARNING algorithm described in the chapter does not handle cases in which some examples have missing attribute values.

- First, we need to find a way to classify such examples, given a decision tree that includes tests on the attributes for which values may be missing. Suppose an example X has a missing value for attribute A , and that the decision tree tests for A at a node that X reaches. One way to handle this case is to pretend that the example has *all* possible values for the

attribute, but to weight each value according to its frequency among all of the examples that reach that node in the decision tree. The classification algorithm should follow all branches at any node for which a value is missing, and should multiply the weights along each path. Write a modified classification algorithm for decision trees that has this behavior.

- b. Now modify the information gain calculation so that in any given collection of examples C at a given node in the tree during the construction process, the examples with missing values for any of the remaining attributes are given “as-if” values according to the frequencies of those values in the set C .

18.12 In the chapter, we noted that attributes with many different possible values can cause problems with the gain measure. Such attributes tend to split the examples into many small classes or even singleton classes, thereby appearing to be highly relevant according to the gain measure. The **gain ratio** criterion selects attributes according to the ratio between their gain and their intrinsic information content, that is, the amount of information contained in the answer to the question, “What is the value of this attribute?” The gain ratio criterion therefore tries to measure how efficiently an attribute provides information on the correct classification of an example. Write a mathematical expression for the information content of an attribute, and implement the gain ratio criterion in DECISION-TREE-LEARNING.

18.13 In this exercise, we will consider the expressiveness of decision lists, as defined in Section 18.6.

- a. Show that if the tests can be of any size, decision lists can represent any Boolean function.
- b. Show that if the tests can contain at most k literals each, then decision lists can represent any function that can be represented by a decision tree of depth k .

18.14 We have shown how a learning element can improve the performance element. What if we wanted to improve the learning element (or the critic or the problem generator)? Give some examples of this kind of improvement in the taxi domain. Is it possible to represent this kind of learning with our general model of learning agents? How?

19

LEARNING IN NEURAL AND BELIEF NETWORKS

In which we see how to train complex networks of simple computing elements, thereby perhaps shedding some light on the workings of the brain.

This chapter can be viewed in two ways. From a computational viewpoint, it is about a method of representing functions using networks of simple arithmetic computing elements, and about methods for learning such representations from examples. These networks represent functions in much the same way that circuits consisting of simple logic gates represent Boolean functions. Such representations are particularly useful for complex functions with continuous-valued outputs and large numbers of noisy inputs, where the logic-based techniques in Chapter 18 sometimes have difficulty.

NEURAL NETWORKS

From a biological viewpoint, this chapter is about a mathematical model for the operation of the brain. The simple arithmetic computing elements correspond to **neurons**—the cells that perform information processing in the brain—and the network as a whole corresponds to a collection of interconnected neurons. For this reason, the networks are called **neural networks**.¹ Besides their useful computational properties, neural networks may offer the best chance of understanding many psychological phenomena that arise from the specific structure and operation of the brain. We will therefore begin the chapter with a brief look at what is known about brains, because this provides much of the motivation for the study of neural networks. In a sense, we thereby depart from our intention, stated in Chapter 1, to concentrate on rational action rather than on imitating humans. These conflicting goals have characterized the study of neural networks ever since the very first paper on the topic by McCulloch and Pitts (1943). Methodologically speaking, the goals can be reconciled by acknowledging the fact that humans (and other animals) *do* think, and use their powers of thought to act quite successfully in complex domains where current computer-based agents would be lost. It is instructive to try to see how they do it.

Section 19.2 then presents the idealized models that are the main subject of study. Simple, single-layer networks called **perceptrons** are covered in Section 19.3, and general multilayer networks in Section 19.4. Section 19.5 illustrates the various uses of neural networks.

¹ Other names that have been used for the field include **connectionism**, **parallel distributed processing**, **neural computation**, **adaptive networks**, and **collective computation**. It should be emphasized that these are artificial neural networks; there is no attempt to build computing elements out of animal tissue.

The network theme is continued in Section 19.6, where we discuss methods for learning belief networks from examples. The connection is deeper than the superficial similarity implied by the word “network”—not only do the two fields share some learning methods, but in some cases, it can be shown that neural networks *are* belief networks.

19.1 How THE BRAIN WORKS

The exact way in which the brain enables thought is one of the great mysteries of science. It has been appreciated for thousands of years that strong blows to the head can lead to unconsciousness, temporary loss of memory, or even permanent loss of mental capability. This suggests that the brain is somehow involved in thought. It has also long been known that the human brain is somehow different; in about 335 B.C. Aristotle wrote, “Of all the animals, man has the largest brain in proportion to his size.”² Still, it was not until the middle of the eighteenth century that the brain was widely recognized as the seat of consciousness, and it was not until the late nineteenth century that the functional regions of animal brains began to be mapped out. Before the nineteenth century, candidate locations for the seat of consciousness included the heart, the spleen, and the pineal body, a small appendage of the brain present in all vertebrates.

NEURON
* SOMA
DENDRITES
AXON

SYNAPSE

ACTION POTENTIAL

EXCITATORY
INHIBITORY
PLASTICITY

We do know that the **neuron**, or nerve cell, is the fundamental functional unit of all nervous system tissue, including the brain. Each neuron consists of a cell body, or **soma**, that contains a cell nucleus. Branching out from the cell body are a number of fibers called **dendrites** and a single long fiber called the **axon**. Dendrites branch into a bushy network around the cell, whereas the axon stretches out for a long distance—usually about a centimeter (100 times the diameter of the cell body), and as far as a meter in extreme cases. Eventually, the axon also branches into strands and substrands that connect to the dendrites and cell bodies of other neurons. The connecting junction is called a **synapse**. Each neuron forms synapses with anywhere from a dozen to a hundred thousand other neurons. Figure 19.1 shows the parts of a neuron.

Signals are propagated from neuron to neuron by a complicated electrochemical reaction. Chemical transmitter substances are released from the synapses and enter the dendrite, raising or lowering the electrical potential of the cell body. When the potential reaches a threshold, an electrical pulse or **action potential** is sent down the axon. The pulse spreads out along the branches of the axon, eventually reaching synapses and releasing transmitters into the bodies of other cells. Synapses that increase the potential are called **excitatory**, and those that decrease it are called **inhibitory**. Perhaps the most significant finding is that synaptic connections exhibit **plasticity**—long-term changes in the strength of connections in response to the pattern of stimulation. Neurons also form new connections with other neurons, and sometimes entire collections of neurons can migrate from one place to another. These mechanisms are thought to form the basis for learning in the brain.

Most information processing goes on in the cerebral cortex, the outer layer of the brain. The basic organizational unit appears to be a barrel-shaped module of tissue about 0.5 mm in

² Since then, it has been discovered that some species of dolphins and whales have relatively larger brains. The large size of human brains is now thought to be enabled in part by recent improvements in its cooling system.

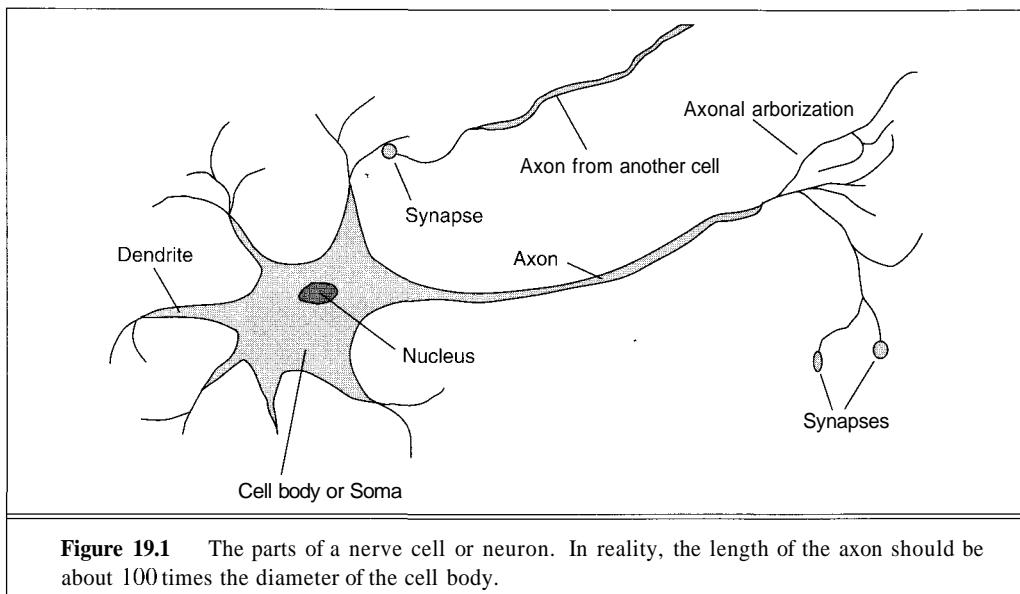


Figure 19.1 The parts of a nerve cell or neuron. In reality, the length of the axon should be about 100 times the diameter of the cell body.

APHASIA

diameter, extending the full depth of the cortex, which is about 4 mm in humans. A module contains about 2000 neurons. It is known that certain areas of the brain have specific functions. In 1861, Pierre Paul Broca was able to demonstrate that the third left frontal convolution of the cerebral cortex is important for speech and language by his studies of patients with **aphasia**—an inability to speak, often brought on by brain damage. This soon led to surgical experiments on animals that mapped out the connection between areas of the cortex and specific motor controls. We now have some data on the mapping between areas of the brain and the parts of the body that they control, or from which they receive sensory input. Such mappings seem to be able to change radically over the course of a few weeks, and some animals seem to have multiple maps. Moreover, we do not fully understand how other areas can take over functions when one area is damaged. There is almost no theory about how an individual memory is stored.

The truly amazing thing is that *a collection of simple cells can lead to thought, action, and consciousness*. Neurobiology is a long way from a complete theory of consciousness, but even if there are some important electrical or chemical processes that have been overlooked, the amazing conclusion is the same: *brains cause minds* (Searle, 1992). The only real alternative theory is mysticism: that there is some mystical realm in which minds operate that is beyond physical science.

Comparing brains with digital computers

Brains and digital computers perform quite different tasks, and have different properties. Figure 19.2 shows that there are more neurons in the typical human brain than there are bits in a typical high-end computer workstation. We can predict that this will not hold true for long, because the human brain is evolving very slowly, whereas computer memories are growing rapidly. In any

	Computer	Human Brain
Computational units	1 CPU, 10^5 gates	10^{11} neurons
Storage units	10^9 bits RAM, 10^{10} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-8} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec
Neuron updates/sec	10^5	10^{14}

Figure 19.2 A crude comparison of the raw computational resources available to computers (circa 1994) and brains.

case, the difference in storage capacity is minor compared to the difference in switching speed and in parallelism. Computer chips can execute an instruction in tens of nanoseconds, whereas neurons require milliseconds to fire. Brains more than make up for this, however, because all the neurons and synapses are active simultaneously, whereas most current computers have only one or at most a few CPUs. A neural network running on a serial computer requires hundreds of cycles to decide if a single neuron-like unit will fire, whereas in a real brain, *all* the neurons do this in a single step. Thus, *even though a computer is a million times faster in raw switching speed, the brain ends up being a billion times faster at what it does*. One of the attractions of the neural network approach is the hope that a device could be built that combines the parallelism of the brain with the switching speed of the computer. Full-scale hardware development will depend on finding a family of neural network algorithms that provides a basis for long-term investment.

A brain can perform a complex task—recognize a face, for example—in less than a second, which is only enough time for a few hundred cycles. A serial computer requires billions of cycles to perform the same task less well. Clearly, there *is* an opportunity for massive parallelism here. Neural networks may provide a model for massively parallel computation that is more successful than the approach of "parallelizing" traditional serial algorithms.

Brains are more fault-tolerant than computers. A hardware error that flips a single bit can doom an entire computation, but brain cells die all the time with no ill effect to the overall functioning of the brain. It is true that there are a variety of diseases and traumas that can affect a brain, but for the most part, brains manage to muddle through for 70 or 80 years with no need to replace a memory card, call the manufacturer's service line, or reboot. In addition, brains are constantly faced with novel input, yet manage to do something with it. Computer programs rarely work as well with novel input, unless the programmer has been exceptionally careful. The third attraction of neural networks is **graceful degradation**: they tend to have a gradual rather than sharp drop-off in performance as conditions worsen.

The final attraction of neural networks is that they are designed to be trained using an inductive learning algorithm. (Contrary to the impression given by the popular media, of course, neural networks are far from being the only AI systems capable of learning.) After the network is initialized, it can be modified to improve its performance on input/output pairs. To the extent that the learning algorithms can be made general and efficient, this increases the value of neural networks as psychological models, and makes them useful tools for creating a wide variety of high-performance applications.



19.2 NEURAL NETWORKS

UNITS A neural network is composed of a number of nodes, or **units**, connected by **links**. Each link has a numeric **weight** associated with it. Weights are the primary means of long-term storage in neural networks, and learning usually takes place by updating the weights. Some of the units are connected to the external environment, and can be designated as input or output units. The weights are modified so as to try to bring the network's input/output behavior more into line with that of the environment providing the inputs.

LINKS ACTIVATION LEVEL Each unit has a set of input links from other units, a set of output links to other units, a current **activation level**, and a means of computing the activation level at the next step in time, given its inputs and weights. The idea is that each unit does a local computation based on inputs from its neighbors, but without the need for any global control over the set of units as a whole. In practice, most neural network implementations are in software and use synchronous control to update all the units in a fixed sequence.

WEIGHT To build a neural network to perform some task, one must first decide how many units are to be used, what kind of units are appropriate, and how the units are to be connected to form a network. One then initializes the weights of the network, and trains the weights using a learning algorithm applied to a set of training examples for the task.³ The use of examples also implies that one must decide how to encode the examples in terms of inputs and outputs of the network.

Notation

Neural networks have lots of pieces, and to refer to them we will need to introduce a variety of mathematical notations. For convenience, these are summarized in Figure 19.3.

Simple computing elements

INPUT FUNCTION Figure 19.4 shows a typical unit. Each unit performs a simple computation: it receives signals from its input links and computes a new activation level that it sends along each of its output links. The computation of the activation level is based on the values of each input signal received from a neighboring node, and the weights on each input link. The computation is split into two components. First is a *linear* component, called the **input function**, in_i , that computes the weighted sum of the unit's input values. Second is a *nonlinear* component called the **activation function**, g , that transforms the weighted sum into the final value that serves as the unit's activation value, a_i . Usually, all units in a network use the same activation function. Exercise 19.3 explains why it is important to have a nonlinear component.

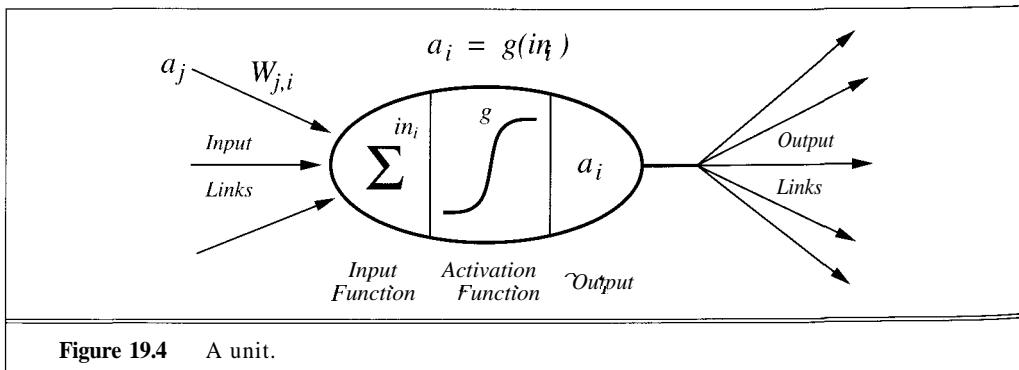
ACTIVATION FUNCTION The total weighted input is the sum of the input activations times their respective weights:

$$in_i = \sum_j W_{j,i} a_j = \mathbf{W}_i \cdot \mathbf{a}_i$$

³ In this chapter, we will assume that all examples are labelled with the correct outputs. In Chapter 20, we will see how to relax this assumption.

Notation	Meaning
a_i	Activation value of unit i (also the output of the unit)
\mathbf{a}_i	Vector of activation values for the inputs to unit i
g	Activation function
g'	Derivative of the activation function
Err_i	Error (difference between output and target) for unit i
Err^e	Error for example e
I	Activation of a unit i in the input layer
\mathbf{I}	Vector of activations of all input units
\mathbf{r}	Vector of inputs for example e
in_i	Weighted sum of inputs to unit i
N	Total number of units in the network
O	Activation of the single output unit of a perceptron
O_i	Activation of a unit i in the output layer
\mathbf{O}	Vector of activations of all units in the output layer
t	Threshold for a step function
T	Target (desired) output for a perceptron
\mathbf{T}	Target vector when there are several output units
\mathbf{T}^e	Target vector for example e
$W_{j,i}$	Weight on the link from unity to unit i
W_i	Weight from unit r to the output in a perceptron
$\mathbf{W},$	Vector of weights leading into unit i
\mathbf{W}	Vector of all weights in the network

Figure 19.3 Neural network notation. Subscripts denote units; superscripts denote examples.



where the final expression illustrates the use of vector notation. In this notation, the weights on links into node i are denoted by \mathbf{W} , the set of input values is called \mathbf{a}_i , and the dot product denotes the sum of the pairwise products.

The elementary computation step in each unit computes the new activation value for the unit by applying the activation function, g , to the result of the input function:

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$

Different models are obtained by using different mathematical functions for g . Three common choices are the step, sign, and sigmoid functions, illustrated in Figure 19.5. The step function has a threshold t such that it outputs a 1 when the input is greater than its threshold, and outputs a 0 otherwise. The biological motivation is that a 1 represents the firing of a pulse down the axon, and a 0 represents no firing. The threshold represents the minimum total weighted input necessary to cause the neuron to fire. Threshold versions of the sign and sigmoid functions can also be defined.

In most cases, we will find it mathematically convenient to replace the threshold with an extra input weight. This allows for a simpler learning element because it need only worry about adjusting weights, rather than adjusting both weights and thresholds. Thus, instead of having a threshold t for each unit, we add an extra input whose activation a_0 is fixed at -1 . The extra weight $W_{0,i}$ associated with a_0 serves the function of a threshold at t , provided that $W_{0,i}a_0 = -t$. Then all units can have a fixed threshold at 0. Mathematically, the two representations for thresholds are entirely equivalent:

$$a_i = \text{step}_t\left(\sum_{j=1}^n W_{j,i} a_j\right) = \text{step}_0\left(\sum_{j=0}^n W_{j,i} a_j\right) \text{ where } W_{0,i} = t \text{ and } a_0 = -1$$

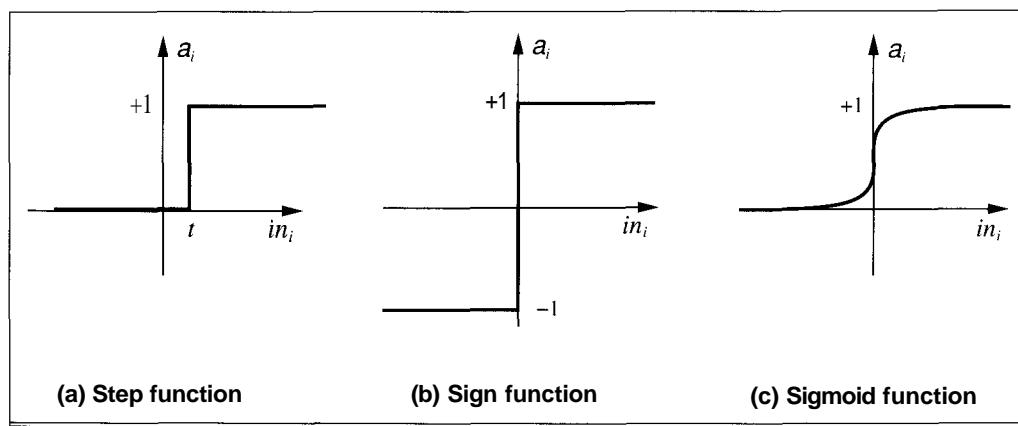
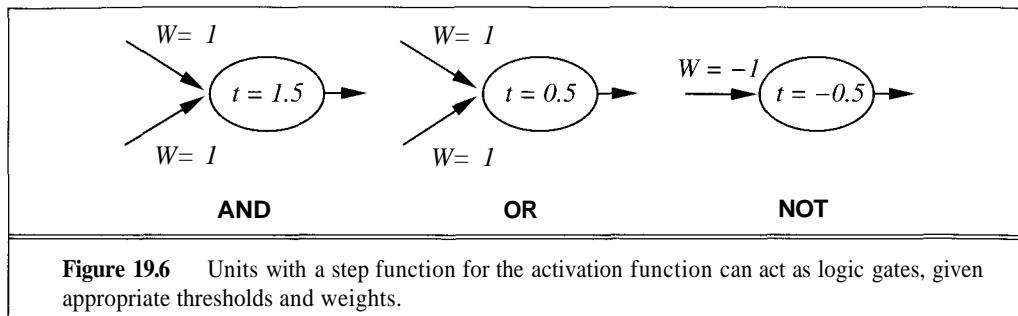


Figure 19.5 Three different activation functions for units.

$$\text{step}_t(x) = \begin{cases} 1, & \text{if } x \geq t \\ 0, & \text{if } x < t \end{cases} \quad \text{sign}(x) = \begin{cases} +1, & \text{if } x > 0 \\ -1, & \text{if } x < 0 \end{cases} \quad \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

We can get a feel for the operation of individual units by comparing them with logic gates. One of the original motivations for the design of individual units (McCulloch and Pitts, 1943) was their ability to represent basic Boolean functions. Figure 19.6 shows how the Boolean functions *AND*, *OR*, and *NOT* can be represented by units with suitable weights and thresholds. This is important because it means we can use these units to build a network to compute any Boolean function of the inputs.



Network structures

FEED-FORWARD
RECURRENT

There are a variety of kinds of network structure, each of which results in very different computational properties. The main distinction to be made is between **feed-forward** and **recurrent** networks. In a feed-forward network, links are unidirectional, and there are no cycles. In a recurrent network, the links can form arbitrary topologies. Technically speaking, a feed-forward network is a directed acyclic graph (DAG). We will usually be dealing with networks that are arranged in layers. In a layered feed-forward network, each unit is linked only to units in the next layer; there are no links between units in the same layer, no links backward to a previous layer, and no links that skip a layer. Figure 19.7 shows a very simple example of a layered feed-forward network. This network has *two* layers; because the input units (square nodes) simply serve to pass activation to the next layer, they are not counted (although some authors would describe this as a three-layer network).

The significance of the lack of cycles is that computation can proceed uniformly from input units to output units. The activation from the previous time step plays no part in the computation, because it is not fed back to an earlier unit. Hence, a feed-forward network simply computes a function of the input values that depends on the weight settings—it has *no internal state* other than the weights themselves. Such networks can implement adaptive versions of simple reflex agents or they can function as components of more complex agents. In this chapter, we will focus on feed-forward networks because they are relatively well-understood.

Obviously, the brain cannot be a feed-forward network, else we would have no short-term memory. Some regions of the brain are largely feed-forward and somewhat layered, but there are rampant back-connections. In our terminology, the brain is a recurrent network. Because activation is fed back to the units that caused it, recurrent networks have internal state stored in the activation levels of the units. This also means that computation can be much less orderly

than in feed-forward networks. Recurrent networks can become unstable, or oscillate, or exhibit chaotic behavior. Given some input values, it can take a long time to compute a stable output, and learning is made more difficult. On the other hand, recurrent networks can implement more complex agent designs and can model systems with state. Because recurrent networks require some quite advanced mathematical methods, we can only provide a few pointers here.

HOPFIELD NETWORKS

ASSOCIATIVE MEMORY

BOLTZMANN MACHINES

INPUT UNITS

OUTPUT UNITS

HIDDEN UNITS

PERCEPTRONS

MULTILAYER NETWORKS

Hopfield networks are probably the best-understood class of recurrent networks. They use *bidirectional* connections with *symmetric* weights (i.e., $W_{ij} = W_{ji}$); all of the units are both input and output units; the activation function g is the sign function; and the activation levels can only be ± 1 . A Hopfield network functions as an **associative memory**—after training on a set of examples, a new stimulus will cause the network to settle into an activation pattern corresponding to the example in the training set that *most closely resembles* the new stimulus. For example, if the training set consists of a set of photographs, and the new stimulus is a small piece of one of the photographs, then the network activation levels will reproduce the photograph from which the piece was taken. Notice that the original photographs are not stored separately in the network; each weight is a partial encoding of all the photographs. One of the most interesting theoretical results is that Hopfield networks can reliably store up to $0.138N$ training examples, where N is the number of units in the network.

Boltzmann machines also use symmetric weights, but include units that are neither input nor output units (cf. the units labelled H_3 and H_4 in Figure 19.7). They also use a *stochastic* activation function, such that the probability of the output being 1 is some function of the total weighted input. Boltzmann machines therefore undergo state transitions that resemble a simulated annealing search for the configuration that best approximates the training set (see Chapter 4). It turns out that Boltzmann machines are formally identical to a special case of belief networks evaluated with a stochastic simulation algorithm (see Section 15.4).

Returning to feed-forward networks, there is one more important distinction to be made. Examine Figure 19.7, which shows the topology of a very simple neural network. On the left are the **input units**. The activation value of each of these units is determined by the environment. At the right-hand end of the network are four **output units**. In between, the nodes labelled H_3 and H_4 have no direct connection to the outside world. These are called **hidden units**, because they cannot be directly observed by noting the input/output behavior of the network. Some networks, called **perceptrons**, have no hidden units. This makes the learning problem much simpler, but it means that perceptrons are very limited in what they can represent. Networks with one or more layers of hidden units are called **multilayer networks**. With one (sufficiently large) layer of hidden units, it is possible to represent any continuous function of the inputs; with two layers, even discontinuous functions can be represented.

With a fixed structure and fixed activation functions g , the functions representable by a feed-forward network are restricted to have a specific parameterized structure. The weights chosen for the network determine which of these functions is actually represented. For example, the network in Figure 19.7 calculates the following function:

$$\begin{aligned} a_5 &= g(W_{3,5}a_3 + W_{4,5}a_4) \\ &= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2)) \end{aligned} \quad (19.1)$$

where g is the activation function, and a_i is the output of node i . Notice that because the activation functions g are nonlinear, the whole network represents a complex nonlinear function. If you

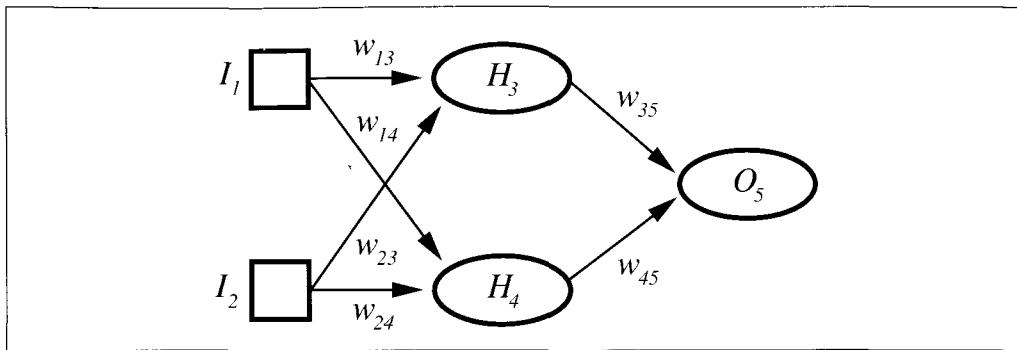


Figure 19.7 A very simple, two-layer, feed-forward network with two inputs, two hidden nodes, and one output node.



think of the weights as parameters or coefficients of this function, then *learning just becomes a process of tuning the parameters to fit the data in the training set*—a process that statisticians call **nonlinear regression**. From the statistical viewpoint, this is what neural networks do.

Optimal network structure

So far we have considered networks with a fixed structure, determined by some outside authority. This is a potential weak point, because the wrong choice of network structure can lead to poor performance. If we choose a network that is too small, then the model will be incapable of representing the desired function. If we choose a network that is too big, it will be able to memorize all the examples by forming a large lookup table, but will not generalize well to inputs that have not been seen before. In other words, like all statistical models, neural networks are subject to **overfitting** when there are too many parameters (i.e., weights) in the model. We saw this in Figure 18.2 (page 530), where the high-parameter models (b) and (c) fit all the data, but may not generalize as well as the low-parameter model (d).

It is known that a feed-forward network with one hidden layer can approximate any continuous function of the inputs, and a network with two hidden layers can approximate any function at all. However, the number of units in each layer may grow exponentially with the number of inputs. As yet, we have no good theory to characterize NERFs, or Network Efficiently Representable Functions—functions that can be approximated with a small number of units.

We can think of the problem of finding a good network structure as a search problem. One approach that has been used is to use a **genetic algorithm** (Chapter 20) to search the space of network structures. However, this is a very large space, and evaluating a state in the space means running the whole neural network training protocol, so this approach is very CPU-intensive. Therefore, it is more common to see hill-climbing searches that selectively modify an existing network structure. There are two ways to do this: start with a big network and make it smaller, or start with a small one and make it bigger.

The zip code reading network described on page 586 uses an approach called **optimal brain damage** to remove weights from the initial fully-connected model. After the network is

initially trained, an information theoretic approach identifies an optimal selection of connections that can be dropped (i.e., the weights are set to zero). The network is then retrained, and if it is performing as well or better, the process is repeated. This process was able to eliminate 3/4 of the weights, and improve overall performance on test data. In addition to removing connections, it is also possible to remove units that are not contributing much to the result.

Several algorithms have been proposed for growing a larger network from a smaller one. The **tiling algorithm** (Mézard and Nadal, 1989) is interesting because it is similar to the decision tree learning algorithm. The idea is to start with a single unit that does its best to produce the correct output on as many of the training examples as possible. Subsequent units are added to take care of the examples that the first unit got wrong. The algorithm adds only as many units as are needed to cover all the examples.

The **cross-validation techniques** of Chapter 18 are useful for deciding when we have found a network of the right size.

19.3 PERCEPTRONS

Layered feed-forward networks were first studied in the late 1950s under the name **perceptrons**. Although networks of all sizes and topologies were considered, the only effective learning element at the time was for single-layered networks, so that is where most of the effort was spent. Today, the name perceptron is used as a synonym for a single-layer, feed-forward network. The left-hand side of Figure 19.8 shows such a perceptron network. Notice that each output unit is independent of the others — each weight only affects one of the outputs. That means that we can limit our study to perceptrons with a single output unit, as in the right-hand side of Figure 19.8, and use several of them to build up a multi-output perceptron. For convenience, we can drop subscripts, denoting the output unit as O and the weight from input unit j to O as W_j . The activation of input unit j is given by I_j . The activation of the output unit is therefore

$$O = \text{Step}_0 \left(\sum_j W_j I_j \right) = \text{Step}_0(\mathbf{W} \cdot \mathbf{I}) \quad (19.2)$$

where, as discussed earlier, we have assumed an additional weight W_0 to provide a threshold for the step function, with $I_0 = -1$.

What perceptrons can represent

We saw in Figure 19.6 that units can represent the simple Boolean functions AND, OR, and NOT, and that therefore a feed-forward network of units can represent any Boolean function, if we allow for enough layers and units. But what Boolean functions can be represented with a single-layer perceptron?

Some complex Boolean functions can be represented. For example, the **majority function**, which outputs a 1 only if more than half of its n inputs are 1, can be represented by a perceptron with each $W_j = 1$ and threshold $t = n/2$. This would require a decision tree with $O(2^n)$ nodes.

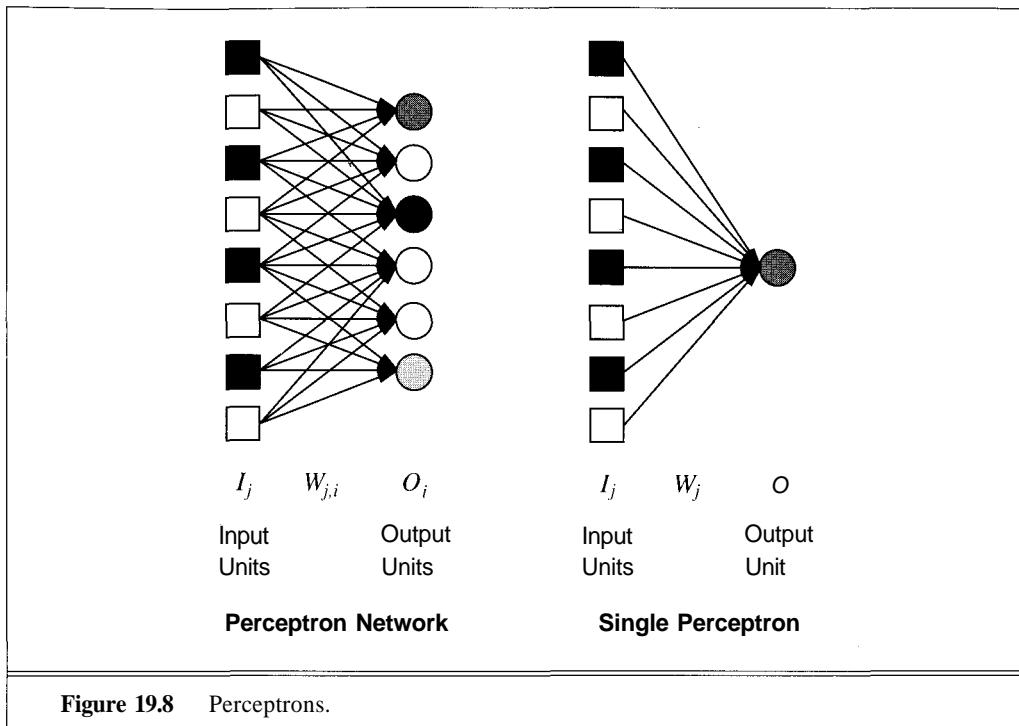


Figure 19.8 Perceptrons.

The perceptron, with 1 unit and n weights, gives a much more compact representation of this function. In accordance with Ockham's razor, we would expect the perceptron to do a much better job of learning a majority function, as we will soon see.

Unfortunately, it turns out that perceptrons are severely limited in the Boolean functions they can represent. The problem is that any input I_j can only influence the final output in one direction, no matter what the other input values are. Consider some input vector a . Suppose that this vector has $a_j = 0$ and that the vector produces a 0 as output. Furthermore, suppose that when a_j is replaced with 1, the output changes to 1. This implies that W_j must be positive. It also implies that there can be no input vector b for which the output is 1 when $b_j = 0$, but the output is 0 when b_j is replaced with 1. Because this limitation applies to each input, the result is a severe limitation in the total number of functions that can be represented. For example, the perceptron is unable to represent the function for deciding whether or not to wait for a table at a restaurant (shown as a decision tree in Figure 18.4).

A little geometry helps make clear what is going on. Figure 19.9 shows three different Boolean functions of two inputs, the AND, OR, and XOR functions. Each function is represented as a two-dimensional plot, based on the values of the two inputs. Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. As we will explain shortly, a perceptron can represent a function only if there is some line that separates all the white dots from the black dots. Such functions are called **linearly separable**. Thus, a perceptron can represent AND and OR, but not XOR.

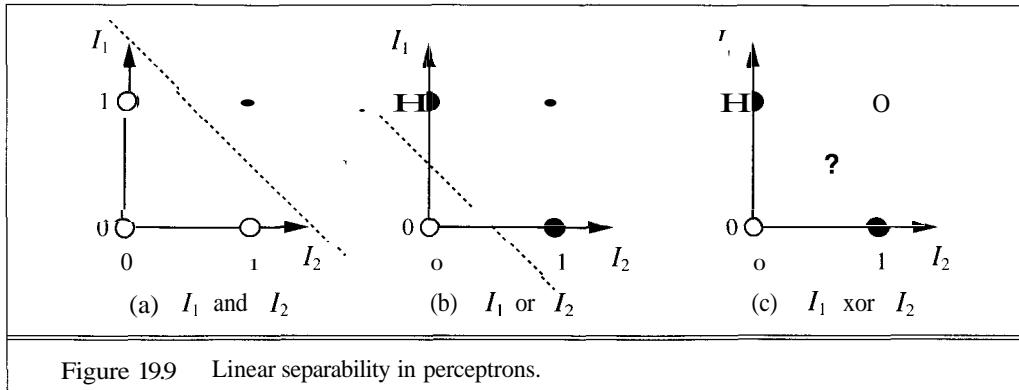


Figure 19.9 Linear separability in perceptrons.

The fact that a perceptron can only represent linearly separable functions follows directly from Equation (19.2), which defines the function computed by a perceptron. A perceptron outputs a 1 only if $\mathbf{W} \cdot \mathbf{I} > 0$. This means that the entire input space is divided in two along a boundary defined by $\mathbf{W} \cdot \mathbf{I} = 0$, that is, a plane in the input space with coefficients given by the weights. With n inputs, the input space is n -dimensional, and linear separability can be rather hard to visualize if n is too large. It is easiest to understand for the case where $n = 2$. In Figure 19.9(a), one possible separating "plane" is the dotted line defined by the equation

$$I_1 = -I_2 + 1.5 \quad \text{or} \quad I_1 + I_2 = 1.5$$

The region above the line, where the output is 1, is therefore given by

$$-1.5 + I_1 + I_2 > 0$$

or, in vector notation,

$$\mathbf{W} \cdot \mathbf{I} = \langle 1.5, 1, 1 \rangle \cdot \langle -1, I_1, I_2 \rangle > 0$$

With three inputs, the separating plane can still be visualized. Figure 19.10(a) shows an example in three dimensions. The function we are trying to represent is true if and only if a minority of its three inputs are true. The shaded separating plane is defined by the equation

$$I_1 + /2 + /3 = 1.5$$

This time the positive outputs lie below the plane, in the region

$$(-I_1) + (-/2) + (-/3) > -1.5$$

Figure 19.10(b) shows a unit to implement the function.

Learning linearly separable functions

As with any performance element, the question of what perceptrons can represent is prior to the question of what they can learn. We have just seen that a function can be represented by a perceptron if and only if it is linearly separable. That is relatively bad news, because there are not many linearly separable functions. The (relatively) good news is that *there is a perceptron algorithm that will learn any linearly separable function, given enough training examples.*



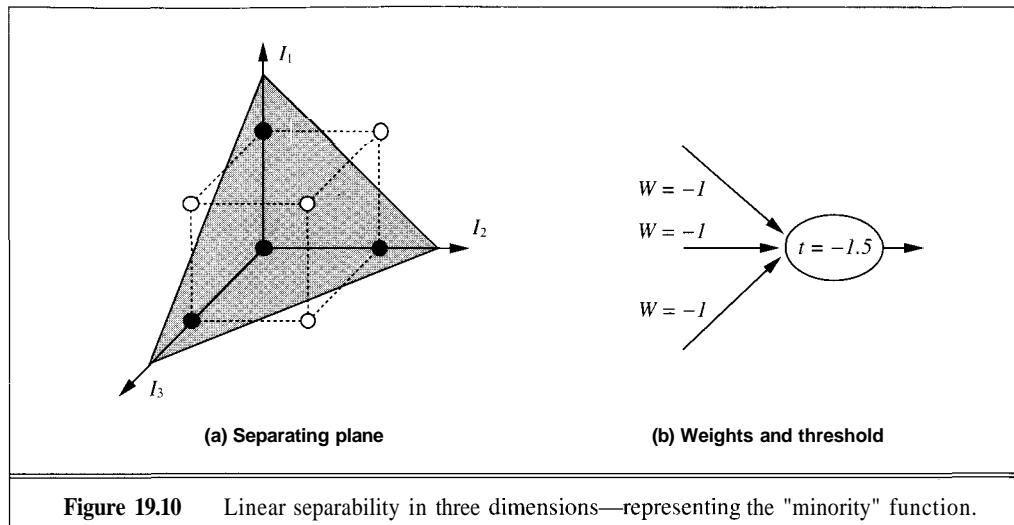


Figure 19.10 Linear separability in three dimensions—representing the "minority" function.

Most neural network learning algorithms, including the perceptron learning method, follow the current-best-hypothesis (CBH) scheme described in Chapter 18. In this case, the hypothesis is a network, defined by the current values of the weights. The initial network has randomly assigned weights, usually from the range $[-0.5, 0.5]$. The network is then updated to try to make it consistent with the examples. This is done by making small adjustments in the weights to reduce the difference between the observed and predicted values. The main difference from the logical algorithms is the need to repeat the update phase several times for each example in order to achieve convergence. Typically, the updating process is divided into **epochs**. Each epoch involves updating all the weights for all the examples. The general scheme is shown as NEURAL-NETWORK-LEARNING in Figure 19.11.

For perceptrons, the weight update rule is particularly simple. If the predicted output for the single output unit is O , and the correct output should be T , then the error is given by

$$Err = T - O$$

If the error is positive, then we need to increase O ; if it is negative, we need to decrease O . Now each input unit contributes $W_j I_j$ to the total input, so if I_j is positive, an increase in W_j will tend to increase O , and if I_j is negative, an increase in W_j will tend to decrease O . Thus, we can achieve the effect we want with the following rule:

$$Wj \leftarrow Wj + a \times lj \times Err$$

where the term a is a constant called the **learning rate**. This rule is a slight variant of the **perceptron learning rule** proposed by Frank Rosenblatt in 1960. Rosenblatt proved that a learning system using the perceptron learning rule will converge to a set of weights that correctly represents the examples, as long as the examples represent a linearly separable function.

The perceptron convergence theorem created a good deal of excitement when it was announced. People were amazed that such a simple procedure could correctly learn any representable function, and there were great hopes that intelligent machines could be built from

```
function NEURAL-NETWORK-LEARNING(examples) returns network
    network  $\leftarrow$  a network with randomly assigned weights
    repeat
        for each e in examples do
            O  $\leftarrow$  NEURAL-NETWORK-OUTPUT(network, e)
            T  $\leftarrow$  the observed output values from e
            update the weights in network based on e, O, and T
        end
    until all examples correctly predicted or stopping criterion is reached
    return network
```

Figure 19.11 The generic neural network learning method: adjust the weights until predicted output values *O* and true values *T* agree.

perceptrons. It was not until 1969 that Minsky and Papert undertook what should have been the first step: analyzing the class of representable functions. Their book *Perceptrons* (Minsky and Papert, 1969) clearly demonstrated the limits of linearly separable functions.

In retrospect, the perceptron convergence theorem should not have been surprising. The perceptron is doing a **gradient descent** search through weight space (see Chapter 4). It is fairly easy to show that the weight space has no local minima. Provided the learning rate parameter is not so large as to cause "overshooting," the search will converge on the correct weights. In short, perceptron learning is easy because the space of representable functions is simple.

We can examine the learning behavior of perceptrons using the method of constructing learning curves, as described in Chapter 18. There is a slight difference between the example descriptions used for neural networks and those used for other attribute-based methods such as decision trees. In a neural network, all inputs are real numbers in some fixed range, whereas decision trees allow for multivalued attributes with a discrete set of values. For example, the attribute for the number of patrons in the restaurant has values *None*, *Some*, and *Full*. There are two ways to handle this. In a **local encoding**, we use a single input unit and pick an appropriate number of distinct values to correspond to the discrete attribute values. For example, we can use *None* = 0.0, *Some* = 0.5, and *Full* = 1.0. In a **distributed encoding**, we use one input unit for each value of the attribute, turning on the unit that corresponds to the correct value.

Figure 19.12 shows the learning curve for a perceptron on two different problems. On the left, we show the curve for learning the majority function with 11 Boolean inputs (i.e., the function outputs a 1 if 6 or more inputs are 1). As we would expect, the perceptron learns the function quite quickly because the majority function is linearly separable. On the other hand, the decision tree learner makes no progress, because the majority function is very hard (although not impossible) to represent as a decision tree. On the right of the figure, we have the opposite situation. The *WillWait* problem is easily represented as a decision tree, but is not linearly separable. The perceptron algorithm draws the best plane it can through the data, but can manage no more than 65% accuracy.

LOCAL ENCODING

DISTRIBUTED
ENCODING

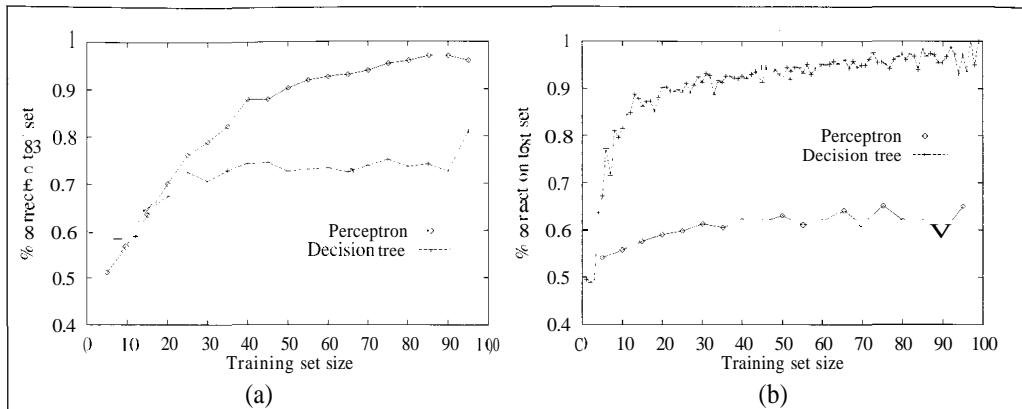


Figure 19.12 Comparing the performance of perceptrons and decision trees. (a) Perceptrons are better at learning the majority function of 11 inputs. (b) Decision trees are better at learning the *WillWait* predicate for the restaurant example.

19.4 MULTILAYER FEED-FORWARD NETWORKS

Rosenblatt and others described multilayer feed-forward networks in the late 1950s, but concentrated their research on single-layer perceptrons. This was mainly because of the difficulty of finding a sensible way to update the weights between the inputs and the hidden units; whereas an error signal can be calculated for the output units, it is harder to see what the error signal should be for the hidden units. When the book *Perceptrons* was published, Minsky and Papert (1969) stated that it was an "important research problem" to investigate multilayer networks more thoroughly, although they speculated that "there is no reason to suppose that any of the virtues [of perceptrons] carry over to the many-layered version." In a sense, they were right. Learning algorithms for multilayer networks are neither efficient nor guaranteed to converge to a global optimum. On the other hand, the results of computational learning theory tell us that learning general functions from examples is an intractable problem in the worst case, regardless of the method, so we should not be too dismayed.

The most popular method for learning in multilayer networks is called **back-propagation**. It was first invented in 1969 by Bryson and Ho, but was more or less ignored until the mid-1980s. The reasons for this may be sociological, but may also have to do with the computational requirements of the algorithm on nontrivial problems.

Back-Propagation Learning

Suppose we want to construct a network for the restaurant problem. We have already seen that a perceptron is inadequate, so we will try a two-layer network. We have ten attributes

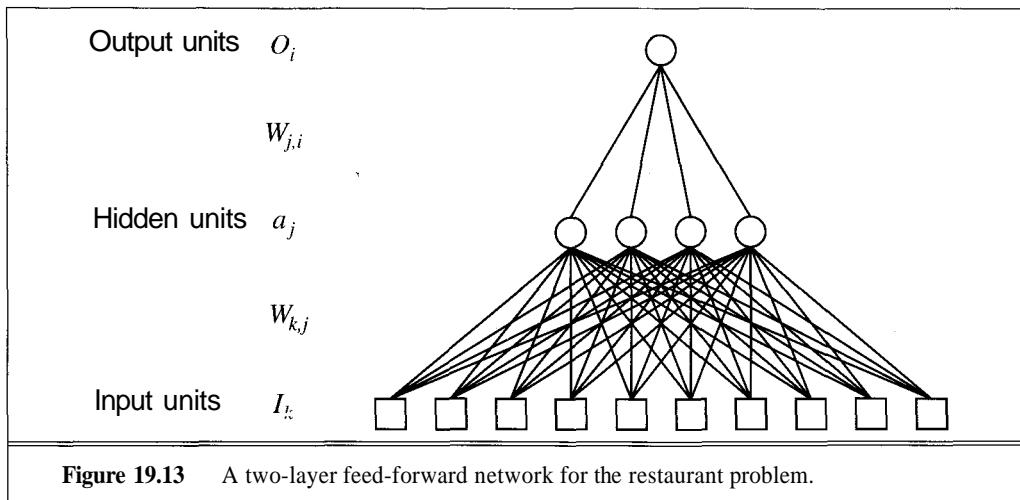


Figure 19.13 A two-layer feed-forward network for the restaurant problem.

describing each example, so we will need ten input units. How many hidden units are needed? In Figure 19.13, we show a network with four hidden units. This turns out to be about right for this problem. The problem of choosing the right number of hidden units in advance is still not well-understood. We cover what is known on page 572.

Learning in such a network proceeds the same way as for perceptrons: example inputs are presented to the network, and if the network computes an output vector that matches the target, nothing is done. If there is an error (a difference between the output and target), then the weights are adjusted to reduce this error. *The trick is to assess the blame for an error and divide it among the contributing weights.* In perceptrons, this is easy, because there is only one weight between each input and the output. But in multilayer networks, there are many weights connecting each input to an output, and each of these weights contributes to more than one output.

The back-propagation algorithm is a sensible approach to dividing the contribution of each weight. As in the perceptron learning algorithm, we try to minimize the error between each target output and the output actually computed by the network.⁴ At the output layer, the weight update rule is very similar to the rule for the perceptron. There are two differences: the activation of the hidden unit a_j is used instead of the input value; and the rule contains a term for the gradient of the activation function. If Err_i is the error ($T_i - O_i$) at the output node, then the weight update rule for the link from unit j to unit i is

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err_i \times g'(in_i)$$

where g' is the derivative of the activation function g . We will find it convenient to define a new error term Δ_i , which for output nodes is defined as $\Delta_i = Err_i g'(in_i)$. The update rule then becomes

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \quad (19.3)$$

For updating the connections between the input units and the hidden units, we need to define a quantity analogous to the error term for output nodes. Here is where we do the error back-propagation. The idea is that hidden node j is "responsible" for some fraction of the error Δ_i in

⁴ Actually, we minimize the square of the error; Section 19.4 explains why, but the result is almost the same.

each of the output nodes to which it connects. Thus, the Δ_i values are divided according to the strength of the connection between the hidden node and the output node, and propagated back to provide the Δ_j values for the hidden layer. The propagation rule for the A values is the following:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \quad .(19.4)$$

Now the weight update rule for the weights between the inputs and the hidden layer is almost identical to the update rule for the output layer:

$$W_{k,j} \leftarrow W_{k,j} + Q \times I_k \times \Delta_j$$

The detailed algorithm is shown in Figure 19.14. It can be summarized as follows:

- Compute the A values for the output units using the observed error.
- Starting with output layer, repeat the following for each layer in the network, until the earliest hidden layer is reached:
 - Propagate the A values back to the previous layer.
 - Update the weights between the two layers.

Recall that in computing the observed error for a given example, NEURAL-NETWORK-LEARNING first feeds the example to the network inputs in order to calculate the predicted output values. During this computation, it is a good idea to save some of the intermediate values computed in each unit. In particular, caching the activation gradient $g'(in_i)$ in each unit speeds up the subsequent back-propagation phase enormously.

Now that we have a learning method for multilayer networks, we can test our claim that adding a hidden layer makes the network more expressive. In Figure 19.15, we show two curves. The first is a **training curve**, which shows the mean squared error on a given training set of 100 restaurant examples during the weight-updating process. This demonstrates that the network does indeed converge to a perfect fit to the training data. The second curve is the standard learning curve for the restaurant data, with one minor exception: the y-axis is no longer the proportion of correct answers on the test set, because sigmoid units do not give 0/1 outputs. Instead, we use the mean squared error on the test set, which happens to coincide with the proportion of correct answers in the 0/1 case. The curve clearly shows that the network is capable of learning in the restaurant domain; indeed, the curve is very similar to that for decision-tree learning, albeit somewhat shallower.

TRAINING CURVE

ERROR SURFACE

Back-propagation as gradient descent search

We have given some suggestive reasons why the back-propagation equations are reasonable. It turns out that the equations also can be given a very simple interpretation as a method for performing **gradient descent** in weight space. In this case, the gradient is on the **error surface**: the surface that describes the error on each example as a function of all the weights in the network. An example error surface is shown in Figure 19.16. The current set of weights defines a point on this surface. At that point, we look at the slope of the surface along the axis formed by each weight. This is known as the *partial derivative* of the surface with respect to each weight—how much the error would change if we made a small change in weight. We then alter

```

function BACK-PROP-UPDATE(network, examples, a) returns a network with modified weights
  inputs: network, a multilayer network
    examples, a set of input/output pairs
     $\alpha$ , the learning rate

  repeat
    for each e in examples do
      /* Compute the output for this example */
       $O \leftarrow \text{RUN-NETWORK}(\text{network}, I^e)$ 
      /* Compute the error and  $\Delta$  for units in the output layer */
       $\text{Err}^e \leftarrow T^e - O$ 
      /* Update the weights leading to the output layer */
       $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \text{Err}^e \times g'(in_i)$ 
    for each subsequent layer in network do
      /* Compute the error at each node */
       $\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$ 
      /* Update the weights leading into the layer */
       $W_{k,j} \leftarrow W_{k,j} + \alpha \times a \times I_k \times A_y$ 
    end
  end
  until network has converged
  return network

```

Figure 19.14 The back-propagation algorithm for updating weights in a multilayer network.

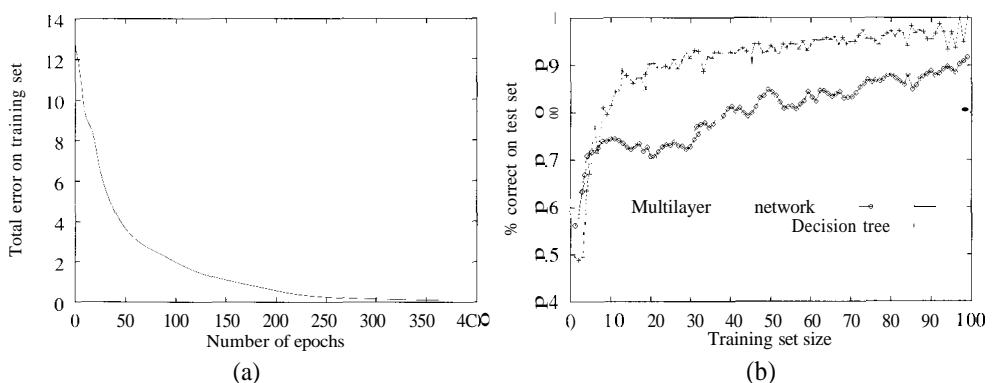
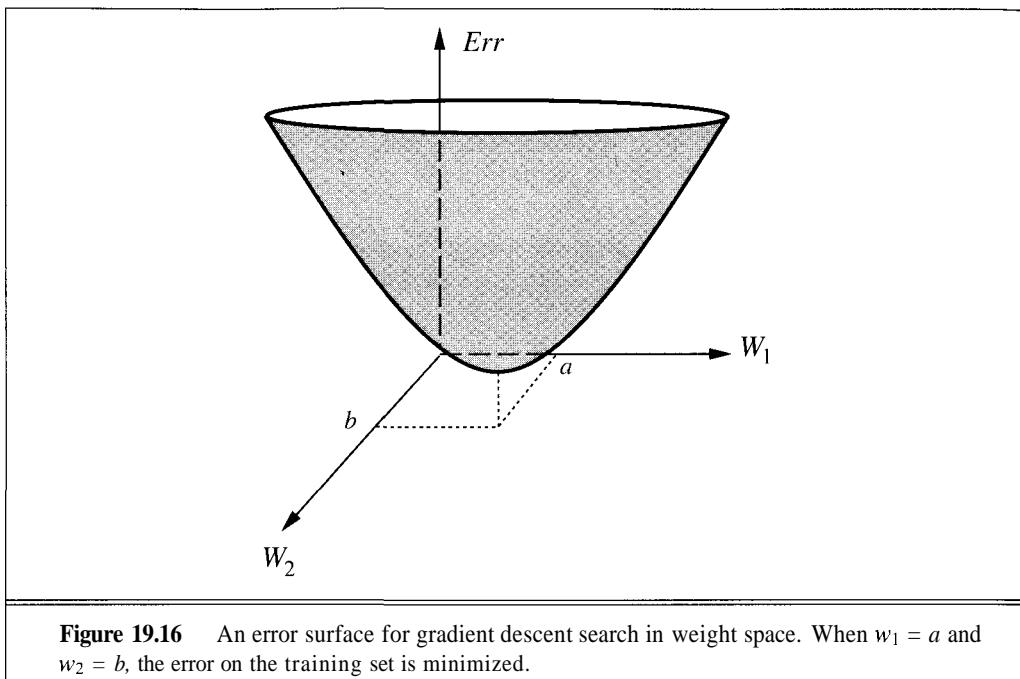


Figure 19.15 (a) Training curve showing the gradual reduction in error as weights are modified over several epochs, for a given set of examples in the restaurant domain. (b) Comparative learning curves for a back-propagation and decision-tree learning.



the weights in an amount proportional to the slope in each direction. This moves the network as a whole in the direction of steepest descent on the error surface.



Basically, this is the key: *back-propagation provides a way of dividing the calculation of the gradient among the units, so the change in each weight can be calculated by the unit to which the weight is attached, using only local information.* Like any gradient descent search, back-propagation has problems with efficiency and convergence, as we will discuss shortly. Nonetheless, the decomposition of the learning algorithm is a major step towards parallelizable and biologically plausible learning mechanisms.

For the mathematically inclined, we will now derive the back-propagation equations from first principles. We begin with the error function itself. Because of its convenient form, we use the sum of squared errors over the output values:

$$E = \frac{1}{2} \sum_i (T_i - O_i)^2$$

The key insight, again, is that the output values O_i are a function of the weights (see Equation (19.1), for example). For a general two-layer network, we can write

$$\begin{aligned} E(\mathbf{W}) &= \frac{1}{2} \sum_i \left(T_i - g \left(\sum_j W_{j,i} a_j \right) \right)^2 \\ &= \frac{1}{2} \sum_i \left(T_i - g \left(\sum_j W_{j,i} g \left(\sum_k W_{k,j} I_k \right) \right) \right)^2 \end{aligned} \tag{19.5}$$

Notice that although the a_j term in the first line represents a complex expression, it does not depend on $W_{j,i}$. Also, only one of the terms in the summation over i and j depends on a particular $W_{j,i}$, so all the other terms are treated as constants with respect to $W_{j,i}$ and will disappear when differentiated. Hence, when we differentiate the first line with respect to $W_{j,i}$, we obtain

$$\begin{aligned}\frac{\partial E}{\partial W_{j,i}} &= -a_j(T_i - O_i)g' \left(\sum_j W_{j,i}a_j \right) \\ &= -a_j(T_i - O_i)g'(in_i) = -a_j\Delta_i\end{aligned}$$

with Δ_i defined as before. The derivation of the gradient with respect to $W_{k,j}$ is slightly more complex, but has a similar result:

$$\frac{dE}{\partial W_{k,j}} = -I_k\Delta_j$$

To obtain the update rules for the weights, we have to remember that the object is to *minimize* the error, so we need to take a small step in the direction *opposite* to the gradient.

There is one minor technical observation to make about these update rules. They require the derivative of the activation function g , so we cannot use the sign or step functions. Back-propagation networks usually use the sigmoid function, or some variant thereof. The sigmoid also has the convenient property that the derivative $g' = g(1 - g)$, so that little extra calculation is needed to find $g'(in_i)$.

Discussion

Let us stand back for a moment from the delightful mathematics and the fascinating biology, and ask whether back-propagation learning in multilayer networks is a good method for machine learning. We can examine the same set of issues that arose in Chapter 18:

- ◊ **Expressiveness:** Neural networks are clearly an attribute-based representation, and do not have the expressive power of general logical representations. They are well-suited for continuous inputs and outputs, unlike most decision tree systems. The class of multilayer networks *as a whole* can represent any desired function of a set of attributes, but any *particular* network may have too few hidden units. It turns out that $2^n/n$ hidden units are needed to represent all Boolean functions of n inputs. This should not be too surprising. Such a network has $O(2^n)$ weights, and we need at least 2^n bits to specify a Boolean function. In practice, most problems can be solved with many fewer weights. Designing a good topology is, however, a black art.
- ◊ **Computational efficiency:** Computational efficiency depends on the amount of computation time required to train the network to fit a given set of examples. If there are m examples, and $|W|$ weights, each epoch takes $O(m|W|)$ time. However, work in computational learning theory has shown that the worst-case number of epochs can be exponential in n , the number of inputs. In practice, time to convergence is highly variable, and a vast array of techniques have been developed to try to speed up the process using an assortment of tunable parameters. Local minima in the error surface are also a problem. Networks quite often converge to give a constant "yes" or "no" output, whichever is most common

in the training set. At the cost of some additional computation, the simulated annealing method (Chapter 4) can be used to assure convergence to a global optimum.

- 0 Generalization:** As we have seen in our experiments on the restaurant data, neural networks can do a good job of generalization. One can say, somewhat circularly, that they will generalize well on functions for which they are well-suited. These seem to be functions in which the interactions between inputs are not too intricate, and for which the output varies smoothly with the input. There is no theorem to be proved here, but it does seem that neural networks have had reasonable success in a number of real-world problems.
- ◊ **Sensitivity to noise:** Because neural networks are essentially doing nonlinear regression, they are very tolerant of noise in the input data. They simply find the best fit given the constraints of the network topology. On the other hand, it is often useful to have some idea of the degree of certainty of the output values. Neural networks do not provide probability distributions on the output values. For this purpose, belief networks seem more appropriate.
 - ◊ **Transparency:** Neural networks are essentially black boxes. Even if the network does a good job of predicting new cases, many users will still be dissatisfied because they will have no idea *why* a given output value is reasonable. If the output value represents, for example, a decision to perform open heart surgery, then an explanation is clearly in order. With decision trees and other logical representations, the output can be explained as a logical derivation and by appeal to a specific set of cases that supports the decision. This is not currently possible with neural networks.
 - ◊ **Prior knowledge:** As we mentioned in Chapter 18, learning systems can often benefit from prior knowledge that is available to the user or expert. Prior knowledge can mean the difference between learning from a few well-chosen examples and failing to learn anything at all. Unfortunately, because of the lack of transparency, it is quite hard to use one's knowledge to "prime" a network to learn better. Some tailoring of the network topology can be done—for example, when training on visual images it is common to connect only small sets of nearby pixels to any given unit in the first hidden layer. On the other hand, such "rules of thumb" do not constitute a *mechanism* by which previously accumulated knowledge can be used to learn from subsequent experience. It is possible that learning methods for belief networks can overcome this problem (see Section 19.6).

All these considerations suggest that simple feed-forward networks, although very promising as construction tools for learning complex input/output mappings, do not fulfill our needs for a comprehensive theory of learning in their present form. Researchers in AI, psychology, theoretical computer science, statistics, physics, and biology are working hard to overcome the difficulties.

19.5 APPLICATIONS OF NEURAL NETWORKS

In this section, we give just a few examples of the many significant applications of neural networks. In each case, the network design was the result of several months of trial-and-error experimentation by researchers. From these examples, it can be seen that neural networks have

wide applicability, but that they cannot magically solve problems without any thought on the part of the network designer. John Denker's remark that "neural networks are the second best way of doing just about anything" may be an exaggeration, but it is true that neural networks provide passable performance on many tasks that would be difficult to solve explicitly with other programming techniques. We encourage the reader to experiment with neural network algorithms to get a feel for what happens when data arrive at an unprepared network.

Pronunciation

Pronunciation of written English text by a computer is a fascinating problem in linguistics, as well as a task with high commercial payoff. It is typically carried by first mapping the text stream to **phonemes**—basic sound elements—and then passing the phonemes to an electronic speech generator. The problem we are concerned with here is learning the mapping from text to phonemes. This is a good task for neural networks because most of the "rules" are only approximately correct. For example, although the letter "k" usually corresponds to the sound [k], the letter "c" is pronounced [k] in *cat* and [s] in *cent*.

The NETtalk program (Sejnowski and Rosenberg, 1987) is a neural network that learns to pronounce written text. The input is a sequence of characters presented in a window that slides through the text. At any time, the input includes the character to be pronounced along with the preceding and following three characters. Each character is actually 29 input units—one for each of the 26 letters, and one each for blanks, periods, and other punctuation. There were 80 hidden units in the version for which results are reported. The output layer consists of features of the sound to be produced: whether it is high or low, voiced or unvoiced, and so on. Sometimes, it takes two or more letters to produce a single sound; in this case, the correct output for the second letter is nothing.

Training consisted of a 1024-word text that had been hand-transcribed into the proper phonemic features. NETtalk learns to perform at 95% accuracy *on the training set* after 50 passes through the training data. One might think that NETtalk should perform at 100% on the text it has trained on. But any program that learns individual words rather than the entire text as a whole will inevitably score less than 100%. The difficulty arises with words like *lead*, which in some cases should be pronounced to rhyme with *bead* and sometimes like *bed*. A program that looks at only a limited window will occasionally get such words wrong.

So much for the ability of the network to reproduce the training data. What about the generalization performance? This is somewhat disappointing. On the test data, NETtalk's accuracy goes down to 78%, a level that is intelligible, but much worse than commercially available programs. Of course, the commercial systems required years of development, whereas NETtalk only required a few dozen hours of training time plus a few months of experimentation with various network designs. However, there are other techniques that require even less development and perform just as well. For example, if we use the input to determine the probability of producing a particular phoneme given the current and previous character and then use a Markov model to find the sequence of phonemes with maximal probability, we do just as well as NETtalk.

NETtalk was perhaps the "flagship" demonstration that converted many scientists, particularly in cognitive psychology, to the cause of neural network research. A post hoc analysis

suggests that this was not because it was a particularly successful program, but rather because it provided a good showpiece for the philosophy of neural networks. Its authors also had a flair for the dramatic: they recorded a tape of NETtalk starting out with poor, babbling speech, and then gradually improving to the point where the output is understandable. Unlike conventional speech generators, which use a midrange tenor voice to generate the phonemes, they used a high-pitched generator. The tape gives the unmistakable impression of a child learning to speak.

FEATURE
DETECTORS

Handwritten character recognition

In one of the largest applications of neural networks to date, Le Cun *et al.* (1989) have implemented a network designed to read zip codes on hand-addressed envelopes. The system uses a preprocessor that locates and segments the individual digits in the zipcode; the network has to identify the digits themselves. It uses a 16×16 array of pixels as input, *three* hidden layers, and a distributed output encoding with 10 output units for digits 0-9. The hidden layers contained 768, 192, and 30 units, respectively. A fully connected network of this size would contain 200,000 weights, and would be impossible to train. Instead, the network was designed with connections intended to act as **feature detectors**. For example, each unit in the first hidden layer was connected by 25 links to a 5×5 region in the input. Furthermore, the hidden layer was divided into 12 groups of 64 units; within each group of 64 units, each unit used the *same* set of 25 weights. Hence the hidden layer can detect up to 12 distinct features, each of which can occur anywhere in the input image. Overall, the complete network used only 9760 weights.

The network was trained on 7300 examples, and tested on 2000. One interesting property of a network with distributed output encoding is that it can display confusion over the correct answer by setting two or more output units to a high value. After rejecting about 12% of the test set as marginal, using a confusion threshold, the performance on the remaining cases reached 99%, which was deemed adequate for an automated mail-sorting system. The final network has been implemented in custom VLSI, enabling letters to be sorted at high speed.

Driving

ALVINN (Autonomous Land Vehicle In a Neural Network) (Pomerleau, 1993) is a neural network that has performed quite well in a domain where some other approaches have failed. It learns to steer a vehicle along a single lane on a highway by observing the performance of a human driver. We described the system briefly on page 26, but here we take a look under the hood.

ALVINN is used to control the NavLab vehicles at Carnegie Mellon University. NavLab 1 is a Chevy van, and NavLab 2 is a U.S. Army HMMWV personnel carrier. Both vehicles are specially outfitted with computer-controlled steering, acceleration, and braking. Sensors include color stereo video, scanning laser range finders, radar, and inertial navigation. Researchers ride along in the vehicle and monitor the progress of the computer and the vehicle itself. (Being inside the vehicle is a big incentive to making sure the program does not “crash.”)

The signal from the vehicle’s video camera is preprocessed to yield an array of pixel values that are connected to a 30×32 grid of input units in a neural network. The output is a layer of 30 units, each corresponding to a steering direction. The output unit with the highest activation

is the direction that the vehicle will steer. The network also has a layer of five hidden units that are fully connected to the input and output layers.

ALVINN'S job is to compute a function that maps from a single video image of the road in front of it to a steering direction. To learn this function, we need some training data—some image/direction pairs with the correct direction. Fortunately, it is easy to collect this data just by having a human drive the vehicle and recording the image/direction pairs. After collecting about five minutes of training data (and applying the back-propagation algorithm for about ten minutes), ALVINN is ready to drive on its own.

One fine point is worth mentioning. There is a potential problem with the methodology of training based on a human driver: the human is too good. If the human never strays from the proper course then there will be no training examples that show how to recover when you are off course. ALVINN corrects this problem by rotating each video image to create additional views of what the road would look like from a position a little to the right or left.

The results of the training are impressive. ALVINN has driven at speeds up to 70 mph for distances up to 90 miles on public highways near Pittsburgh. It has also driven at normal speeds on single lane dirt roads, paved bike paths, and two lane suburban streets.

ALVINN is unable to drive on a road type for which it has not been trained, and is also not very robust with respect to changes in lighting conditions or the presence of other vehicles. A more general capability is exhibited by the MANIAC system (Jochem *et al.*, 1993). MANIAC is a neural network that has as subnets two or more ALVINN models that have each been trained for a particular type of road. MANIAC takes the output from each subnet and combines them in a second hidden layer. With suitable training, MANIAC can perform well on any of the road types for which the component subnets have been trained.

Some previous autonomous vehicles employed traditional vision algorithms that used various image-processing techniques on the entire scene in order to find the road and then follow it. Such systems achieved top speeds of 3 or 4 mph.⁵ Why has ALVINN proven to be successful? There are two reasons. First and foremost, a neural network of this size makes an efficient performance element. Once it has been trained, ALVINN is able to compute a new steering direction from a video image 10 times a second. This is important because it allows for some slack in the system. Individual steering directions can be off by 10% from the ideal as long as the system is able to make a correction in a few tenths of a second. Second, the use of a learning algorithm is more appropriate for this domain than knowledge engineering or straight programming. There is no good existing theory of driving, but it is easy to collect sample input/output pairs of the desired functional mapping. This argues for a learning algorithm, but not necessarily for neural nets. But driving is a continuous, noisy domain in which almost all of the input features contribute some useful information; this means that neural nets are a better choice than, say, decision trees. Of course, ALVINN and MANIAC are pure reflex agents, and cannot execute maneuvers that are much more complex than lane-following, especially in the presence of other traffic. Current research by Pomerleau and other members of the group is aimed at combining ALVINN'S low-level expertise with higher-level symbolic knowledge. Hybrid systems of this kind are becoming more common as AI moves into the real (physical) world.

⁵ A notable exception is the work by Dickmanns and Zapp (1987), whose autonomous vehicle drove several hundred miles at 75 mph using traditional image processing and Kalman filtering to track the lane boundaries.

19.6 BAYESIAN METHODS FOR LEARNING BELIEF NETWORKS

Part V made the case for the importance of probabilistic representations of uncertain knowledge, and presented belief networks as a general and useful performance element based on probability theory. In this section, we discuss the general problem of learning probabilistic knowledge, and the specific problem of learning belief networks. We will see that a Bayesian view of learning is extremely powerful, providing general solutions to the problems of noise, overfitting, and optimal prediction. We will also find striking parallels between belief networks and neural networks in their amenability to local, gradient-descent learning methods. Most of this section is fairly mathematical, although the general lessons can be understood without plunging into the details. It may be helpful at this point to review the material in Chapters 14 and 15.

Bayesian learning

BAYESIAN LEARNING

Bayesian learning views the problem of constructing hypotheses from data as a subproblem of the more fundamental problem of making predictions. The idea is to use hypotheses as intermediaries between data and predictions. First, the probability of each hypothesis is estimated, given the data. Predictions are then made from the hypotheses, using the posterior probabilities of the hypotheses to weight the predictions. As a simple example, consider the problem of predicting tomorrow's weather. Suppose the available experts are divided into two camps: some propose model A, and some propose model B. The Bayesian method, rather than choosing between A and B, gives some weight to each based on their likelihood. The likelihood will depend on how much the known data support each of the two models.

Suppose that we have data D and hypotheses H_1, H_2, \dots , and that we are interested in making a prediction concerning an unknown quantity X . Furthermore, suppose that each H_i specifies a complete distribution for X . Then we have

$$\mathbf{P}(X|D) = \underbrace{\sum_i}_{i} \mathbf{P}(X|D, H_i) \mathbf{P}(H_i|D) = \mathbf{P}(X|H_i) \mathbf{P}(H_i|D)$$

This equation describes full Bayesian learning, and may require a calculation of $\mathbf{P}(H_i|D)$ for all H_i . In most cases, this is intractable; it can be shown, however, that there is no better way to make predictions.

The most common approximation is to use a *most probable* hypothesis, that is, an H_i that maximizes $\mathbf{P}(H_i|D)$. This often called a **maximum a posteriori** or MAP hypothesis H_{MAP} :

$$\mathbf{P}(X|D) \approx \mathbf{P}(X|H_{\text{MAP}}) \mathbf{P}(H_{\text{MAP}}|D)$$

The problem is now to find H_{MAP} . By applying Bayes' rule, we can rewrite $\mathbf{P}(H_i|D)$ as follows:

$$\mathbf{P}(H_i|D) = \frac{\mathbf{P}(D|H_i) \mathbf{P}(H_i)}{\mathbf{P}(D)}$$

Notice that in comparing hypotheses, $P(D)$ remains fixed. Hence, to find H_{MAP} , we need only maximize the numerator of the fraction.

The first term, $\mathbf{P}(D|H_i)$, represents the probability that this particular data set would have been observed, given H_i as the underlying model of the world. The second term represents the

prior probability assigned to the given model. Arguments over the nature and significance of this prior probability distribution, and its relation to preference for simpler hypotheses (Ockham's razor), have raged unchecked in the statistics and learning communities for decades. The only reasonable policy seems to be to assign prior probabilities based on some simplicity measure on hypotheses, such that the prior of the entire hypothesis space adds up to 1. The more we bias the priors towards simpler hypotheses, the more we will be immune to noise and overfitting. Of course, if the priors are *too* biased, then we get underfitting, where the data is largely ignored. There is a careful trade-off to make.

In some cases, a **uniform** prior over belief networks seems to be appropriate, as we shall see. With a uniform prior, we need only choose an H_i that maximizes $P(D|H_i)$. This is called a **maximum-likelihood** (ML) hypothesis, H_{ML} .

MAXIMUM-LIKELIHOOD

Belief network learning problems

The learning problem for belief networks comes in several varieties. The structure of the network can be *known* or *unknown*, and the variables in the network can be *observable* or *hidden*.

- 0 **Known structure, fully observable:** In this case, the only learnable part is the set of conditional probability tables. These can be estimated directly using the statistics of the set of examples. Some belief network systems incorporate automatic updating of conditional probability table entries to reflect the cases seen.
- ◊ **Unknown structure, fully observable:** In this case, the problem is to reconstruct the topology of the network. This problem can be cast as a search through the space of structures, guided by the ability of each structure to model the data correctly. Fitting the data to a given structure reduces to the fixed-structure problem, and the MAP or ML probability value can be used as heuristic for hill-climbing or simulated annealing search.
- ◊ **Known structure, hidden variables:** This case is analogous to neural network learning. We discuss methods for this problem in the next section.
- ◊ **Unknown structure, hidden variables:** When some variables are sometimes or always unobservable, the prior techniques for recovering structure become difficult to apply, because they essentially require averaging over all possible combinations of values of the unknown variables. At present, no good, general algorithms are known for this problem.

Learning networks with fixed structure

Experience in constructing belief networks for applications has shown that finding the topology of the network is often the easy part. Humans find it easy to say what causes what, but hard to put exact numbers on the links. This is particularly true when some of the variables cannot be observed directly in actual cases. The "known structure, hidden variable" learning problem is therefore of great importance.

One might ask why the problem cannot be reduced to the fully observable case by eliminating the hidden variables using marginalization ("averaging out"). There are two reasons for this. First, it is not necessarily the case that any particular variable is hidden in all the observed

cases (although we do not rule this out). Second, networks with hidden variables can be more *compact* than the corresponding fully observable network. Figure 19.17 shows an example. If the underlying domain has significant local structure, then with hidden variables it is possible to take advantage of that structure to find a more concise representation for the joint distribution on the observable variables. This, in turn, makes it possible to learn from fewer examples.

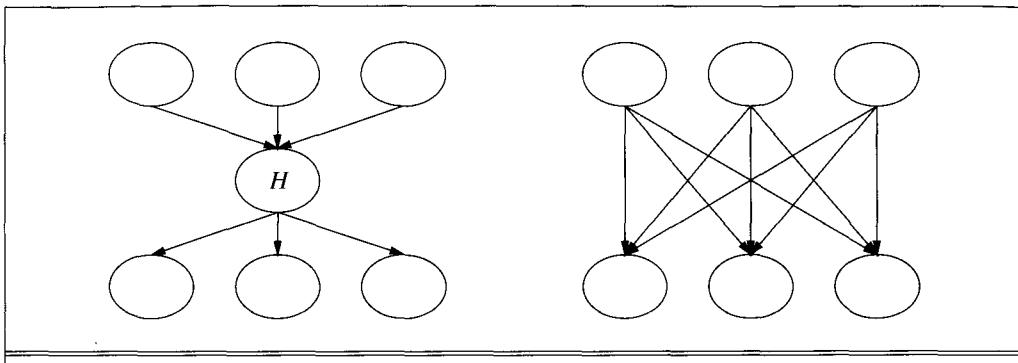


Figure 19.17 A network with a hidden variable (labelled H), and the corresponding fully observable network. If the variables are Boolean, then the hidden-variable network requires 17 independent conditional probability values, whereas the fully observable network requires 27.

If we are to approach this problem in Bayesian terms, then the "hypotheses" H_i are the different possible complete assignments to all the conditional probability table (CPT) entries. We will assume that all possible assignments are equally likely a priori, which means that we are looking for a maximum likelihood hypothesis. That is, we wish to find the set of CPT entries that maximizes the probability of the data, $P(D|H_i)$.

The method we will use to do this is quite similar to the gradient descent method for neural networks. We will write the probability of the data as a function of the CPT entries, and then calculate a gradient. As with neural networks, we will find that the gradient can be calculated locally by each node using information that is available in the normal course of belief network calculations. Thus, the CPT entries are analogous to the weights, and $P(D|H_i)$ is (inversely) analogous to the error E . Belief network systems equipped with this kind of learning scheme are called **adaptive probabilistic networks** (APNs).

Suppose we have a training set $D = \{D_1, \dots, D_m\}$, where each case D_j consists of an assignment of values to some subset of the variables in the network. We assume that each case is drawn independently from some underlying distribution that we are trying to model. The problem is to adjust the conditional probabilities in the network in order to maximize the likelihood of the data. We will write this likelihood as $P(\mathbf{D})$. The reader should bear in mind that here $P(\cdot)$ is really $P_{\mathbf{w}}(\cdot)$, that is, the probability according to a joint distribution specified by \mathbf{w} , the set of all of the conditional probability values in the network. In order to construct a hill-climbing algorithm to maximize the likelihood, we need to calculate the derivative of the likelihood with respect to each of the conditional probability values w_i in \mathbf{w} .

It turns out to be easiest to compute the derivative of the logarithm of the likelihood. Because the log-likelihood is monotonically related to the likelihood itself, a maximum on the

log-likelihood surface is also a maximum on the likelihood surface. We calculate the gradient using partial derivatives, varying a single value w_i while keeping the others constant:⁶

$$\frac{\partial \ln P(\mathbf{D})}{\partial w_i} = \frac{\partial \ln \prod_j P(D_j)}{\partial w_i} \underset{?}{=} \sum_j \frac{\partial \ln P(D_j)}{\partial w_i} \underset{?}{=} \sum_j \frac{\partial P(D_j)/\partial w_i}{P(D_j)}$$

Hence, we can calculate separately the gradient contribution of each case and sum the results.

Now the aim is to find an expression for the gradient contribution from a single case, such that the contribution can be calculated using only information local to the node with which w_i is associated. Let w_i be a specific entry in the conditional probability table for a node X given its parent variables \mathbf{U} . We'll assume that it is the entry for the case $X = x_i$ given $\mathbf{U} = \mathbf{u}_i$:

$$w_i = P(X = x_i \mid \mathbf{U} = \mathbf{u}_i) = P(x_i \mid \mathbf{u}_i)$$

In order to get an expression in terms of local information, we introduce X and \mathbf{U} by averaging over their possible values:

$$\begin{aligned} \frac{\partial P(D_j)/\partial w_i}{P(D_j)} &= \frac{\frac{\partial}{\partial w_i} (\sum_{x,\mathbf{u}} P(D_j \mid x, \mathbf{u})P(x, \mathbf{u}))}{P(D_j)} \\ &= \frac{\frac{\partial}{\partial w_i} (\sum_{x,\mathbf{u}} P(D_j \mid x, \mathbf{u})P(x \mid \mathbf{u})P(\mathbf{u}))}{P(D_j)} \end{aligned}$$

For our purposes, the important property of this expression is that w_i appears only in linear form. In fact, w_i appears only in one term in the summation, namely the term for x_i and \mathbf{u}_i . For this term, $P(x \setminus \mathbf{u})$ is just w_i , hence

$$\frac{\partial P(D_j)/\partial w_i}{P(D_j)} = \frac{P(D_j \mid x_i, \mathbf{u}_i)P(\mathbf{u}_i)}{P(D_j)}$$

Further manipulation reveals that the gradient calculation can "piggyback" on the calculations of posterior probabilities done in the normal course of belief network operation—that is, calculations of the probabilities of network variables given the observed data. To do this, we apply Bayes' theorem to the above equation, yielding

$$\frac{\partial P(D_j)/\partial w_i}{P(D_j)} \underset{P(D_j) \sim}{=} \frac{P(x_i, \mathbf{u}_i \mid D_j)P(D_j)P(\mathbf{u}_i)}{P(\mathbf{X}, \mathbf{u}_i)P(D_j)} = \frac{P(x_i, \mathbf{u}_i \mid D_j)}{P(x_i \mid \mathbf{u}_i)} \underset{w_i}{\sim} \frac{P(x_i, \mathbf{u}_i \mid D_j)}{w_i}$$

In most implementations of belief network inference, the term $P(x_i, \mathbf{u}_i \mid D_j)$ is either computed directly or is easily obtained by summing a small number of table entries. The complete gradient vector is obtained by summing the above expression over the data cases to give the gradient component with respect to each w_i for the likelihood of the entire training set. Thus, the necessary information for calculating the gradient can be derived directly from the normal computations done by belief networks as new evidence is obtained.

Once we have a locally computable expression for the gradient, we can apply the same kinds of hill-climbing or simulated annealing methods as are used for neural networks. Learning with belief networks has the advantage that a human expert can easily provide a structure for the

⁶ We also need to include the constraint that the conditional probability values for any given conditioning case must remain normalized. A formal analysis shows that the derivative of the constrained system (where the columns sum to one) is equal to the orthogonally projection of the unconstrained derivative onto the constraint surface.

network that reflects the causal structure of the domain. This prior knowledge should help the network to learn much faster from a given set of examples. Moreover, the results of learning are more easily understood, and, because probabilities are produced, the results can be used in making rational decisions.

One can also use the gradient-descent method in association with an algorithm designed to generate the structure of the network. Because such algorithms usually work by evaluating candidate structures for their ability to model the data, one can simply use the gradient descent to find the best fit between any candidate structure and the data.

A comparison of belief networks and neural networks

Given the close similarity between belief networks (particularly the adaptive variety) and neural networks, a detailed comparison is in order. The two formalisms can be compared as representation systems, inference systems, and learning systems.

Both neural networks and belief networks are attribute-based representations. Both handle discrete and continuous inputs, although algorithms for handling continuous variables in belief networks are less developed. The principal difference is that belief networks are localized representations, whereas neural networks are distributed representations. Nodes in belief networks represent propositions with well-defined semantics and well-defined probabilistic relationships to other propositions. Units in neural networks, on the other hand, typically do not represent specific propositions. Even if they did, the calculations carried by the network do not treat propositions in any semantically meaningful way. In practical terms, this means that humans can neither construct nor understand neural network representations. The well-defined semantics of belief networks also means that they can be constructed automatically by programs that manipulate first-order representations.

Another representational difference is that belief network variables have *two* dimensions of “activation”—the range of values for the proposition, and the probability assigned to each of those values. The outputs of a neural network can be viewed as *either* the probability of a Boolean variable, or an exact value for a continuous variable, but neural networks cannot handle both probabilities and multivalued or continuous variables simultaneously.

As inference mechanisms—once they have been trained—feedforward neural networks can execute in linear time, whereas general belief network inference is NP-hard. On closer inspection, this is not as clear an advantage as it might seem, because a neural network would in some cases have to be exponentially larger in order to represent the same input/output mapping as a belief network (else we would be able to solve hard problems in polynomial time). Practically speaking, any neural network that can be trained is small enough so that inference is fast, whereas it is not hard to construct belief networks that take a long time to run. One other important aspect of belief networks is their flexibility, in the sense that at any time any subset of the variables can be treated as inputs, and any other subset as outputs, whereas feedforward neural networks have fixed inputs and outputs.

With respect to learning, a comparison is difficult because adaptive probabilistic networks (APNs) are a very recent development. One can expect the time per iteration of an APN to be slower, because it involves an inference process. On the other hand, a human (or another part of

the agent) can provide prior knowledge to the APN learning process in the form of the network structure and/or conditional probability values. Since this reduces the hypothesis space, it should allow the APN to learn from fewer examples. Also, the ability of belief networks to represent propositions locally may mean that they converge faster to a correct representation of a domain that has local structure—that is, in which each proposition is directly affected by only a small number of other propositions.

19.7 SUMMARY

Learning in complex network representations is currently one of the hottest topics in science. It promises to have broad applications in computer science, neurobiology, psychology, and physics. This chapter has presented some of the basic ideas and techniques, and given a flavor of the mathematical underpinnings. The basic points are as follows:

- A **neural network** is a computational model that shares some of the properties of brains: it consists of many simple units working in parallel with no central control. The connections between units have numeric weights that can be modified by the learning element.
- The behavior of a neural network is determined by the connection topology and the nature of the individual units. **Feed-forward** networks, in which the connections form an acyclic graph, are the simplest to analyze. Feed-forward networks implement state-free functions.
- A **perceptron** is a feed-forward network with a single layer of units, and can only represent **linearly separable** functions. If the data are linearly separable, the **perceptron learning rule** can be used to modify the network's weights to fit the data exactly.
- **Multilayer feed-forward** networks can represent any function, given enough units.
- The **back-propagation** learning algorithm works on multilayer feed-forward networks, using gradient descent in weight space to minimize the output error. It converges to a locally optimal solution, and has been used with some success in a variety of applications. As with all hill-climbing techniques, however, there is no guarantee that it will find a global solution. Furthermore, its convergence is often very slow.
- **Bayesian learning** methods can be used to learn representations of probabilistic functions, particularly belief networks. Bayesian learning methods must trade off the prior belief in a hypothesis against its degree of agreement with the observed data.
- There are a variety of learning problems associated with belief networks, depending on whether the structure is fixed or unknown, and whether variables are hidden or observable.
- With a fixed structure and hidden variables, belief network learning has a remarkable similarity to neural network learning. Gradient descent methods can be used, but belief networks also have the advantage of a well-understood semantics for individual nodes. This allows the provision of prior knowledge in order to speed up the learning process.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

McCulloch and Pitts (1943) introduced the fundamental idea of analyzing neural activity via thresholds and weighted sums. Early cybernetics and control theory (Wiener, 1948), based on the notion of negative feedback loops, played a role as a model for learning in neural networks. *The Organization of Behavior* (Hebb, 1949) was influential in promoting the hypothesis that human and animal long-term memory is mediated by permanent alterations in the synapses. *Design for a Brain* (Ashby, 1952) put forth the idea that intelligence could be created by the use of "homeostatic" devices which learn through a kind of exhaustive search.

Minsky and Papert (1988, pp. ix-x) mention a machine built by Marvin Minsky in 1951 that may well be the first actual neural network learning system ever built. Minsky's (1954) doctoral dissertation continued the exploration of neural networks. The aptly-named "Pandemonium" system (Selfridge, 1959; Selfridge and Neisser, 1960) involved a relatively fine-grained distributed control regime reminiscent of neural networks. Cragg and Temperley (1954; 1955) drew parallels between McCulloch–Pitts neural networks and "spin systems" in physics. Caianello (1961) designed a statistical theory of learning in neural networks, drawing on classical statistical mechanics. Von Neumann (1958) provides a comparison between the functioning of the brain and the operation of digital computers. Frank Rosenblatt (1957) invented the modern "perceptron" style of neural network, composed of trainable threshold units.

Similar devices called "adalines" (for "Adaptive Linear") were invented about the same time (Widrow and Hoff, 1960; Widrow, 1962). Hawkins (1961) gives a detailed history of early work in "self-organizing systems" or "neural cybernetics," as these approaches were then called.

Frank Rosenblatt (1960) found the first proof of the perceptron convergence theorem, although it had been foreshadowed by purely mathematical work outside the context of neural networks (Agmon, 1954; Motzkin and Schoenberg, 1954). Two good books on this period of research are *Neurodynamics* (Rosenblatt, 1962) and *Learning Machines* (Nilsson, 1965). Nilsson's book is especially comprehensive and detailed. It has recently been republished as *The Mathematical Foundations of Learning Machines* (Nilsson, 1990) with a new introduction by Terrence Sejnowski and Halbert White.

Most work in neural networks before 1970 focused on the one-layer perceptron type of machine, but there were some exceptions. Widrow designed multilayer machines called "madalines" (Widrow, 1962). Other early multilayer machines are described in (Palmieri and Sanna, 1960; Gamba *et al.*, 1961).

The publication of *Perceptrons* (Minsky and Papert, 1969) marked the end of an era. The authors were severely critical of the unguided experimentation and lack of mathematical rigor that characterized much of the early work on perceptrons. They established the linear separability criterion for tasks that could be accomplished by one-layer perceptrons, thus explaining the failure of the early efforts at solving problems that violated this criterion. Minsky and Papert also gave some results on early multilayer systems. In the Epilogue to the expanded edition of *Perceptrons* (Minsky and Papert, 1988), they forcefully rebut the charge that the publication of the first edition was responsible for the long perceptron winter of the 1970s, arguing that perceptron research had already lost its momentum and that the first edition merely explained this phenomenon. They reaffirm the long-term promise of mathematically sound neural network research, while at the

same time criticizing contemporary connectionism circa 1988 for the same lack of rigor that had plagued the early perceptron work.

The papers in (Hinton and Anderson, 1981), based on a conference in San Diego in 1979, can be regarded as marking the renaissance of connectionism. The two-volume "PDP" (Parallel Distributed Processing) anthology (Rumelhart *et al.*, 1986) really put neural networks on the map for many AI researchers, as well as popularizing the back-propagation algorithm. Several advances made this possible. Hopfield (1982) analyzed symmetric networks using statistical mechanics and analogies from physics. The Boltzmann Machine (Hinton and Sejnowski, 1983; Hinton and Sejnowski, 1986) and the analysis of neural networks using the physical theory of magnetic spin glasses (Amit *et al.*, 1985) tightened the links between statistical mechanics and neural network theory—providing not only useful mathematical insights but also *respectability*. The back-propagation technique had been invented quite early (Bryson and Ho, 1969) but it was rediscovered several times (Werbos, 1974; Parker, 1985). Minsky and Papert (1988) criticize the generalized delta rule as a straightforward variant of simple hill-climbing, just as the perceptron learning algorithm had been.

The expressiveness of multilayer networks was investigated by Cybenko (1988; 1989), who showed that two hidden layers are enough to represent any function and a single layer is enough to represent any *continuous* function. These results, although reassuring, are not very exciting when one realizes that they are achieved by allocating a separate collection of units to represent the output value for each small region of the (exponentially large) input space.

The problem of finding a good structure for a multilayer network was addressed using genetic algorithms by Harp *et al.* (1990) and by Miller *et al.* (1989). The "optimal brain damage" method for removing useless connections is by LeCun *et al.* (1989), and Sietsma and Dow (1988) show how to remove useless units. The tiling algorithm for growing larger structures is by Mézard and Nadal (1989). Similar algorithms that grow slightly different topologies were proposed by Marchand *et al.* (1990) and by Frean (1990).

The complexity of neural network learning has been investigated by researchers in computational learning theory. Some of the earliest results were obtained by Judd (1990), who showed that the general problem of finding a set of weights consistent with a set of examples is NP-complete, even under very restrictive assumptions. Avrim Blum and Ron Rivest (1992) proved that training even a *three-node network* is NP-complete! These results suggest that weight space can contain an exponential number of local minima, for otherwise a random-restart hill-climbing algorithm would be able to find a global optimum in polynomial time.

One topic of great current interest in neural network research is the use of specialized parallel hardware, including analog computation. Systems may use analog VLSI (Alspector *et al.*, 1987; Mead, 1989), optoelectronics (Farhat *et al.*, 1985; Peterson *et al.*, 1990), or exotic, fully optical computing technologies such as spatial light modulation (Abu-Mostafa and Psaltis, 1987; Hsu *et al.*, 1988).

Neural networks constitute a large field of study with an abundance of resources available for the inquirer. Probably the best available textbook is *Introduction to the Theory of Neural Computation* (Hertz *et al.*, 1991), which emphasizes the connections with statistical mechanics (the authors are physicists). *Self-Organization and Associative Memory* (Kohonen, 1989) provides considerable mathematical background. For biological nervous systems, a very thorough introduction is (Kandel *et al.*, 1991). A good introduction to the detailed functioning of individual

neurons is (Miles, 1969). Articles by Cowan and Sharp (1988b; 1988a) present inclusive surveys of the history of neural network research. A very comprehensive bibliography is available in *NeuralSource* (Wasserman and Oetzel, 1990).

The most important conference in the field is the annual NIPS (Neural Information Processing Conference) conference, whose proceedings are published as the series *Advances in Neural Information Processing Systems*, starting with (Touretzky, 1989). Current research also appears in the International Joint Conference on Neural Networks (IJCNN). Major journals for the field include *Neural Computation*; *Neural Networks*; *IEEE Transactions on Neural Networks*; the *International Journal of Neural Systems*; and *Concepts in Neuroscience*.

The topic of learning belief networks has received attention only very recently. For the fixed-structure, fully observable case, Spiegelhalter, Dawid, Lauritzen, and Cowell (Spiegelhalter *et al.*, 1993) provide a thorough analysis of the statistical basis of belief network modification using Dirichlet priors. They also give a heuristic approximation for the hidden-variable case. Pearl (1988, Chapter 8) describes an algorithm for learning polytrees with unknown structure and fully observable variables. Heckerman, Geiger, and Chickering (1994) describe an elegant and effective heuristic algorithm for recovering the structure of general networks in the fully observable case, building on the work of Cooper and Herskovits (1992). For the case of hidden variables and unknown structure, see (Spirtes *et al.*, 1993).

The general problem of recovering distributions from data with missing values and hidden variables is addressed by the EM algorithm (Dempster *et al.*, 1977). The algorithm in the chapter (Russell *et al.*, 1994) can be seen as a variant of EM in which the "maximize" phase is carried out by a gradient-following method. Lauritzen (1991) also considers the application of EM to belief networks. A gradient-following algorithm for learning **sigmoid networks** (belief networks in which each CPT represents the same function as a standard neural-network unit) was proposed by Radford Neal (1991), who went on to show that Boltzmann Machines are a special case of belief networks. Neal was among the first to point out the extremely close connection between neural and belief networks.

EXERCISES

19.1 Construct by hand a neural network that computes the XOR function of two inputs. Make sure to specify what sort of units you are using.

19.2 We know that a simple perceptron cannot represent XOR (or, generally, the parity function of its inputs). Describe what happens to the weights of a four-input, step-function perceptron, beginning with all weights set to 0.1, as examples of the parity function arrive.

19.3 Suppose you had a neural network with linear activation functions. That is, for each unit the output is some constant c times the weighted sum of the inputs.

- a. Assume that the network has one hidden layer. For a given assignment to the weights W , write down equations for the value of the units in the output layer as a function of W and

the input layer I , without any explicit mention to the output of the hidden layer. Show that there is a network with no hidden units that computes the same function.

- b. Repeat the calculation in part (a), this time for a network with any number of hidden layers. What can you conclude about linear activation functions?



- 19.4** Consider the following set of examples. Each example has six inputs and one target output:

I_1	1	1	1	1	1	1	1	0	0	0	0	0	0
I_2	0	0	0	1	1	0	0	1	1	0	1	0	1
I_3	1	1	1	0	1	0	0	1	1	0	0	0	1
h	0	1	0	0	1	0	0	1	0	1	1	1	0
I_5	0	0	1	1	0	1	1	0	1	1	0	0	1
I_6	0	0	0	1	0	1	0	1	1	0	1	1	0
T	1	1	1	1	1	1	0	1	0	0	0	0	0

- a. Run the perceptron learning rule on this example set, and show the resulting set of weights.
 b. Run the decision tree learning rule, and show the resulting decision tree.
 c. Comment on your results.



- 19.5** Implement a data structure for layered, feed-forward neural networks, remembering to provide the information needed for both forward evaluation and backward propagation. Using this data structure, write a function NEURAL-NETWORK-OUTPUT that takes an example and a network and computes the appropriate output values.

- 19.6** Suppose that a training set contains only a single example, repeated 100 times. In 80 of the 100 cases, the single output value is 1; in the other 20, it is 0. What will a back-propagation network predict for this example, assuming that it has been trained and reaches a global optimum? (*Hint:* to find the global optimum, differentiate the error function and set to zero.)

- 19.7** The network in Figure 19.13 has four hidden nodes. This number was chosen somewhat arbitrarily. Run systematic experiments to measure the learning curves for networks with different numbers of hidden nodes. What is the optimal number? Would it be possible to use a cross-validation method to find the best network before the fact?

20

REINFORCEMENT LEARNING

In which we examine how an agent can learn from success and failure, reward and punishment.

20.1 INTRODUCTION

In the previous two chapters, we have studied learning methods that learn from examples. That is, the environment provides input/output pairs, and the task is to learn a function that could have generated those pairs. These supervised learning methods are appropriate when a teacher is providing correct values or when the function's output represents a prediction about the future that can be checked by looking at the percepts in the next time step. In this chapter, we will study how agents can learn in much less generous environments, where the agent receives no examples, and starts with no model of the environment and no utility function.

For example, we know an agent can learn to play chess by supervised learning—by being given examples of game situations along with the best move for that situation. But if there is no friendly teacher providing examples, what can the agent do? By trying random moves, the agent can eventually build a predictive model of its environment: what the board will be like after it makes a given move, and even how the opponent is likely to reply in a given situation. But without some feedback as to what is good and what is bad, the agent will have no grounds for deciding which move to make. Fortunately, the chess-playing agent does receive some feedback, even without a friendly teacher—at the end of the game, the agent perceives whether it has won or lost. This kind of feedback is called a **reward, or reinforcement**. In games like chess, the reinforcement is received only at the end of the game. We call this a **terminal state** in the state history sequence. In other environments, the rewards come more frequently—in ping-pong, each point scored can be considered a reward. Sometimes rewards are given by a teacher who says "nice move" or "uh-oh" (but does not say what the best move is).

The task of **reinforcement learning** is to use rewards to learn a successful agent function. This is difficult because the agent is never told what the right actions are, nor which rewards are

REWARD
TERMINAL STATE

due to which actions. A game-playing agent may play flawlessly except for one blunder, and at the end of the game get a single reinforcement that says "you lose." The agent must somehow determine which move was the blunder.

Within our framework of agents as functions from percepts to actions, a reward can be provided by a percept, but the agent must be "hardwired" to recognize that percept as a reward rather than as just another sensory input. Thus, animals seem to be hardwired to recognize pain and hunger as negative rewards, and pleasure and food as positive rewards. Training a dog is made easier by the fact that humans and dogs happen to agree that a low-pitched sound (either a growl or a "bad dog!") is a negative reinforcement. Reinforcement has been carefully studied by animal psychologists for over 60 years.

In many complex domains, reinforcement learning is the only feasible way to train a program to perform at high levels. For example, in game playing, it is very hard for a human to provide accurate and consistent evaluations of large numbers of positions, which would be needed to train an evaluation function directly from examples. Instead, the program can be told when it has won or lost, and can use this information to learn an evaluation function that gives reasonably accurate estimates of the probability of winning from any given position. Similarly, it is extremely difficult to program a robot to juggle; yet given appropriate rewards every time a ball is dropped or caught, the robot can learn to juggle by itself.

In a way, reinforcement learning is a restatement of the entire AI problem. An agent in an environment gets percepts, maps some of them to positive or negative utilities, and then has to decide what action to take. To avoid reconsidering all of AI and to get at the principles of reinforcement learning, we need to consider how the learning task can vary:

- The environment can be accessible or inaccessible. In an accessible environment, states can be identified with percepts, whereas in an inaccessible environment, the agent must maintain some internal state to try to keep track of the environment.
- The agent can begin with knowledge of the environment and the effects of its actions; or it will have to learn this model as well as utility information.
- Rewards can be received only in terminal states, or in any state.
- Rewards can be components of the actual utility (points for a ping-pong agent or dollars for a betting agent) that the agent is trying to maximize, or they can be hints as to the actual utility ("nice move" or "bad dog").
- The agent can be a **passive learner** or an **active learner**. A passive learner simply watches the world going by, and tries to learn the utility of being in various states; an active learner must also act using the learned information, and can use its problem generator to suggest explorations of unknown portions of the environment.

PASSIVE LEARNER
ACTIVE LEARNER

Furthermore, as we saw in Chapter 2, there are several different basic designs for agents. Because the agent will be receiving rewards that relate to utilities, there are two basic designs to consider:

- The agent learns a utility function on states (or state histories) and uses it to select actions that maximize the expected utility of their outcomes.
- The agent learns an **action-value** function giving the expected utility of taking a given action in a given state. This is called **Q-learning**.

ACTION-VALUE
Q-LEARNING

An agent that learns utility functions must also have a model of the environment in order to make decisions, because it must know the states to which its actions will lead. For example, in order to make use of a backgammon evaluation function, a backgammon program must know what its legal moves are *and how they affect the board position*. Only in this way can it apply the utility function to the outcome states. An agent that learns an action-value function, on the other hand, need not have such a model. As long as it knows its legal moves, it can compare their values directly without having to consider their outcomes. Action-value learners therefore can be slightly simpler in design than utility learners. On the other hand, because they do not know where their actions lead, they cannot look ahead; this can seriously restrict their ability to learn, as we shall see.

We first address the problem of learning utility functions, which has been studied in AI since the earliest days of the field. (See the discussion of Samuel's checker player in Chapter 5.) We examine increasingly complex versions of the problem, while keeping initially to simple state-based representations. Section 20.6 discusses the learning of action-value functions, and Section 20.7 discusses how the learner can generalize across states. Throughout this chapter, we will assume that the environment is nondeterministic. At this point the reader may wish to review the basics of decision making in complex, nondeterministic environments, as covered in Chapter 17.

20.2 PASSIVE LEARNING IN A KNOWN ENVIRONMENT

TRAINING
SEQUENCE

To keep things simple, we start with the case of a passive learning agent using a state-based representation in a known, accessible environment. In passive learning, the environment generates state transitions and the agent perceives them.¹ Consider an agent trying to learn the utilities of the states shown in Figure 20.1(a). We assume, for now, that it is provided with a model M giving the probability of a transition from state i to state j , as in Figure 20.1(b). In each **training sequence**, the agent starts in state $(1,1)$ and experiences a sequence of state transitions until it reaches one of the terminal states $(3,2)$ or $(3,3)$, where it receives a reward.² A typical set of training sequences might look like this:

$$\begin{aligned} &(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,1) \rightarrow (1,2) \rightarrow (2,2) \rightarrow (3,2) \underline{-1} \\ &(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (2,2) \rightarrow (2,3) \rightarrow (3,3) \pm 1 \\ &(1,1) \rightarrow (1,2) \rightarrow (1,1) \rightarrow (1,2) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (2,2) \rightarrow (2,3) \rightarrow (3,3) \pm 1 \\ &(1,1) \rightarrow (1,2) \rightarrow (2,2) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \underline{+1} \\ &(1,1) \rightarrow (2,1) \rightarrow (2,2) \rightarrow (2,1) \rightarrow (1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (2,2) \rightarrow (3,2) \underline{-1} \\ &(1,1) \rightarrow (2,1) \rightarrow (1,1) \rightarrow (1,2) \rightarrow (2,2) \rightarrow (3,2) \underline{-1} \end{aligned}$$

The object is to use the information about rewards to learn the expected utility $U(i)$ associated with each nonterminal state i . We will make one big simplifying assumption: the utility of a sequence is the sum of the rewards accumulated in the states of the sequence. That is, the utility

¹ Another way to think of a passive learner is as an agent with a fixed policy trying to determine its benefits.

² The period from initial state to terminal state is often called an **epoch**.

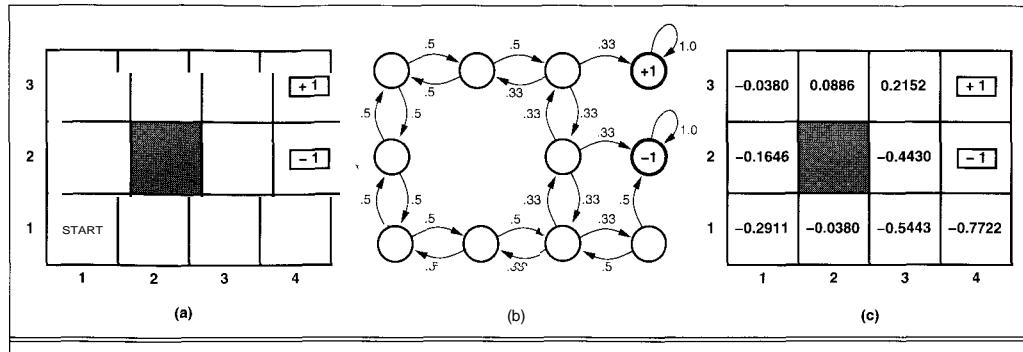


Figure 20.1 (a) A simple stochastic environment. State (1,1) is the start state. (b) Each state transitions to a neighboring state with equal probability among all neighboring states. State (4,2) is terminal with reward -1 , and state (4,3) is terminal with reward $+1$. (c) The exact utility values.

REWARD-TO-GO



function is **additive** in the sense defined on page 502. We define the **reward-to-go** of a state as the sum of the rewards from that state until a terminal state is reached. Given this definition, it is easy to see that *the expected utility of a state is the expected reward-to-go of that state*.

The generic agent design for passive reinforcement learning of utilities is shown in Figure 20.2. The agent keeps an estimate U of the utilities of the states, a table N of counts of how many times each state was seen, and a table M of transition probabilities from state to state. We assume that each percept e is enough to determine the STATE (i.e., the state is accessible), the agent can determine the REWARD component of a percept, and the agent can tell if a percept indicates a TERMINAL? state. In general, an agent can update its current estimated utilities after each observed transition. The key to reinforcement learning lies in the algorithm for updating the utility values given the training sequences. The following subsections discuss three possible approaches to UPDATE.

Naive updating

ADAPTIVE CONTROL THEORY

A simple method for updating utility estimates was invented in the late 1950s in the area of **adaptive control theory** by Widrow and Hoff (1960). We will call it the LMS (least mean squares) approach. In essence, it assumes that for each state in a training sequence, the *observed* reward-to-go on that sequence provides direct evidence of the actual expected reward-to-go. Thus, at the end of each sequence, the algorithm calculates the observed reward-to-go for each state and updates the estimated utility for that state accordingly. It can easily be shown (Exercise 20.1) that the LMS approach generates utility estimates that minimize the mean square error with respect to the observed data. When the utility function is represented by a table of values for each state, the update is simply done by maintaining a running average, as shown in Figure 20.3.

If we think of the utility function as a function, rather than just a table, then it is clear that the LMS approach is simply learning the utility function directly from examples. Each example has the state as input and the observed reward-to-go as output. This means that we have reduced reinforcement learning to a standard inductive learning problem, as discussed in Chapter 18. As

```

function PASSIVE-RL-AGENT( $e$ ) returns an action
  static:  $U$ , a table of utility estimates
   $N$ , a table of frequencies for states
   $M$ , a table of transition probabilities from state to state
   $percepts$ , a percept sequence (initially empty)

  add  $e$  to  $percepts$ 
  increment  $N[\text{STATE}[e]]$ 
   $U \leftarrow \text{UPDATE}(U, e, percepts, M, N)$ 
  if TERMINAL? $[e]$  then  $percepts \leftarrow$  the empty sequence
  return the action Observe

```

Figure 20.2 Skeleton for a passive reinforcement learning agent that just observes the world and tries to learn the utilities, U , of each state. The agent also keeps track of transition frequencies and probabilities. The rest of this section is largely devoted to defining the UPDATE function.

```

function LMS-UPDATE( $U, e, percepts, M, N$ ) returns an updated  $U$ 
  if TERMINAL? $[e]$  then reward-to-go  $\leftarrow 0$ 
  for each  $e_i$  in  $percepts$  (starting at end) do
     $reward-to-go \leftarrow reward-to-go + REWARD[e_i]$ 
     $U[\text{STATE}[e_i]] \leftarrow \text{RUNNING-AVERAGE}(U[\text{STATE}[e_i]], reward-to-go, N[\text{STATE}[e_i]])$ 
  end

```

Figure 20.3 The update function for least mean square (LMS) updating of utilities.

we will show, it is an easy matter to use more powerful kinds of representations for the utility function, such as neural networks. Learning techniques for those representations can be applied directly to the observed data.

One might think that the LMS approach more or less solves the reinforcement learning problem—or at least, reduces it to one we already know a lot about. In fact, the LMS approach misses a very important aspect of the reinforcement learning problem, namely, the fact that the utilities of states are not independent! The structure of the transitions among the states in fact imposes very strong additional constraints: *The actual utility of a state is constrained to be the probability-weighted average of its successors' utilities, plus its own reward.* By ignoring these constraints, LMS-UPDATE usually ends up converging very slowly on the correct utility values for the problem. Figure 20.4 shows a typical run on the 4 x 3 environment in Figure 20.1, illustrating both the convergence of the utility estimates and the gradual reduction in the root-mean-square error with respect to the *correct* utility values. It takes the agent well over a thousand training sequences to get close to the correct values.



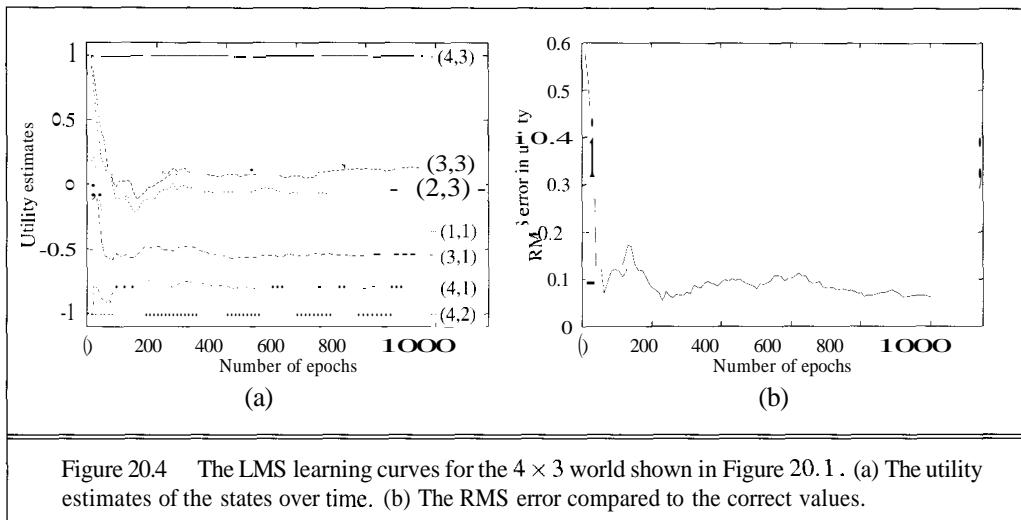


Figure 20.4 The LMS learning curves for the 4×3 world shown in Figure 20.1 . (a) The utility estimates of the states over time. (b) The RMS error compared to the correct values.

Adaptive dynamic programming

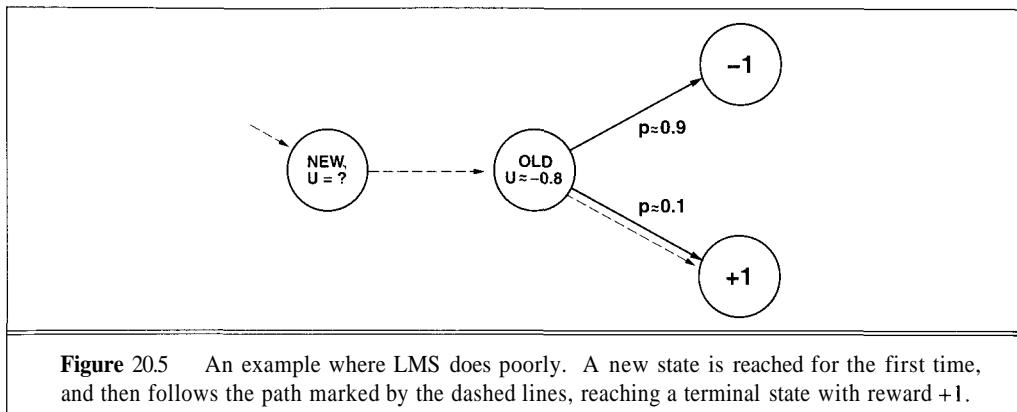
Programs that use knowledge of the structure of the environment usually learn much faster. In the example in Figure 20.5 (from (Sutton, 1988)), the agent already has a fair amount of experience with the three states on the right, and has learned the values indicated. However, the path followed from the new state reaches an unusually good terminal state. The LMS algorithm will assign a utility estimate of +1 to the new state, whereas it is clear that the new state is much less promising, because it has a transition to a state known to have utility ≈ -0.8 , and no other known transitions.

Fortunately, this drawback can be fixed. Consider again the point concerning the constraints on neighboring utilities. Because, for now, we are assuming that the transition probabilities are listed in the known table M_{ij} , the reinforcement learning problem becomes a well-defined sequential decision problem (see Chapter 17) as soon as the agent has observed the rewards for all the states. In our 4×3 environment, this usually happens after a handful of training sequences, at which point the agent can compute the exact utility values for all states. The utilities are computed by solving the set of equations

$$U(i) = R(i) + \sum_j M_{ij} U(j) \quad (20.1)$$

where $R(i)$ is the reward associated with being in state i , and M_{ij} is the probability that a transition will occur from state i to state j . This set of equations simply formalizes the basic point made in the previous subsection. Notice that because the agent is passive, no maximization over actions is involved (unlike Equation (17.3)). The process of solving the equations is therefore identical to a single **value determination** phase in the policy iteration algorithm. The exact utilities for the states in the 4×3 world are shown in Figure 20.1(c). Notice how the "safest" squares are those along the top row, away from the negative reward state.

We will use the term **adaptive dynamic programming (or ADP)** to denote any reinforcement learning method that works by solving the utility equations with a dynamic programming algorithm. In terms of its ability to make good use of experience, ADP provides a standard



against which to measure other reinforcement learning algorithms. It is, however, somewhat intractable for large state spaces. In backgammon, for example, it would involve solving roughly 10^{50} equations in 10^{50} unknowns.

Temporal difference learning

It is possible to have (almost) the best of both worlds—that is, one can approximate the constraint equations shown earlier without solving them for all possible states. *The key is to use the observed transitions to adjust the values of the observed states so that they agree with the constraint equations.* Suppose that we observe a transition from state i to state j , where currently $U(i) = -0.5$ and $U(j) = +0.5$. This suggests that we should consider increasing $U(i)$ to make it agree better with its successor. This can be achieved using the following updating rule:

$$U(i) \leftarrow U(i) + \alpha (R(i) + U(j) - U(i)) \quad (20.2)$$

where α is the **learning rate** parameter. Because this update rule uses the difference in utilities between successive states, it is often called the **temporal-difference**, or **TD**, equation.

TEMPORAL-DIFFERENCE

The basic idea of all temporal-difference methods is to first define the conditions that hold locally when the utility estimates are correct; and then to write an update equation that moves the estimates toward this ideal "equilibrium" equation. In the case of passive learning, the equilibrium is given by Equation (20.1). Now Equation (20.2) does in fact cause the agent to reach the equilibrium given by Equation (20.1), but there is some subtlety involved. First, notice that the update only involves the actual successor, whereas the actual equilibrium conditions involve all possible next states. One might think that this causes an improperly large change in $U(i)$ when a very rare transition occurs; but, in fact, because rare transitions occur only rarely, the *average value* of $U(i)$ will converge to the correct value. Furthermore, if we change α from a fixed parameter to a function that decreases as the number of times a state has been visited increases, then $U(i)$ itself will converge to the correct value (Dayan, 1992). This gives us the algorithm TD-UPDATE, shown in Figure 20.6. Figure 20.7 shows atypical run of the TD learning algorithm on the world in Figure 20.1. Although TD generates noisier values, the RMS error is actually significantly less than that for LMS after 1000 iterations.

```

function TD-UPDATE( $U, e, \text{percepts}, M, N$ ) returns the utility table  $U$ 
  if TERMINAL? $[e]$  then
     $U[\text{STATE}[e]] \leftarrow \text{RUNNING-AVERAGE}(U[\text{STATE}[e]], \text{REWARD}[e], N[\text{STATE}[e]])$ 
  else if  $\text{percepts}$  contains more than one element then
     $e' \leftarrow$  the penultimate element of  $\text{percepts}$ 
     $i, j \leftarrow \text{STATE}[e'], \text{STATE}[e]$ 
     $U[i] \leftarrow U[i] + \alpha(N[i])(\text{REWARD}[e'] + U[j] - U[i])$ 
  
```

Figure 20.6 An algorithm for updating utility estimates using temporal differences.

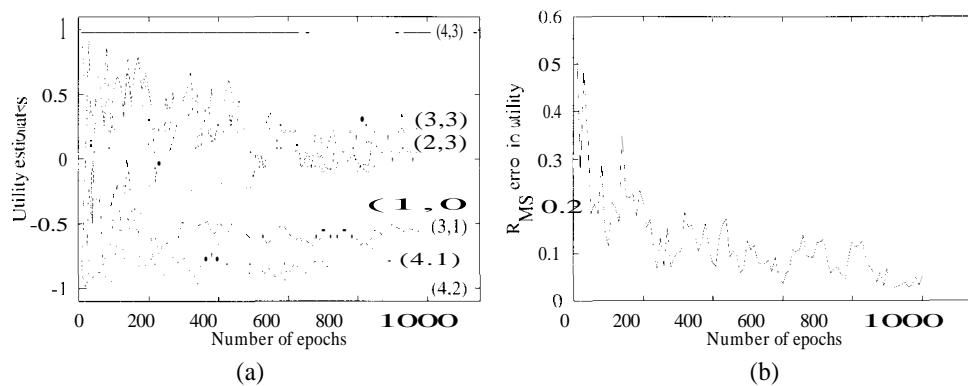


Figure 20.7 The TD learning curves for the 4×3 world. (a) The utility estimates of the states over time. (b) The RMS error compared to the correct values.

20.3 PASSIVE LEARNING IN AN UNKNOWN ENVIRONMENT

The previous section dealt with the case in which the environment model M is already known. Notice that of the three approaches, only the dynamic programming method used the model in full. TD uses information about connectedness of states, but only from the current training sequence. (As we mentioned before, all utility-learning methods will use the model during subsequent action selection.) Hence LMS and TD will operate unchanged in an initially unknown environment.

The adaptive dynamic programming approach simply adds a step to **PASSIVE-RL-AGENT** that updates an estimated model of the environment. Then the estimated model is used as the basis for a dynamic programming phase to calculate the corresponding utility estimates after each observation. As the environment model approaches the correct model, the utility estimates will,

of course, converge to the correct utilities. Because the environment model usually changes only slightly with each observation, the dynamic programming phase can use value iteration with the previous utility estimates as initial values and usually converges quite quickly.

The environment model is learned by direct observation of transitions. In an accessible environment, each percept identifies the state, and hence each transition provides a direct input/output example for the transition function represented by M . The transition function is usually stochastic—that is, it specifies a probability for each possible successor rather than a single state. A reinforcement learning agent can use any of the techniques for learning stochastic functions from examples discussed in Chapters 18 and 19. We discuss their application further in Section 20.7.

Continuing with our tabular representation for the environment, we can update the environment model M simply by keeping track of the percentage of times each state transitions to each of its neighbors. Using this simple technique in the 4×3 world from Figure 20.1, we obtain the learning performance shown in Figure 20.8. Notice that the ADP method converges far faster than either LMS or TD learning.

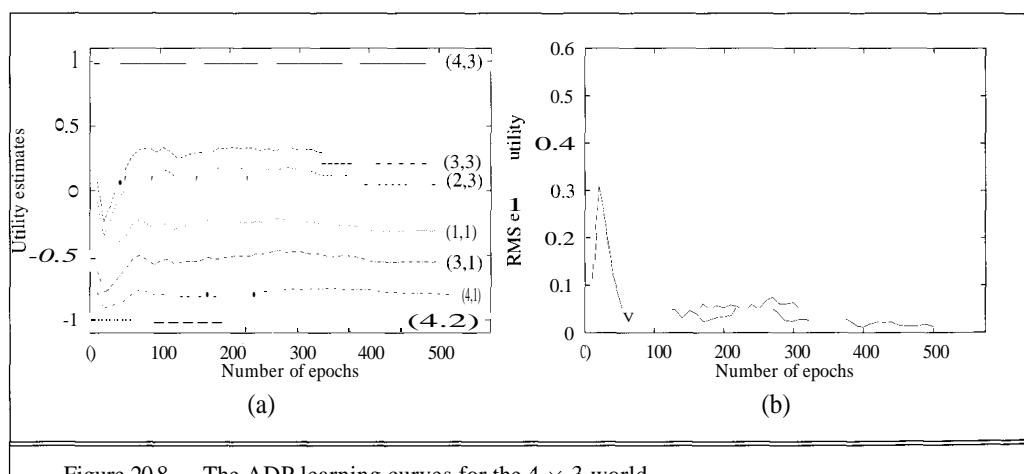


Figure 20.8 The ADP learning curves for the 4×3 world.

The ADP approach and the TD approach are actually closely related. Both try to make local adjustments to the utility estimates in order to make each state "agree" with its successors. One minor difference is that TD adjusts a state to agree with its *observed* successor (Equation (20.2)), whereas ADP adjusts the state to agree with *all* of the successors that might occur given an optimal action choice, weighted by their probabilities (Equation (20.1)). This difference disappears when the effects of TD adjustments are averaged over a large number of transitions, because the frequency of each successor in the set of transitions is approximately proportional to its probability. A more important difference is that whereas TD makes a single adjustment per observed transition, ADP makes as many as it needs to restore consistency between the utility estimates U and the environment model M . Although the observed transition only makes a local change in M , its effects may need to be propagated throughout U . Thus, TD can be viewed as a crude but efficient first approximation to ADP.

Each adjustment made by ADP could be viewed, from the TD point of view, as a result of a "pseudo-experience" generated by simulating the current environment model. It is possible to extend the TD approach to use an environment model to generate several pseudo-experiences—transitions that the TD agent can imagine *might* happen given its current model. For each observed transition, the TD agent can generate a large number of imaginary transitions. In this way, the resulting utility estimates will approximate more and more closely those of ADP—of course, at the expense of increased computation time.

In a similar vein, we can generate more efficient versions of ADP by directly approximating the algorithms for value iteration or policy iteration. Recall that full value iteration can be intractable when the number of states is large. Many of the adjustment steps, however, are extremely tiny. One possible approach to generating reasonably good answers quickly is to bound the number of adjustments made after each observed transition. One can also use a heuristic to rank the possible adjustments so as to carry out only the most significant ones. The **prioritized-sweeping** heuristic prefers to make adjustments to states whose *likely* successors have just undergone a *large* adjustment in their own utility estimates. Using heuristics like this, approximate ADP algorithms usually can learn roughly as fast as full ADP, in terms of the number of training sequences, but can be several orders of magnitude more efficient in terms of computation (see Exercise 20.3). This enables them to handle state spaces that are far too large for full ADP. Approximate ADP algorithms have an additional advantage: in the early stages of learning a new environment, the environment model M often will be far from correct, so there is little point in calculating an exact utility function to match it. An approximation algorithm can use a minimum adjustment size that decreases as the environment model becomes more accurate. This eliminates the very long value iterations that can occur early in learning due to large changes in the model.

PRIORITY
SWEEPING

20.4 ACTIVE LEARNING IN AN UNKNOWN ENVIRONMENT

A passive learning agent can be viewed as having a fixed policy, and need not worry about which actions to take. An active agent must consider what actions to take, what their outcomes may be, and how they will affect the rewards received.

The PASSIVE-RL-AGENT model of page 602 needs only minor changes to accommodate actions by the agent:

- The environment model must now incorporate the probabilities of transitions to other states *given a particular action*. We will use M_{ij}^a to denote the probability of reaching state j if the action a is taken in state i .
- The constraints on the utility of each state must now take into account the fact that the agent has a choice of actions. A rational agent will maximize its expected utility, and instead of Equation (20.1) we use Equation (17.3), which we repeat here:

$$U(i) = R(i) + \max_a \sum_j M_{ij}^a U(j) \quad (20.3)$$

- The agent must now choose an action at each step, and will need a performance element to do so. In the algorithm, this means calling `PERFORMANCE-ELEMENT(e)` and returning the resulting action. We assume that the model M and the utilities U are shared by the performance element; that is the whole point of learning them.

We now reexamine the dynamic programming and temporal-difference approaches in the light of the first two changes. The question of how the agent should act is covered in Section 20.5.

Because the ADP approach uses the environment model, we will need to change the algorithm for learning the model. Instead of learning the probability M_{yj} of a transition, we will need to learn the probability M_{yj}^a of a transition conditioned on taking an action a . In the explicit tabular representation for M , this simply means accumulating statistics in a three-dimensional table. With an implicit functional representation, as we will soon see, the input to the function will include the action taken. We will assume that a procedure `UPDATE-ACTIVE-MODEL` takes care of this. Once the model has been updated, then the utility function can be recalculated using a dynamic programming algorithm and then the performance element chooses what to do next. We show the overall design for `ACTIVE-ADP-AGENT` in Figure 20.9.

An active temporal-difference learning agent that learns utility functions also will need to learn a model in order to use its utility function to make decisions. The model acquisition problem for the TD agent is identical to that for the ADP agent. What of the TD update rule itself? Perhaps surprisingly, the update rule (20.2) remains unchanged. This might seem odd, for the following reason. Suppose the agent takes a step that normally leads to a good destination, but because of nondeterminism in the environment the agent ends up in a catastrophic state. The TD update rule will take this as seriously as if the outcome had been the normal result of the

```

function ACTIVE-ADP-AGENT( $e$ ) returns an action
  static:  $U$ , a table of utility estimates
   $M$ , a table of transition probabilities from state to state for each action
   $R$ , a table of rewards for states
  percepts, a percept sequence (initially empty)
  last-action, the action just executed

  add  $e$  to percepts
   $R[\text{STATE}[e]] \leftarrow \text{REWARD}[e]$ 
   $M \leftarrow \text{UPDATE-ACTIVE-MODEL}(M, \text{percepts}, \text{last-action})$ 
   $U \leftarrow \text{VALUE-ITERATION}(U, M, R)$ 
  if TERMINAL? $[e]$  then
    percepts — the empty sequence
    last-action  $\leftarrow \text{PERFORMANCE-ELEMENT}(e)$ 
  return last-action

```

Figure 20.9 Design for an active ADP agent. The agent learns an environment model M by observing the results of its actions, and uses the model to calculate the utility function U using a dynamic programming algorithm (here `POLICY-ITERATION` could be substituted for `VALUE-ITERATION`).

action, whereas one might suppose that because the outcome was a fluke, the agent should not worry about it too much. In fact, of course, the unlikely outcome will only occur infrequently in a large set of training sequences; hence in the long run its effects will be weighted proportionally to its probability, as we would hope. Once again, it can be shown that the TD algorithm will converge to the same values as ADP as the number of training sequences tends to infinity.

20.5 EXPLORATION

The only remaining issue to address for active reinforcement learning is the question of what actions the agent should take—that is, what PERFORMANCE-ELEMENT should return. This turns out to be harder than one might imagine.

One might suppose that the correct way for the agent to behave is to choose whichever action has the highest expected utility given the current utility estimates—after all, that is all the agent has to go on. But this overlooks the contribution of action to learning. In essence, an action has two kinds of outcome:³

- It gains rewards on the current sequence.
- It affects the percepts received, and hence the ability of the agent to learn—and receive rewards in future sequences.

An agent therefore must make a trade-off between its immediate good—as reflected in its current utility estimates—and its long-term well-being. An agent that simply chooses to maximize its rewards on the current sequence can easily get stuck in a rut. At the other extreme, continually acting to improve one's knowledge is of no use if one never puts that knowledge into practice. In the real world, one constantly has to decide between continuing in a comfortable existence and striking out into the unknown in the hopes of discovering a new and better life.

In order to illustrate the dangers of the two extremes, we will need a suitable environment. We will use the stochastic version of the 4×3 world shown in Figure 17.1. In this world, the agent can attempt to move *North*, *South*, *East*, or *West*; each action achieves the intended effect with probability 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction. As before, we assume a reward of -0.04 (i.e., a cost of 0.04) for each action that doesn't reach a terminal state. The optimal policy and utility values for this world are shown in Figure 17.2, and the object of the learning agent is to converge towards these.

Let us consider two possible approaches that the learning agent might use for choosing what to do. The "wacky" approach acts randomly, in the hope that it will eventually explore the entire environment; and the "greedy" approach acts to maximize its utility using current estimates. As we see from Figure 20.10, the wacky approach succeeds in learning good utility estimates for all the states (top left). Unfortunately, its wacky policy means that it never actually gets better at reaching the positive reward (top right). The greedy agent, on the other hand, often finds a path to the +1 reward along the lower route via (2,1), (3,1), (3,2), and (3,3). Unfortunately, it then sticks to that path, never learning the utilities of the other states (bottom left). This means

³ Notice the direct analogy to the theory of information value in Chapter 16.

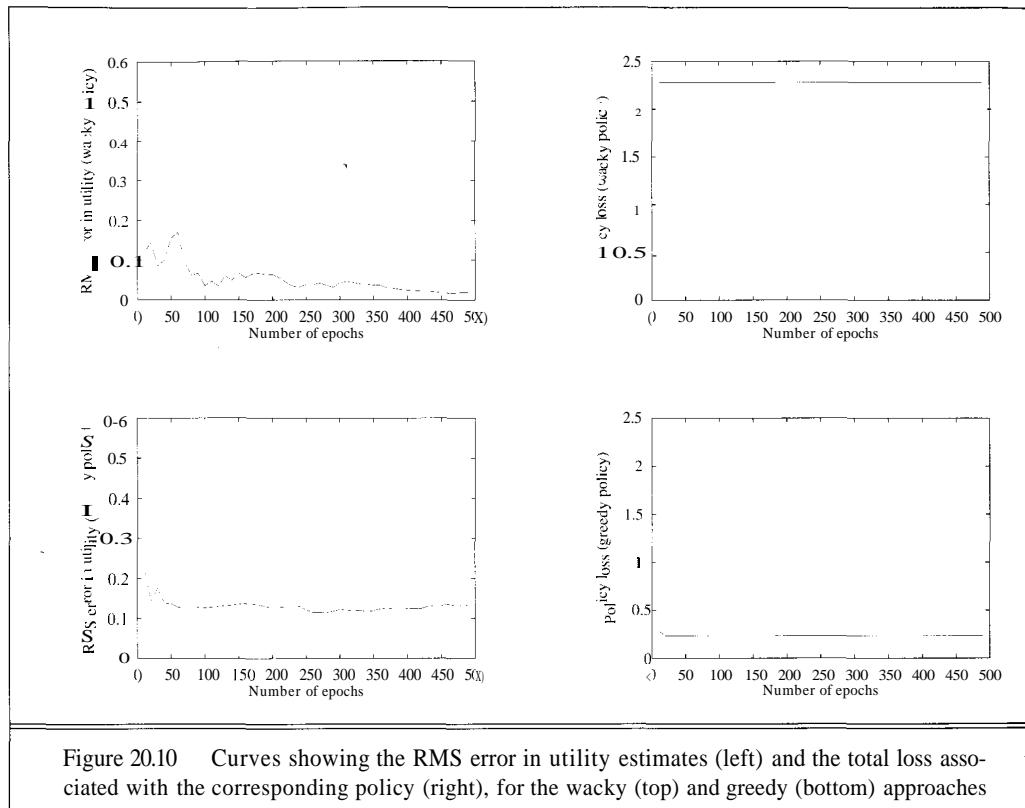


Figure 20.10 Curves showing the RMS error in utility estimates (left) and the total loss associated with the corresponding policy (right), for the wacky (top) and greedy (bottom) approaches to exploration.

that it too fails to achieve perfection (bottom right) because it does not find the optimal route via (1,2), (1,3), and (2,3).

Obviously, we need an approach somewhere between wackiness and greediness. The agent should be more wacky when it has little idea of the environment, and more greedy when it has a model that is close to being correct. Can we be a little more precise than this? Is there an *optimal* exploration policy? It turns out that this question has been studied in depth in the subfield of statistical decision theory that deals with so-called **bandit problems** (see sidebar).

BANDIT PROBLEMS

Although bandit problems are extremely difficult to solve exactly to obtain an *optimal* exploration policy, it is nonetheless possible to come up with a *reasonable* policy that seems to have the desired properties. In a given state, the agent should give some weight to actions that it has not tried very often, while being inclined to avoid actions that are believed to be of low utility. This can be implemented by altering the constraint equation (20.3) so that it assigns a higher utility estimate to relatively unexplored action-state pairs. Essentially, this amounts to an optimistic prior over the possible environments, and causes the agent to behave initially as if there were wonderful rewards scattered all over the place. Let us use $U^+(i)$ to denote the optimistic estimate of the utility (i.e., the expected reward-to-go) of the state i , and let $N(a, i)$ be the number of times action a has been tried in state i . Suppose we are using value iteration

EXPLORATION AND BANDITS

In Las Vegas, a *one-armed bandit* is a slot machine. A gambler can insert a coin, pull the lever, and collect the winnings (if any). An *n-armed bandit* has n levers. The gambler must choose which lever to play on each successive coin—the one that has paid off best, or maybe one that has not been tried?

The *n-armed bandit* problem is a formal model for real problems in many vitally important areas, such as deciding on the annual budget for AI research and development. Each arm corresponds to an action (such as allocating \$20 million for development of new AI textbooks) and the payoff from pulling the arm corresponds to the benefits obtained from taking the action (immense). Exploration, whether it is exploration of a new research field or exploration of a new shopping mall, is risky, expensive, and has uncertain payoffs; on the other hand, failure to explore at all means that one never discovers *any* actions that are worthwhile.

To formulate a bandit problem properly, one must define exactly what is meant by optimal behavior. Most definitions in the literature assume that the aim is to maximize the expected total reward obtained over the agent's lifetime. These definitions require that the expectation be taken over the possible worlds that the agent could be in, as well as over the possible results of each action sequence in any given world. Here, a "world" is defined by the transition model M_{ij}^a . Thus, in order to act optimally, the agent needs a prior distribution over the possible models. The resulting optimization problems are usually wildly intractable. In some cases, however, appropriate independence assumptions enable the problem to be solved in closed form. With a row of real slot machines, for example, the rewards in successive time steps and on different machines can be assumed to be independent. It turns out that the fraction of one's coins invested in a given machine should drop off proportionally to the probability that the machine is in fact the best, given the observed distributions of rewards.

The formal results that have been obtained for optimal exploration policies apply only to the case in which the agent represents the transition model as an explicit table and is not able to generalize across states and actions. For more realistic problems, it is possible to prove only convergence to a correct model and optimal behavior in the limit of infinite experience. This is easily obtained by acting randomly on some fraction of steps, where that fraction decreases appropriately over time.

One can use the theory of *n-armed bandits* to argue for the reasonableness of the selection strategy in genetic algorithms (see Section 20.8). If you consider each arm in an *n-armed bandit* problem to be a possible string of genes, and the investment of a coin in one arm to be the reproduction of those genes, then genetic algorithms allocate coins optimally, given an appropriate set of independence assumptions.

in an ADP learning agent; then we need to rewrite the update equation (i.e., Equation (17.4)) to incorporate the optimistic estimate. The following equation does this:

$$U^+(i) \leftarrow R(i) + \max_a f \left(\sum_{j,j} M_{ij}^a U^+ j), N(a,i) \right) \quad (20.4)$$

EXPLORATION
FUNCTION

where $f(u,n)$ is called the **exploration function**. It determines how greed (preference for high values of u) is traded off against curiosity (preference for low values of n , i.e., actions that have not been tried often). The function $f(u,n)$ should be increasing in u , and decreasing in n . Obviously, there are many possible functions that fit these conditions. One particularly simple definition is the following:

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

where R^+ is an optimistic estimate of the best possible reward obtainable in any state, and N_e is a fixed parameter. This will have the effect of making the agent try each action-state pair at least N_e times.

The fact that U^+ rather than U appears on the right-hand side of Equation (20.4) is very important. As exploration proceeds, the states and actions near the start state may well be tried a large number of times. If we used U , the nonoptimistic utility estimate, then the agent would soon become disinclined to explore further afield. The use of U^+ means that the benefits of exploration are propagated back from the edges of unexplored regions, so that actions that lead toward unexplored regions are weighted more highly, rather than just actions that are themselves unfamiliar. The effect of this exploration policy can be seen clearly in Figure 20.11, which shows a rapid convergence toward optimal performance, unlike that of the wacky or the greedy approaches. A very nearly optimal policy is found after just 18 trials. Notice that the utility estimates themselves do not converge as quickly. This is because the agent stops exploring the unrewarding parts of the state space fairly soon, visiting them only "by accident" thereafter. However, it makes perfect sense for the agent not to care about the exact utilities of states that it knows are undesirable and can be avoided.

20.6 LEARNING AN ACTION-VALUE FUNCTION

An action-value function assigns an expected utility to taking a given action in a given state; as mentioned earlier, such values are also called **Q-values**. We will use the notation $Q(a,i)$ to denote the value of doing action a in state i . Q-values are directly related to utility values by the following equation:

$$U(i) = \max_a Q(a, i) \quad (20.5)$$

Q-values play an important role in reinforcement learning for two reasons: first, like condition-action rules, they suffice for decision making without the use of a model; second, unlike condition-action rules, they can be learned directly from reward feedback.

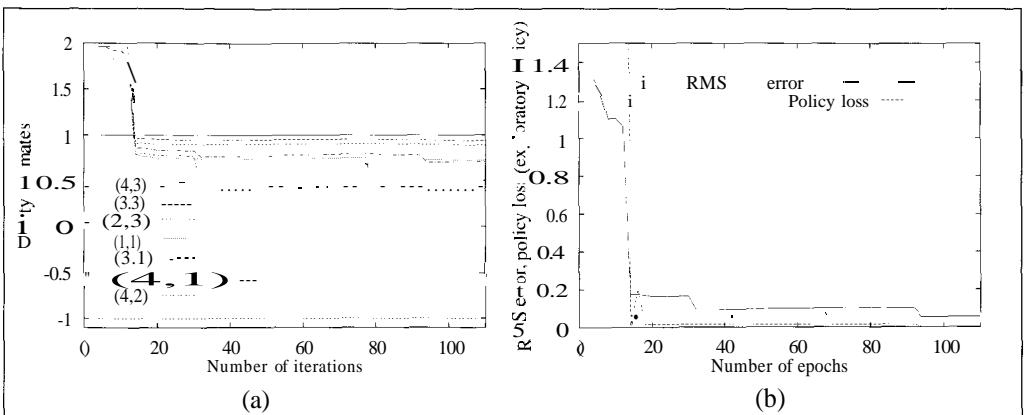


Figure 20.11 Performance of the exploratory ADP agent, using $R^+ = 2$ and $N_e = 5$. (a) Utility estimates for selected states over time. After the initial exploratory phase in which the states get an exploration bonus, the high-valued states quickly reach their correct values. The low-valued states converge slowly because they are seldom visited. (b) The RMS error in utility values and the associated policy loss.

As with utilities, we can write a constraint equation that must hold at equilibrium when the Q-values are correct:

$$Q(a, i) = R(i) + \sum_j M_{ij}^a \max_{a'} Q(a', j) \quad (20.6)$$

As in the ADP learning agent, we can use this equation directly as an update equation for an iteration process that calculates exact Q-values given an estimated model. This does, however, require that a model be learned as well because the equation uses M_{ij}^a . The temporal-difference approach, on the other hand, requires no model. The update equation for TD Q-learning is

$$Q(a, i) \leftarrow Q(a, i) + \alpha (R(i) + \max_{a'} Q(a', j) - Q(a, i)) \quad (20.7)$$

which is calculated after each transition from state i to state j .

The complete agent design for an exploratory Q-learning agent using TD is shown in Figure 20.12. Notice that it uses exactly the same exploration function f as used by the exploratory ADP agent, hence the need to keep statistics on actions taken (the table N). If a simpler exploration policy is used—say, acting randomly on some fraction of steps, where the fraction decreases over time—then we can dispense with the statistics.

Figure 20.13 shows the performance of the Q-learning agent in our 4×3 world. Notice that the utility estimates (derived from the Q-values using Equation (20.5)) take much longer to settle down than they did with the ADP agent. This is because TD does not enforce consistency among values via the model. Although a good policy is found after only 26 trials, it is considerably further from optimality than that found by the ADP agent (Figure 20.11).

Although these experimental results are for just one set of trials on one specific environment, they do raise a general question: is it better to learn a model and a utility function or to learn an action-value function with no model? In other words, what is the best way to represent the

```

function Q-LEARNING-AGENT( $e$ ) returns an action
  static:  $Q$ , a table of action values
     $N$ , a table of state-action frequencies
     $a$ , the last action taken
     $c$ , the previous state visited
     $r$ , the reward received in state  $i$ 

   $j \leftarrow \text{STATE}[e]$ 
  if  $c$  is non-null then
     $N[a, i] \leftarrow N[a, i] + 1$ 
     $Q[a, i] \leftarrow Q[a, i] + \alpha(r + \max_{a'} Q[a', j] - Q[a, i])$ 
  if TERMINAL? $[e]$  then
     $i \leftarrow \text{null}$ 
  else
     $i \leftarrow j$ 
     $r \leftarrow \text{REWARD}[e]$ 
     $a \leftarrow \arg \max_{a'} f(Q[a', j], N[a', j])$ 
  return  $a$ 

```

Figure 20.12 An exploratory Q-learning agent. It is an active learner that learns the value $Q(a, i)$ of each action in each situation. It uses the same exploration function f as the exploratory ADP agent, but avoids having to learn the transition model M_{ij}^a because the Q-value of s state can be related directly to those of its neighbors.

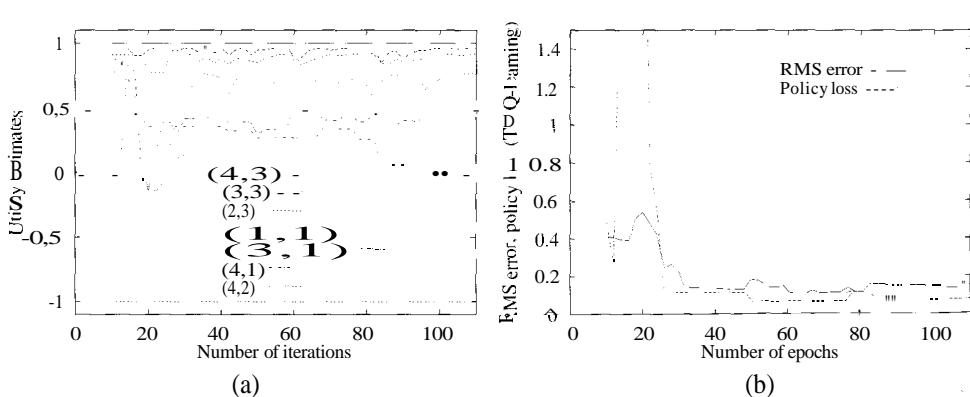


Figure 20.13 Performance of the exploratory TD Q-learning agent, using $R^+ = 2$ and $N_e = 5$. (a) Utility estimates for selected states over time. (b) The RMS error in utility values and the associated policy loss.

agent function? This is an issue at the foundations of artificial intelligence. As we stated in Chapter 1, one of the key historical characteristics of much of AI research is its (often unstated) adherence to the **knowledge-based** approach. This amounts to an assumption that the best way to represent the agent function is to construct an explicit representation of at least some aspects of the environment in which the agent is situated.

Some researchers, both inside and outside AI, have claimed that the availability of model-free methods such as Q-learning means that the knowledge-based approach is unnecessary. There is, however, little to go on but intuition. Our intuition, for what it's worth, is that as the environment becomes more complex, the advantages of a knowledge-based approach become more apparent. This is borne out even in games such as chess, checkers (draughts), and backgammon (see next section), where efforts to learn an evaluation function using a model have met with more success than Q-learning methods. Perhaps one day there will be a deeper theoretical understanding of the advantages of explicit knowledge; but as yet we do not even have a formal definition of the difference between model-based and model-free systems. All we have are some purported examples of each.

20.7 GENERALIZATION IN REINFORCEMENT LEARNING

EXPLICIT
REPRESENTATION

So far we have assumed that all the functions learned by the agents (U, M, R, Q) are represented in tabular form—that is, an **explicit representation** of one output value for each input tuple. Such an approach works reasonably well for small state spaces, but the time to convergence and (for ADP) the time per iteration increase rapidly as the space gets larger. With carefully controlled, approximate ADP methods, it may be possible to handle 10,000 states or more. This suffices for two-dimensional, maze-like environments, but more realistic worlds are out of the question. Chess and backgammon are tiny subsets of the real world, yet their state spaces contain on the order of 10^{50} to 10^{120} states. It would be absurd to suppose that one must visit all these states in order to learn how to play the game!

IMPLICIT
REPRESENTATION

The only way to handle such problems is to use an **implicit representation** of the function—a form that allows one to calculate the output for any input, but that is usually much more compact than the tabular form. For example, an estimated utility function for game playing can be represented as a weighted linear function of a set of board features f_1, \dots, f_n :

$$U(i) = w_1 f_1(i) + w_2 f_2(i) + \dots + w_n f_n(i)$$



INPUT
GENERALIZATION

Thus, instead of, say, 10^{120} values, the utility function is characterized by the n weights. A typical chess evaluation function might only have about 10 weights, so this is an *enormous* compression. *The compression achieved by an implicit representation allows the learning agent to generalize from states it has visited to states it has not visited.* That is, the most important aspect of an implicit representation is not that it takes up less space, but that it allows for inductive generalization over input states. For this reason, methods that learn such representations are said to perform **input generalization**. To give you some idea of the power of input generalization: by examining only one in 10^{44} of the possible backgammon states, it is possible to learn a utility function that allows a program to play as well as any human (Tesauro, 1992).

On the flip side, of course, there is the problem that there may be no function in the chosen space of implicit representations that faithfully approximates the true utility function. As in all inductive learning, there is a trade-off between the size of the hypothesis space and the time it takes to learn the function. A larger hypothesis space increases the likelihood that a good approximation can be found, but also means that convergence is likely to be delayed.

Let us now consider exactly how the inductive learning problem should be formulated. We begin by considering how to learn utility and action-value functions, and then move on to learning the transition function for the environment.

In the LMS (least mean squares) approach, the formulation is straightforward. At the end of each training sequence, the LMS algorithm associates a reward-to-go with each state visited along the way. The $(state, reward)$ pair can be used directly as a labelled example for any desired inductive learning algorithm. This yields a utility function $U(i)$.

It is also possible for a TD (temporal-difference) approach to apply inductive learning directly, once the U and/or Q tables have been replaced by implicit representations. The values that would be inserted into the tables by the update rules (20.2 and 20.7) can be used instead as labelled examples for a learning algorithm. The agent has to use the learned function on the next update, so the learning algorithm must be incremental.

One can also take advantage of the fact that the TD update rules provide small changes in the value of a given state. This is especially true if the function to be learned is characterized by a vector of weights w (as in linear weighted functions and neural networks). Rather than update a single tabulated value of U , as in Equation (20.2), we simply adjust the weights to try to reduce the temporal difference in U between successive states. Suppose that the parameterized utility function is $U_w(i)$. Then after a transition $i \rightarrow j$, we apply the following update rule:

$$w \leftarrow w + \alpha [r + U_w(j) - U_w(i)] \nabla_w U_w(i) \quad (20.8)$$

This form of updating performs gradient descent in weight space, trying to minimize the observed local error in the utility estimates. A similar update rule can be used for Q-learning (Exercise 20.9). Because the utility and action-value functions have real-valued outputs, neural networks and other continuous function representations are obvious candidates for the performance element. Decision-tree learning algorithms that provide real-valued output can also be used (see for example Quinlan's (1993) **model trees**), but cannot use the gradient descent method.

MODEL TREES

The formulation of the inductive learning problem for constructing a model of the environment is also very straightforward. Each transition provides the agent with the next state (at least in an accessible environment), so that labelled examples consist of a state-action pair as input and a state as output. It is not so easy, however, to find a suitable implicit representation for the model. In order to be useful for value and policy iteration and for the generation of pseudo-experiences in TD learning, the output state description must be sufficiently detailed to allow prediction of outcomes several steps ahead. Simple parametric forms cannot usually sustain this kind of reasoning. Instead, it may be necessary to learn general action models in the logical form used in Chapters 7 and 11. In a nondeterministic environment, one can use the conditional-probability-table representation of state evolution typical of dynamic belief networks (Section 17.5), in which generalization is achieved by describing the state in terms of a large set of features and using only sparse connections. Although model-based approaches have advantages in terms of their ability to learn value functions quickly, they are currently hampered by a lack

of suitable inductive generalization methods for learning the model. It is also not obvious how methods such as value and policy iteration can be applied with a generalized model.

We now turn to examples of large-scale applications of reinforcement learning. We will see that in cases where a utility function (and hence a model) is used, the model is usually taken as given. For example, in learning an evaluation function for backgammon, it is normally assumed that the legal moves, and their effects, are known in advance.

Applications to game-playing

The first significant application of reinforcement learning was also the first significant learning program of any kind—the checker-playing program written by Arthur Samuel (1959; 1967). Samuel first used a weighted linear function for the evaluation of positions, using up to 16 terms at any one time. He applied a version of Equation (20.8) to update the weights. There were some significant differences, however, between his program and current methods. First, he updated the weights using the difference between the current state and the backed-up value generated by full lookahead in the search tree. This works fine, because it amounts to viewing the state space at a different granularity. A second difference was that the program did *not* use any observed rewards! That is, the values of terminal states were ignored. This means that it is quite possible for Samuel's program not to converge, or to converge on a strategy designed to lose rather than win. He managed to avoid this fate by insisting that the weight for material advantage should always be positive. Remarkably, this was sufficient to direct the program into areas of weight space corresponding to good checker play (see Chapter 5).

The TD-gammon system (Tesauro, 1992) forcefully illustrates the potential of reinforcement learning techniques. In earlier work (Tesauro and Sejnowski, 1989), Tesauro tried learning a neural network representation of $Q(a, i)$ directly from examples of moves labelled with relative values by a human expert. This approach proved extremely tedious for the expert. It resulted in a program, called Neurogammon, that was strong by computer standards but not competitive with human grandmasters. The TD-gammon project was an attempt to learn from self-play alone. The only reward signal was given at the end of each game. The evaluation function was represented by a fully connected neural network with a single hidden layer containing 40 nodes. Simply by repeated application of Equation (20.8), TD-gammon learned to play considerably better than Neurogammon, even though the input representation contained just the raw board position with no computed features. This took about 200,000 training games and two weeks of computer time. Although this may seem like a lot of games, it is only a vanishingly small fraction of the state space. When precomputed features were added to the input representation, a network with 80 hidden units was able, after 300,000 training games, to reach a standard of play comparable with the top three human players worldwide.

Application to robot control

The setup for the famous **cart-pole** balancing problem, also known as the **inverted pendulum**, is shown in Figure 20.14. The problem is to control the position x of the cart so that the pole stays roughly upright ($\theta \approx \pi/2$), while staying within the limits of the cart track as shown. This problem has been used as a test bed for research in control theory as well as reinforcement

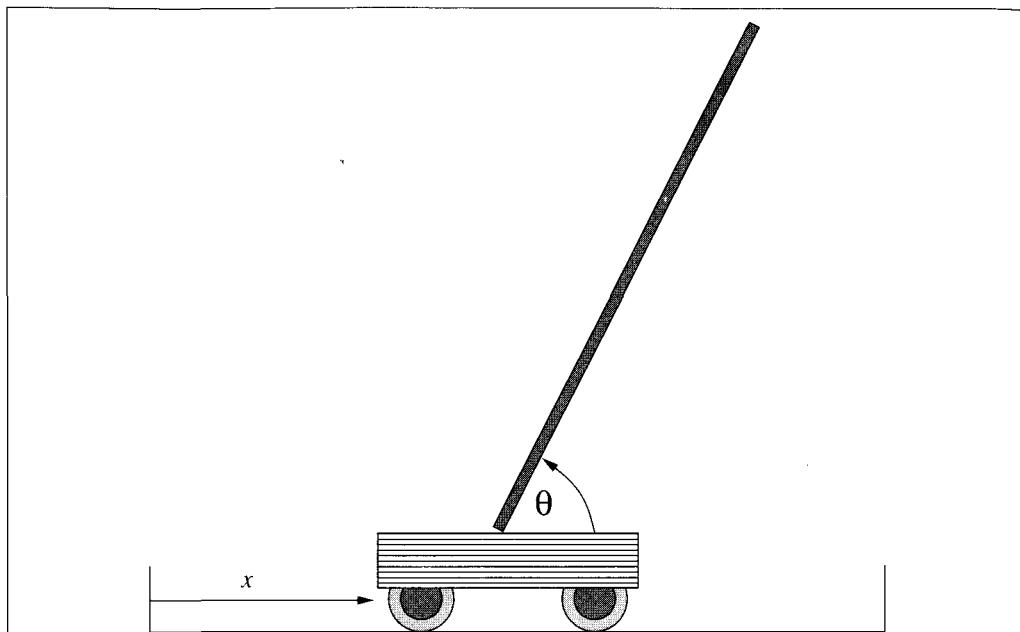


Figure 20.14 Setup for the problem of balancing a long pole on top of a moving cart. The cart can be jerked left or right by a controller that observes x , \dot{x} , \ddot{x} , and $\ddot{\theta}$.

BANG-BANG CONTROL

learning, and over 200 papers have been published on it. The cart-pole problem differs from the problems described earlier in that the state variables x , \dot{x} , \ddot{x} , and θ are continuous. The actions are usually discrete—jerk left or jerk right, the so-called **bang-bang control** regime.

The earliest work on learning for this problem was carried out by Michie and Chambers (1968). Their BOXES algorithm was able to balance the pole for over an hour after only about 30 trials. Moreover, unlike many subsequent systems, BOXES was implemented using a real cart and pole, not a simulation. The algorithm first discretized the four-dimensional state space into boxes, hence the name. It then ran trials until the pole fell over or the cart hit the end of the track. Negative reinforcement was associated with the final action in the final box, and then propagated back through the sequence. It was found that the discretization causes some problems when the apparatus was initialized in a different position from those used in training, suggesting that generalization was not perfect. Improved generalization and faster learning can be obtained using an algorithm that *adaptively* partitions the state space according to the observed variation in the reward.

More recently, neural networks have been used to provide a continuous mapping from the state space to the actions, with slightly better results. The most impressive performance, however, belongs to the control algorithm derived using classical control theory for the *triple* inverted pendulum, in which three poles are balanced one on top of another with torque controls at the joints (Furuta *et al.*, 1984). (One is disappointed, but not surprised, that this algorithm was implemented only in simulation.)

20.8 GENETIC ALGORITHMS AND EVOLUTIONARY PROGRAMMING

Nature has a robust way of evolving successful organisms. The organisms that are ill-suited for an environment die off, whereas the ones that are fit live to reproduce. Offspring are similar to their parents, so each new generation has organisms that are similar to the fit members of the previous generation. If the environment changes slowly, the species can gradually evolve along with it, but a sudden change in the environment is likely to wipe out a species. Occasionally, random mutations occur, and although most of these mean a quick death for the mutated individual, some mutations lead to new successful species. The publication of Darwin's *The Origin of Species on the Basis of Natural Selection* was a major turning point in the history of science.

It turns out that what's good for nature is also good for artificial systems. Figure 20.15 shows the GENETIC-ALGORITHM, which starts with a set of one or more individuals and applies selection and reproduction operators to "evolve" an individual that is successful, as measured by a **fitness function**. There are several choices for what the individuals are. They can be entire agent functions, in which case the fitness function is a performance measure or reward function, and the analogy to natural selection is greatest. They can be component functions of an agent, in which case the fitness function is the critic. Or they can be anything at all that can be framed as an optimization problem.

Since the evolutionary process learns an agent function based on occasional rewards (offspring) as supplied by the selection function, it can be seen as a form of reinforcement learning. Unlike the algorithms described in the previous sections, however, no attempt is made to learn the relationship between the rewards and the actions taken by the agent or the states of the environment. GENETIC-ALGORITHM simply searches directly in the space of individuals, with the goal of finding one that maximizes the fitness function. The search is parallel because each individual in the population can be seen as a separate search. It is hill climbing because we are making small genetic changes to the individuals and using the best resulting offspring. The key question is how to allocate the searching resources: clearly, we should spend most of our time on the most promising individuals, but if we ignore the low-scoring ones, we risk getting stuck on a local maximum. It can be shown that, under certain assumptions, the genetic algorithm allocates resources in an optimal way (see the discussion of *n*-armed bandits in, e.g., Goldberg (1989)).

Before we can apply GENETIC-ALGORITHM to a problem, we need to answer the following four questions:

- What is the fitness function?
- How is an individual represented?
- How are individuals selected?
- How do individuals reproduce?

The fitness function depends on the problem, but in any case, it is a function that takes an individual as input and returns a real number as output.

In the "classic" genetic algorithm approach, an individual is represented as a string over a finite alphabet. Each element of the string is called a **gene**. In real DNA, the alphabet is AGTC (adenine, guanine, thymine, cytosine), but in genetic algorithms, we usually use the binary alphabet (0,1). Some authors reserve the term "genetic algorithm" for cases where the representation

```

function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
           FITNESS-FN, a function that measures the fitness of an individual

  repeat
    parents  $\leftarrow$  SELECTION(population, FITNESS-FN)
    population  $\leftarrow$  REPRODUCTION(parents)
  until some individual is fit enough
  return the best individual in population, according to FITNESS-FN

```

Figure 20.15 The genetic algorithm finds a fit individual using simulated evolution.

EVOLUTIONARY
PROGRAMMING

is a bit string, and use the term **evolutionary programming** when the representation is more complicated. Other authors make no distinction, or make a slightly different one.

The selection strategy is usually randomized, with the probability of selection proportional to fitness. That is, if individual *X* scores twice as high as *Y* on the fitness function, then *X* is twice as likely to be selected for reproduction than is *Y*. Usually, selection is done with replacement, so that a very fit individual will get to reproduce several times.

CROSS-OVER

Reproduction is accomplished by cross-over and mutation. First, all the individuals that have been selected for reproduction are randomly paired. Then for each pair, a cross-over point is randomly chosen. Think of the genes of each parent as being numbered from 1 to *N*. The **cross-over** point is a number in that range; let us say it is 10. That means that one offspring will get genes 1 through 10 from the first parent, and the rest from the second parent. The second offspring will get genes 1 through 10 from the second parent, and the rest from the first. However, each gene can be altered by random **mutation** to a different value, with small independent probability. Figure 20.16 diagrams the process.

MUTATION

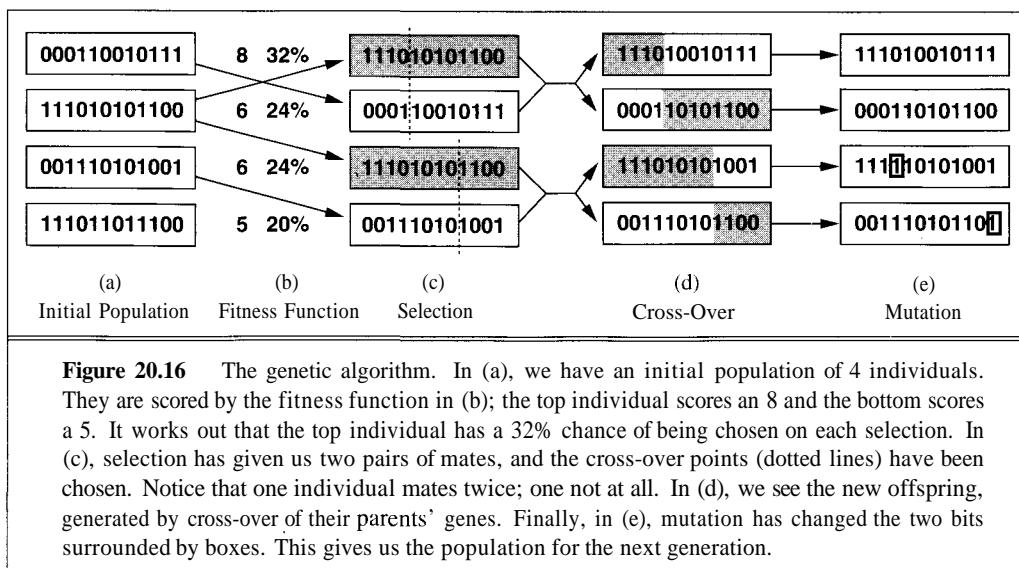
For example, suppose we are trying to learn a decision list representation for the restaurant waiting problem (see page 556). The fitness function in this case is simply the number of examples that an individual is consistent with. The representation is the tricky part. There are ten attributes in the problem, but not all of them are binary. It turns out that we need 5 bits to represent each distinct attribute/value pair:

```

00000 : Alternate(x)
00001 :  $\neg$ Alternate(x)
:
10111 : WaitEstimate(x, 0–10)
11000 : WaitEstimate(x, 10–30)
11001 : WaitEstimate(x, 30–60)
11010 : WaitEstimate(x, >60)

```

We also need one bit for each test to say if the outcome is *Yes* or *No*. Thus, if we want to represent a *k*-DL with a length of up to *t* tests, we need a representation with $t(5k + 1)$ bits. We can use the standard selection and reproduction approaches. Mutation can flip an outcome or change an attribute. Cross-over combines the head of one decision list with the tail of another.



Like neural networks, genetic algorithms are easy to apply to a wide range of problems. The results can be very good on some problems, and rather poor on others. In fact, Denker's remark that "neural networks are the second best way of doing just about anything" has been extended with "and genetic algorithms are the third." But don't be afraid to try a quick implementation of a genetic algorithm on a new problem—just to see if it does work—before investing more time thinking about another approach.

20.9 SUMMARY

This chapter has examined the reinforcement learning problem—how an agent can become proficient in an unknown environment given only its percepts and occasional rewards. Reinforcement learning can be viewed as a microcosm for the entire AI problem, but is studied in a number of simplified settings to facilitate progress. The following major points were made:

- The overall agent design dictates the kind of information that must be learned. The two main designs studied are the model-based design, using a model M and a utility function U , and the model-free approach, using an action-value function Q .
- The utility of a state is the expected sum of rewards received between now and termination of the sequence.
- Utilities can be learned using three approaches.
 1. The LMS (least-mean-square) approach uses the total observed reward-to-go for a given state as direct evidence for learning its utility. LMS uses the model only for the purposes of selecting actions.

2. The ADP (adaptive dynamic programming) approach uses the value or policy iteration algorithm to calculate exact utilities of states given an estimated model. ADP makes optimal use of the local constraints on utilities of states imposed by the neighborhood structure of the environment.
 3. The TD (temporal-difference) approach updates utility estimates to match those of successor states, and can be viewed as a simple approximation to the ADP approach that requires no model for the learning process. Using the model to generate pseudo-experiences can, however, result in faster learning.
- Action-value functions, or Q-functions, can be learned by an ADP approach or a TD approach. With TD, Q-learning requires no model in either the learning or action-selection phases. This simplifies the learning problem but potentially restricts the ability to learn in complex environments.
 - When the learning agent is responsible for selecting actions while it learns, it must trade off the estimated value of those actions against the potential for learning useful new information. Exact solution of the exploration problem is infeasible, but some simple heuristics do a reasonable job.
 - In large state spaces, reinforcement learning algorithms must use an implicit functional representation in order to perform input generalization over states. The temporal-difference signal can be used directly to direct weight changes in parametric representations such as neural networks.
 - Combining input generalization with an explicit model has resulted in excellent performance in complex domains.
 - Genetic algorithms achieve reinforcement learning by using the reinforcement to increase the proportion of successful functions in a population of programs. They achieve the effect of generalization by mutating and cross-breeding programs with each other.

Because of its potential for eliminating hand coding of control strategies, reinforcement learning continues to be one of the most active areas of machine learning research. Applications in robotics promise to be particularly valuable. As yet, however, there is little understanding of how to extend these methods to the more powerful performance elements described in earlier chapters. Reinforcement learning in *inaccessible* environments is also a topic of current research.

BIBLIOGRAPHICAL AND HISTORICAL NOTLS

Arthur Samuel's work (1959) was probably the earliest successful machine learning research. Although this work was informal and had a number of flaws, it contained most of the modern ideas in reinforcement learning, including temporal differencing and input generalization. Around the same time, researchers in adaptive control theory (Widrow and Hoff, 1960), building on work by Hebb (1949), were training simple networks using the LMS rule. (This early connection between neural networks and reinforcement learning may have led to the persistent misperception that the latter is a subfield of the former.) The cart-pole work of Michie and Chambers (1968) can also be seen as a reinforcement learning method with input generalization.

A more recent tradition springs from work at the University of Massachusetts in the early 1980s (Barto *et al.*, 1981). The paper by Sutton (1988) reinvigorated reinforcement learning research in AI, and provides a good historical overview. The Ph.D. theses by Watkins (1989) and Kaelbling (1990) and the survey by Barto *et al.* (1991) also contain good reviews of the field. Watkin's thesis originated Q-learning, and proved its convergence in the limit. Some recent work appears in a special issue of *Machine Learning* (Vol. 8, Nos. 3/4, 1992), with an excellent introduction by Sutton. The presentation in this chapter is heavily influenced by Moore and Atkeson (1993), who make a clear connection between temporal differencing and classical dynamic programming techniques. The latter paper also introduced the idea of prioritized sweeping. An almost identical method was developed independently by Peng and Williams (1993). Bandit problems, which model the problem of exploration, are analyzed in depth by Berry and Fristedt (1985).

Reinforcement learning in games has also undergone a renaissance in recent years. In addition to Tesauro's work, a world-class Othello system was developed by Lee and Mahajan (1988). Reinforcement learning papers are published frequently in the journal *Machine Learning*, and in the International Conferences on Machine Learning.

Genetic algorithms originated in the work of Friedberg (1958), who attempted to produce learning by mutating small FORTRAN programs. Since most mutations to the programs produced inoperative code, little progress was made. John Holland (1975) reinvigorated the field by using bit-string representations of agents such that any possible string represented a functioning agent. John Koza (1992) has championed more complex representations of agents coupled with mutation and mating techniques that pay careful attention to the syntax of the representation language. Current research appears in the annual Conference on Evolutionary Programming.

EXERCISES

 **20.1** Show that the estimates developed by the LMS-UPDATE algorithm do indeed minimize the mean square error on the training data.

20.2 Implement a passive learning agent in a simple environment, such as that shown in Figure 20.1. For the case of an initially unknown environment model, compare the learning performance of the LMS, TD, and ADP algorithms.

20.3 Starting with the passive ADP agent, modify it to use an approximate ADP algorithm as discussed in the text. Do this in two steps:

- Implement a priority queue for adjustments to the utility estimates. Whenever a state is adjusted, all of its predecessors also become candidates for adjustment, and should be added to the queue. The queue is initialized using the state from which the most recent transition took place. Change ADP-UPDATE to allow only a fixed number of adjustments.
- Experiment with various heuristics for ordering the priority queue, examining their effect on learning rates and computation time.

20.4 The environments used in the chapter all assume that training sequences are finite. In environments with no clear termination point, the unlimited accumulation of rewards can lead to problems with infinite utilities. To avoid this, a discount factor γ is often used, where $\gamma < 1$. A reward k steps in the future is discounted by a factor of γ^k . For each constraint and update equation in the chapter, explain how to incorporate the discount factor.

20.5 The description of reinforcement learning agents in Section 20.1 uses distinguished terminal states to indicate the end of a training sequence. Explain how this additional complication could be eliminated by modelling the "reset" as a transition like any other. How will this affect the definition of utility?

20.6 Prove formally that Equations (20.1) and (20.3) are consistent with the definition of utility as the expected reward-to-go of a state.

20.7 How can the value determination algorithm be used to calculate the expected loss experienced by an agent using a given set of utility estimates U and an estimated model M , compared to an agent using correct values?

20.8 Adapt the vacuum world (Chapter 2) for reinforcement learning by including rewards for picking up each piece of dirt and for getting home and switching off. Make the world accessible by providing suitable percepts. Now experiment with different reinforcement learning agents. Is input generalization necessary for success?

20.9 Write down the update equation for Q-learning with a parameterized implicit representation. That is, write the counterpart to Equation (20.8).



20.10 Extend the standard game-playing environment (Chapter 5) to incorporate a reward signal. Put two reinforcement learning agents into the environment (they may of course share the agent program) and have them play against each other. Apply the generalized TD update rule (Equation (20.8)) to update the evaluation function. You may wish to start with a simple linear weighted evaluation function, and a simple game such as tic-tac-toe.

20.11 (Discussion topic.) Is reinforcement learning an appropriate abstract model for human learning? For evolution?

21

KNOWLEDGE IN LEARNING

In which we examine the problem of learning when you already know something.

PRIOR KNOWLEDGE

In all of the approaches to learning described in the previous three chapters, the idea is to construct a program that has the input/output behavior observed in the data. We have seen how this general problem can be solved for simple logical representations, for neural networks, and for belief networks. In each case, the learning methods can be understood as searching a hypothesis space to find a suitable program. The learning methods also made few assumptions, if any, concerning the nature of the correct program. In this chapter, we go beyond these approaches to study learning methods that can take advantage of **prior knowledge** about the environment. We also examine learning algorithms that can learn general first-order logical theories. These are essential steps toward a truly autonomous intelligent agent.

21.1 KNOWLEDGE IN LEARNING

We begin by examining the ways in which prior knowledge can get into the act. In order to do this, it will help to have a general *logical* formulation of the learning problem, as opposed to the *function-learning* characterization of pure inductive inference given in Section 18.2. The reason that a logical characterization is helpful is that it provides a very natural way to specify *partial* information about the function to be learned. This is analogous to the distinction between problem solving (which uses a "black-box" functional view of states and goals) and planning (which opens up the black boxes and uses logical descriptions of states and actions).

Recall from Section 18.5 that examples are composed of descriptions and classifications. The object of inductive learning in the logical setting is to find a hypothesis that explains the classifications of the examples, given their descriptions. We can make this logically precise as follows. If we use *Descriptions* to denote the conjunction of all the example descriptions, and *Classifications* to denote the conjunction of all the example classifications, then the *Hypothesis* must satisfy the following property:

$$\text{Hypothesis A Descriptions} \models \text{Classifications}$$

(21.1)

ENTAILMENT
CONSTRAINT

We call this kind of relationship an **entailment constraint**, in which *Hypothesis* is the "unknown." Pure inductive learning means solving this constraint, where *Hypothesis* is drawn from some predefined hypothesis space. For example, if we consider a decision tree as a logical formula (see page 532), then a decision tree that is consistent with all the examples will satisfy Equation (21.1). If we place *no* restrictions on the logical form of the hypothesis, of course, then *Hypothesis* = *Classifications* also satisfies the constraint. Normally, Ockham's razor tells us to prefer *small*, consistent hypotheses, so we try to do better than simply memorizing the examples.

This simple picture of inductive learning persisted until the early 1980s. The modern approach is to design agents that *already know something* and are trying to learn some more. This may not sound like a terrifically deep insight, but it makes quite a difference to the way in which we write programs. It might also have some relevance to our theories about how science itself works. The general idea is shown schematically in Figure 21.1.

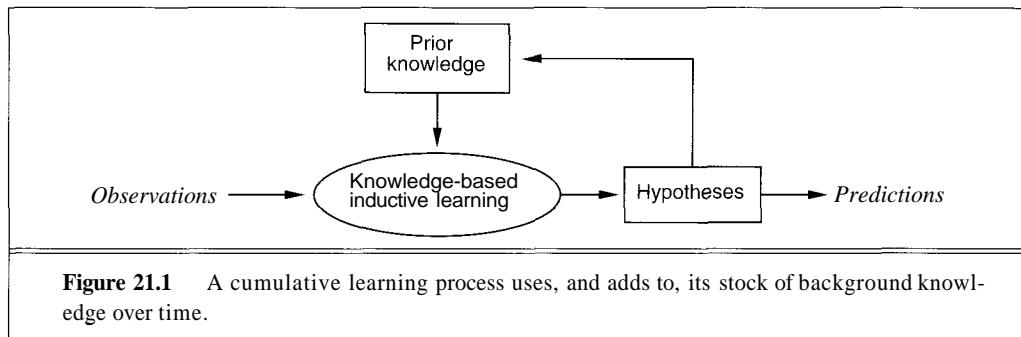


Figure 21.1 A cumulative learning process uses, and adds to, its stock of background knowledge over time.

If we want to build an autonomous learning agent that uses background knowledge, the agent must have some method for obtaining the background knowledge in the first place, in order for it to be used in the new learning episodes. This method must itself be a learning process. The agent's life history will therefore be characterized by *cumulative*, or *incremental*, development. Presumably, the agent could start out with nothing, performing inductions *in vacuo* like a good little pure induction program. But once it has eaten from the Tree of Knowledge, it can no longer pursue such naive speculations, and should use its background knowledge to learn more and more effectively. The question is then how to actually do this.

Some simple examples

Let us consider some commonsense examples of learning with background knowledge. Many apparently rational cases of inferential behavior in the face of observations clearly do not follow the simple principles of pure induction.

- Sometimes one leaps to general conclusions after only one observation. Gary Larson once drew a cartoon in which a bespectacled caveman, Thag, is roasting his lizard on the end of a pointed stick. He is watched by an amazed crowd of his less intellectual contemporaries, who have been using their bare hands to hold their viuctuals over the fire. This enlightening experience is enough to convince the watchers of a general principle of painless cooking.

- Or consider the case of the traveller to Brazil meeting her first Brazilian. On hearing him speak Portuguese, she immediately concludes that Brazilians speak Portuguese, yet on discovering that his name is Fernando, she does not conclude that all Brazilians are called Fernando. Similar examples appear in science. For example, when a freshman physics student measures the density and conductance of a sample of copper at a particular temperature, she is quite confident in generalizing those values to all pieces of copper. Yet when she measures its mass, she does not even consider the hypothesis that all pieces of copper have that mass. On the other hand, it would be quite reasonable to make such a generalization over all pennies.
- Finally, consider the case of a pharmacologically ignorant but diagnostically sophisticated medical student observing a consulting session between a patient and an expert internist. After a series of questions and answers, the expert tells the patient to take a course of a particular antibiotic. The medical student infers the general rule that that particular antibiotic is effective for a particular type of infection.

These are all cases in which *the use of background knowledge allows much faster learning than one might expect from a pure induction program.*



Some general schemes

In each of the preceding examples, one can appeal to prior knowledge to try to justify the generalizations chosen. We will now look at what kinds of entailment constraints are operating in each of these cases. The constraints will involve the *Background* knowledge, in addition to the *Hypothesis* and the observed *Descriptions* and *Classifications*.

In the case of lizard toasting, the cavemen generalize by *explaining* the success of the pointed stick: it supports the lizard while keeping the hand intact. From this explanation, they can infer a general rule: that any long, thin, rigid, sharp object can be used to toast small, soft-bodied edibles. This kind of generalization process has been called **explanation-based learning**, or **EBL**. Notice that the general rule *follows logically* from the background knowledge possessed by the cavemen. Hence, the entailment constraints satisfied by EBL are the following:

$$\begin{aligned} \textit{Hypothesis A Descriptions} &\models \textit{Classifications} \\ \textit{Background} &\models \textit{Hypothesis} \end{aligned}$$

The first constraint looks the same as Equation (21.1), so EBL was initially thought to be a better way to learn from examples. But because it requires that the background knowledge be sufficient to explain the *Hypothesis*, which in turn explains the observations, the agent does not actually learn anything *factually new* from the instance. The agent *could have* derived the example from what it already knew, although that might have required an unreasonable amount of computation. EBL is now viewed as a method for converting first-principles theories into useful, special-purpose knowledge. We describe algorithms for EBL in Section 21.2.

The situation of our traveller in Brazil is quite different. For she cannot necessarily explain why Fernando speaks the way he does, unless she knows her Papal bulls. But the same generalization would be forthcoming from a traveller entirely ignorant of colonial history. The relevant prior knowledge in this case is that, within any given country, most people tend to speak

RELEVANCE

RELEVANCE-BASED LEARNING



KNOWLEDGE-BASED INDUCTIVE LEARNING

INDUCTIVE LOGIC PROGRAMMING

the same language; on the other hand, Fernando is not assumed to be the name of all Brazilians because this kind of regularity does not hold for names. Similarly, the freshman physics student also would be hard put to explain the particular values that she discovers for the conductance and density of copper. She does know, however, that the material of which an object is composed and its temperature together determine its conductance. In each case, the prior knowledge *Background* concerns the **relevance** of a set of features to the goal predicate. This knowledge, *together with the observations*, allows the agent to infer a new, general rule that explains the observations:

$$\begin{aligned} \textit{Hypothesis A Descriptions} &\models \textit{Classifications} \\ \textit{Background A Descriptions A Classifications} &\models \textit{Hypothesis} \end{aligned} \tag{21.2}$$

We call this kind of generalization **relevance-based learning**, or **RBL** (although the name is not standard). Notice that whereas RBL does make use of the content of the observations, it does not produce hypotheses that go beyond the logical content of the background knowledge and the observations. It is a *deductive* form of learning, and cannot by itself account for the creation of new knowledge starting from scratch. We discuss applications of RBL in Section 21.3.

In the case of the medical student watching the expert, we assume that the student's prior knowledge is sufficient to infer the patient's disease D from the symptoms. This is not, however, enough to explain the fact that the doctor prescribes a particular medicine M . The student needs to propose another rule, namely, that M generally is effective against D . Given this rule, and the student's prior knowledge, the student can now explain why the expert prescribes M in this particular case. We can generalize this example to come up with the entailment constraint:

$$\textit{Background A Hypothesis A Descriptions} \models \textit{Classifications} \tag{21.3}$$

That is, *the background knowledge and the new hypothesis combine to explain the examples*. As with pure inductive learning, the learning algorithm should propose hypotheses that are as simple as possible, consistent with this constraint. Algorithms that satisfy constraint 21.3 are called **knowledge-based inductive learning**, or **KBIL**, algorithms.

KBIL algorithms, which are described in detail in Section 21.4, have been studied mainly in the field of **inductive logic programming** or **ILP**. In ILP systems, prior knowledge plays two key roles in reducing the complexity of learning:

1. Because any hypothesis generated must be consistent with the prior knowledge as well as with the new observations, the effective hypothesis space size is reduced to include only those theories that are consistent with what is already known.
2. For any given set of observations, the size of the hypothesis required to construct an explanation for the observations can be much reduced, because the prior knowledge will be available to help out the new rules in explaining the observations. The smaller the hypothesis, the easier it is to find.

In addition to allowing the use of prior knowledge in induction, ILP systems can formulate hypotheses in general first-order logic, rather than the restricted languages used in Chapter 18. This means that they can learn in environments that cannot be understood by simpler systems.

21.2 EXPLANATION-BASED LEARNING

As we explained in the introduction to this chapter, explanation-based learning is a method for extracting general rules from individual observations. As an example, consider the problem of differentiating and simplifying algebraic expressions (Exercise 10.4). If we differentiate an expression such as X^2 with respect to X , we obtain $2X$. (Notice that we use a capital letter for the arithmetic unknown X , to distinguish it from the logical variable x .) In a logical reasoning system, the goal might be expressed as $\text{ASK}(\text{Derivative}(X^2X) = d, KB)$, with solution $d = 2X$.

We can see this solution "by inspection" because we have many years of practice in solving such problems. A student encountering such problems for the first time, or a program with no experience, will have a much more difficult job. Application of the standard rules of differentiation eventually yields the expression $1 \times (2 \times (X^{(2-1)}))$, and eventually this simplifies to $2X$. In the authors' logic programming implementation, this takes 136 proof steps, of which 99 are on dead-end branches in the proof. After such an experience, we would like the program to solve the same problem much more quickly the next time.

MEMOIZATION

The technique of **memoization** has long been used in computer science to speed up programs by saving the results of computation. The basic idea of memo functions is to accumulate a database of input/output pairs; when the function is called, it first checks the database to see if it can avoid solving the problem from scratch. Explanation-based learning takes this a good deal further, by creating *general* rules that cover an entire class of cases. In the case of differentiation, memoization would remember that the derivative of X^2 with respect to X is $2X$, but would leave the agent to calculate the derivative of Z^2 with respect to Z from scratch. We would like to be able to extract the general rule¹ that for any arithmetic unknown u , the derivative of u^2 with respect to u is $2u$. In logical terms, this is expressed by the rule

$$\text{ArithmeticUnknown}(u) \Rightarrow \text{Derivative}(u^2, u) = 2u$$

If the knowledge base contains such a rule, then any new case that is an instance of this rule can be solved immediately.

This is, of course, merely a trivial example of a very general phenomenon. Once something is understood, it can be generalized and reused in other circumstances. It becomes an "obvious" step, and can then be used as a building block in solving still more complex problems. Alfred North Whitehead (1911), co-author with Bertrand Russell of *Principia Mathematica*, wrote that

"Civilization advances by extending the number of important operations that we can do without thinking about them," perhaps himself applying EBL to his understanding of events such as Thag's discovery. If you have understood the basic idea of the differentiation example, then your brain is already busily trying to extract the general principles of explanation-based learning from it. Notice that unless you are a good deal smarter than the authors, you hadn't *already* invented EBL before we showed you an example of it. Like the cavemen watching Thag, you (and we) needed an example before we could generate the basic principles. This is because *explaining why* something is a good idea is much easier than coming up with the idea in the first place.



¹ Of course, a general rule for u^2 can also be produced, but the current example suffices to make the point.

Extracting general rules from examples

The basic idea behind EBL is first to construct an explanation of the observation using prior knowledge, and then to establish a definition of the class of cases for which the same explanation structure can be used. This definition provides the basis for a rule covering all of the cases in the class. The "explanation" can be a logical proof, but more generally it can be any reasoning or problem-solving process whose steps are well-defined. The key is to be able to identify the necessary conditions for those same steps to apply to another case.

We will use for our reasoning system the simple backward-chaining theorem prover described in Chapter 9. The proof tree for $\text{Derivative}(X^2, X) \models d$ is too large to use as an example, so we will use a somewhat simpler problem to illustrate the generalization method. Suppose our problem is to simplify $1 \times (0 + X)$. The knowledge base includes the following rules:

```

Rewrite(u, v) A Simplify(vw) => Simplify(uw)
Primitive(u) => Simplify(uu)
ArithmeticUnknown(u) => Primitive(u)
Number(u) => Primitive(u)
Rewrite(1 x u, u)
Rewrite(0 + u, u)
⋮

```

The proof that the answer is X is shown in the top half of Figure 21.2. The EBL method actually constructs two proof trees simultaneously. The second proof tree uses a *variabilized* goal in which the constants from the original goal are replaced by variables. As the original proof proceeds, the variabilized proof proceeds using *exactly the same rule applications*. This may cause some of the variables to become instantiated. For example, in order to use the rule $\text{Rewrite}(1 \times u, u)$, the variable x in the subgoal $\text{Rewrite}(x x (y+z), v)$ must be bound to 1. Similarly, y must be bound to 0 in the subgoal $\text{Rewrite}(y+z, v')$ in order to use the rule $\text{Rewrite}(0 + u, u)$.

Once we have the generalized proof tree, we take the leaves (with the necessary bindings) and form a general rule for the goal predicate:

```

Rewrite(1 x (0 + z), 0 + z) A Rewrite(0 + z, z) A ArithmeticUnknown(z)
=> Simplify(1x (0 + z), z)

```

Notice that the first two conditions on the left-hand side are true *regardless of the value of z*. We can therefore drop them from the rule, yielding

```
ArithmeticUnknown(z) => Simplify(1x (0 + z), z)
```

In general, conditions can be dropped from the final rule if they impose no constraints on the variables on the right-hand side of the rule, because the resulting rule will still be true and will be more efficient. Notice that we cannot drop the condition $\text{ArithmeticUnknown}(z)$, because not all possible values of z are arithmetic unknowns. Values other than arithmetic unknowns might require different forms of simplification—for example, if z were 2×3 , then the correct simplification of $1 \times (0 + (2 \times 3))$ would be 6 and not 2×3 .

To recap, the basic EBL process works as follows:

- Given an example, construct a proof that the goal predicate applies to the example using the available background knowledge.

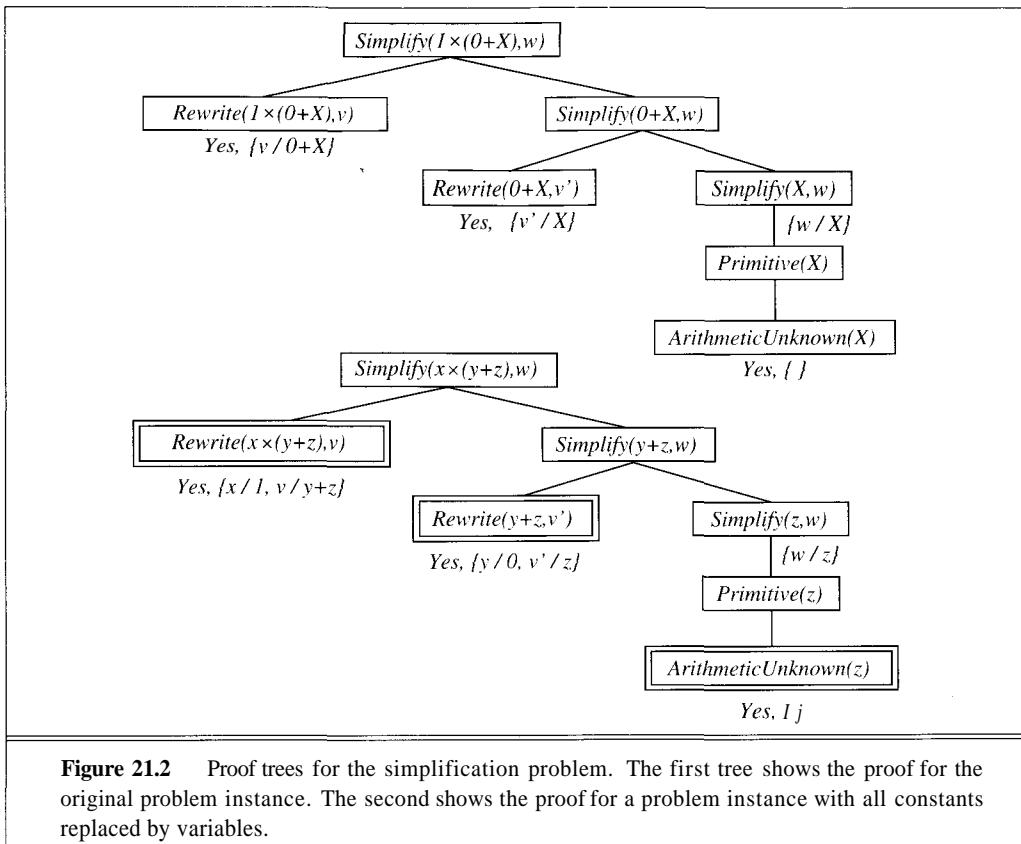


Figure 21.2 Proof trees for the simplification problem. The first tree shows the proof for the original problem instance. The second shows the proof for a problem instance with all constants replaced by variables.

2. In parallel, construct a generalized proof tree for the variabilized goal using the same inference steps as in the original proof.
3. Construct a new rule whose left-hand side consists of the leaves of the proof tree, and whose right-hand side is the variabilized goal (after applying the necessary bindings from the generalized proof).
4. Drop any conditions that are true regardless of the values of the variables in the goal.

Improving efficiency

Examining the generalized proof tree in Figure 21.2, we see that there is more than one generalized rule that can be extracted from the tree. For example, if we terminate, or **prune**, the growth of the right-hand branch in the proof tree when it reaches the *Primitive* step, we get the rule

$$\text{Primitive}(z) \Rightarrow \text{Simplify}(1x (0 + z), z)$$

This rule is equally valid, but also *more general* than the rule using *ArithmeticUnknown*, because it covers cases where *z* is a number. We can extract a still more general rule by pruning after the

step $\text{Simplify}(y + z, w)$, yielding the rule

$$\text{Simplify}(y + z, w) \Rightarrow \text{Simplify}(\text{Ix } (y + z), w)$$

In general, a rule can be extracted from *any partial subtree* of the generalized proof tree. Now we have a problem: which of these rules do we choose?

The choice of which rule to generate comes down to the question of efficiency. There are three factors involved in the analysis of efficiency gains from EBL:

1. Adding large numbers of rules to a knowledge base can slow down the reasoning process, because the inference mechanism must still check those rules even in cases where they do not yield a solution. In other words, it increases the **branching factor** in the search space.
2. To compensate for this, the derived rules must offer significant increases in speed for the cases that they do cover. This mainly comes about because the derived rules avoid dead ends that would otherwise be taken, as well as shortening the proof itself.
3. Derived rules should also be as general as possible, so that they apply to the largest possible set of cases.

OPERATIONALITY

A common approach to ensuring that derived rules are efficient is to insist on the **operationality** of each subgoal in the rule. A subgoal is operational, roughly speaking, if it is "easy" to solve. For example, the subgoal $\text{Primitive}(z)$ is easy to solve, requiring at most two steps, whereas the subgoal $\text{Simplify}(y + z, w)$ could lead to an arbitrary amount of inference, depending on the values of y and z . If a test for operationality is carried out at each step in the construction of the generalized proof, then we can prune the rest of a branch as soon as an operational subgoal is found, keeping just the operational subgoal as a conjunct of the new rule.

Unfortunately, there is usually a trade-off between operationality and generality. More specific subgoals are usually easier to solve but cover fewer cases. Also, operationality is a matter of degree; one or two steps is definitely operational, but what about 10, or 100? Finally, the cost of solving a given subgoal depends on what other rules are available in the knowledge base. It can go up or down as more rules are added. Thus, EBL systems really face a very complex optimization problem in trying to maximize the efficiency of a given initial knowledge base. It is sometimes possible to derive a mathematical model of the effect on overall efficiency of adding a given rule, and to use this model to select the best rule to add. The analysis can become very complicated, however, especially when recursive rules are involved. One promising approach is to address the problem of efficiency empirically, simply by adding several rules and seeing which ones are useful and actually speed things up.

The idea of empirical analysis of efficiency is actually at the heart of EBL. What we have been calling loosely the "efficiency of a given knowledge base" is actually the average-case complexity on a population of problems that the agent will have to solve. By generalizing from past example problems, EBL makes the knowledge base more efficient for the kind of problems that it is reasonable to expect. This works as long as the distribution of past examples is roughly the same as for future examples—the same assumption used for PAC-learning in Section 18.6. If the EBL system is carefully engineered, it is possible to obtain very significant improvements on future problems. In a very large Prolog-based natural language system designed for real-time speech-to-speech translation between Swedish and English, Samuelsson and Rayner (1991) report that EBL made the system more than 1200 times faster.

21.3 LEARNING USING RELEVANCE INFORMATION

Our traveller in Brazil seems to be able to make a confident generalization concerning the language spoken by other Brazilians. The inference is sanctioned by her background knowledge, namely, that people in a given country (usually) speak the same language. We can express this in first-order logic as follows:²

$$\forall x, y, n, l \text{ Nationality}(x, n) \wedge \text{Nationality}(y, n) \wedge \text{Language}(x, l) \Rightarrow \text{Language}(y, l) \quad (21.4)$$

(Literal translation: "If x and y have a common nationality n and x speaks language l , then y also speaks it.") It is not difficult to show that, given this sentence and the observation

$$\text{Nationality}(\text{Fernando}, \text{Brazil}) \wedge \text{Language}(\text{Fernando}, \text{Portuguese})$$

the conclusion

$$\forall x \text{ Nationality}(x, \text{Brazil}) \Rightarrow \text{Language}(x, \text{Portuguese})$$

follows logically (see Exercise 21.1).

Sentences such as (21.4) express a strict form of relevance: given nationality, language is fully determined. Put another way: language is a function of nationality. These sentences are called **functional dependencies or determinations**. They occur so commonly in certain kinds of applications (e.g., defining database designs) that a special syntax is used to write them. We adopt the notation used by Davies (1985):

$$\text{Nationality}(x, n) \succ \text{Language}(x, l)$$

As usual, this is simply a syntactic sugararing, but it makes it clear that the determination is really a relationship between the predicates: nationality determines language. The relevant properties determining conductance and density can be expressed similarly:

$$\text{Material}(x, m) \wedge \text{Temperature}(x, t) \succ \text{Conductance}(x, p)$$

$$\text{Material}(x, m) \wedge \text{Temperature}(x, t) \succ \text{Density}(x, d)$$

The corresponding generalizations follow logically from the determinations and observations.

Determining the hypothesis space

Although the determinations sanction general conclusions concerning all Brazilians, or all pieces of copper at a given temperature, they cannot, of course, yield a general predictive theory for *all* nationalities, or for *all* temperatures and materials, from a single example. Their main effect can be seen as limiting the space of hypotheses that the learning agent need consider. In predicting conductance, for example, one has only to consider material and temperature and can ignore mass, ownership, day of the week, the current president, and so on. Hypotheses can certainly include terms that are in turn determined by material and temperature, such as molecular structure, thermal energy, or free-electron density. *Determinations specify a sufficient basis vocabulary*



² We assume for the sake of simplicity that a person speaks only one language. Clearly, the rule also would have to be amended for countries such as Switzerland or India.

from which to construct hypotheses concerning the target predicate. This statement can be proved by showing that a given determination is logically equivalent to a statement that the correct definition of the target predicate is one of the set of all definitions expressible using the predicates in the left-hand side of the determination.

Intuitively, it is clear that a reduction in the hypothesis space size should make it easier to learn the target predicate. Using the basic results of computational learning theory (Section 18.6), we can quantify the possible gains. First, recall that for Boolean functions, $\log(|H|)$ examples are required to converge to a reasonable hypothesis, where $|H|$ is the size of the hypothesis space. If the learner has n Boolean features with which to construct hypotheses, then, in the absence of further restrictions, $H = O(2^{2^n})$, so the number of examples is $O(2^n)$. If the determination contains d predicates in the left-hand side, the learner will require only $O(2^d)$ examples, a reduction of $O(2^{n-d})$. For biased hypothesis spaces, such as a conjunctively biased space, the reduction will be less dramatic but still significant.

Learning and using relevance information

As we stated in the introduction to this chapter, prior knowledge is useful in learning, but it also has to be learned. In order to provide a complete story of relevance-based learning, we must therefore provide a learning algorithm for determinations. The learning algorithm we now present is based on a straightforward attempt to find the simplest determination consistent with the observations. A determination $P \succ Q$ says that if any examples match on P , then they must also match on Q . A determination is therefore consistent with a set of examples if every pair that matches on the predicates on the left-hand side also matches on the target predicate, that is, has the same classification. For example, suppose we have the following examples of conductance measurements on material samples:

Sample	Mass	Temperature	Material	Size	Conductance
S1	12	26	Copper	3	0.59
S1	12	100	Copper	3	0.57
S2	24	26	Copper	6	0.59
S3	12	26	Lead	2	0.05
S3	12	100	Lead	2	0.04
S4	24	26	Lead	4	0.05

The minimal consistent determination is *Material A Temperature \succ Conductance*. There is a nonminimal but consistent determination, namely, *Mass A Size A Temperature \succ Conductance*. This is consistent with the examples because mass and size determine density, and in our data set, we do not have two different materials with the same density. As usual, we would need a larger sample set in order to eliminate a nearly correct hypothesis.

There are several possible algorithms for finding minimal consistent determinations. The most obvious approach is to conduct a search through the space of determinations, checking all determinations with one predicate, two predicates, and so on, until a consistent determination is found. We will assume a simple attribute-based representation, like that used for decision-tree

learning in Chapter 18. A determination d will be represented by the set of attributes on the left-hand side, because the target predicate is assumed fixed. The basic algorithm is outlined in Figure 21.3.

```

function MINIMAL-CONSISTENT-DET( $E, A$ ) returns a determination
  inputs:  $E$ , a set of examples
            $A$ , a set of attributes, of size  $n$ 

  for  $i \leftarrow 0, \dots, n$  do
    for each subset  $A_i$  of  $A$  of size  $i$  do
      if CONSISTENT-DET?( $A_i, E$ ) then return  $A_i$ 
    end
  end

function CONSISTENT-DET?( $A, E$ ) returns a truth-value
  inputs:  $A$ , a set of attributes
            $E$ , a set of examples
  local variables:  $H$ , a hash table

  for each example  $e$  in  $E$  do
    if some example in  $H$  has the same values as  $e$  for the attributes  $A$ 
      but a different classification then return False
    store the class of  $e$  in  $H$ , indexed by the values for attributes  $A$  of the example  $e$ 
  end
  return True

```

Figure 21.3 An algorithm for finding a minimal consistent determination.

The time complexity of this algorithm depends on the size of the smallest consistent determination. Suppose this determination has p attributes out of the n total attributes. Then the algorithm will not find it until searching the subsets of A of size p . There are $\binom{n}{p} = O(n^p)$ such subsets, hence the algorithm is exponential in the size of the minimal determination. It turns out that the problem is NP-complete, so we cannot expect to do better in the general case. In most domains, however, there will be sufficient local structure (see Chapter 15 for a definition of locally structured domains) such that p will be small.

Given an algorithm for learning determinations, a learning agent has a way to construct a minimal hypothesis within which to learn the target predicate. We can combine MINIMAL-CONSISTENT-DET with the DECISION-TREE-LEARNING algorithm, for example, in order to create a relevance-based decision-tree learning algorithm RBDTL:

```

function RBDTL( $E, A, v$ ) returns a decision tree
  return DECISION-TREE-LEARNING( $E, \text{MINIMAL-CONSISTENT-DET}(E, A), v$ )

```

Unlike DECISION-TREE-LEARNING, RBDTL simultaneously learns and uses relevance information in order to minimize its hypothesis space. We expect that RBDTL's learning curve will show some improvement over the learning curve achieved by DECISION-TREE-LEARNING, and this is in fact the case. Figure 21.4 shows the learning performance for the two algorithms on randomly generated data for a function that depends on only 5 of 16 attributes. Obviously, in cases where all the available attributes are relevant, RBDTL will show no advantage.

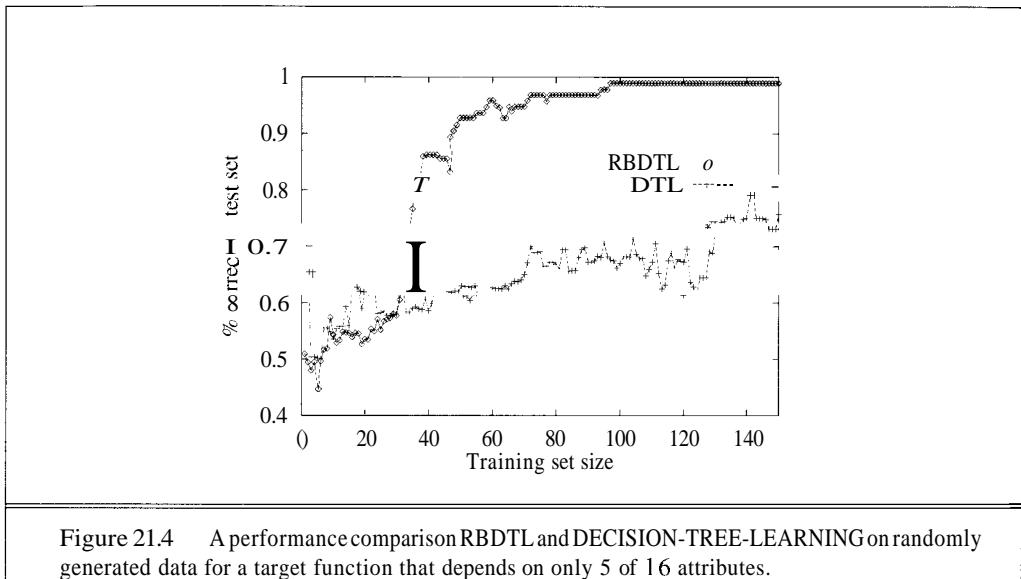


Figure 21.4 A performance comparison RBDTL and DECISION-TREE-LEARNING on randomly generated data for a target function that depends on only 5 of 16 attributes.

DECLARATIVE BIAS

This section has only scratched the surface of the field of **declarative bias**, which aims to understand how prior knowledge can be used to identify the appropriate hypothesis space within which to search for the correct target definition. There are many unanswered questions:

- How can the algorithms be extended to handle noise?
- How can other kinds of prior knowledge be used, besides determinations?
- How can the algorithms be generalized to cover any first-order theory, rather than just an attribute-based representation?

Some of these are addressed in the next section.

21.4 INDUCTIVE LOGIC PROGRAMMING

Inductive logic programming (ILP) is one of the newest subfields in AI. It combines inductive methods with the power of first-order representations, concentrating in particular on the representation of theories as logic programs. Over the last five years, it has become a major part of

the research agenda in machine learning. This has happened for two reasons. First, it offers a rigorous approach to the general KBIL problem mentioned in the introduction. Second, it offers complete algorithms for inducing general, first-order theories from examples, which can therefore learn successfully in domains where attribute-based algorithms fail completely. ILP is a highly technical field, relying on some fairly advanced material from the study of computational logic. We therefore cover only the basic principles of the two major approaches, referring the reader to the literature for more details.³

An example

Recall from Equation (21.3) that the general knowledge-based induction problem is to "solve" the entailment constraint

$$\textit{Background} \wedge \textit{Hypothesis} \wedge \textit{Descriptions} \models \textit{Classifications}$$

for the unknown *Hypothesis*, given the *Background* knowledge and examples described by *Descriptions* and *Classifications*. To illustrate this, we will use the problem of learning family relationships from examples. The observations will consist of an extended family tree, described in terms of *Mother*, *Father*, and *Married* relations, and *Male* and *Female* properties. The target predicates will be such things as *Grandparent*, *BrotherInLaw*, and *Ancestor*. We will use the family tree from Exercise 7.6, shown here in Figure 21.5. The example *Descriptions* include facts such as

<i>Father(Philip, Charles)</i>	<i>Father(Philip, Anne)</i>	...
<i>Mother(Mum, Margaret)</i>	<i>Mother(Mum, Elizabeth)</i>	...
<i>Married(Diana, Charles)</i>	<i>Married(Elizabeth, Philip)</i>	...
<i>Male(Philip)</i>	<i>Male(Charles)</i>	...
<i>Female(Beatrice)</i>	<i>Female(Margaret)</i>	...

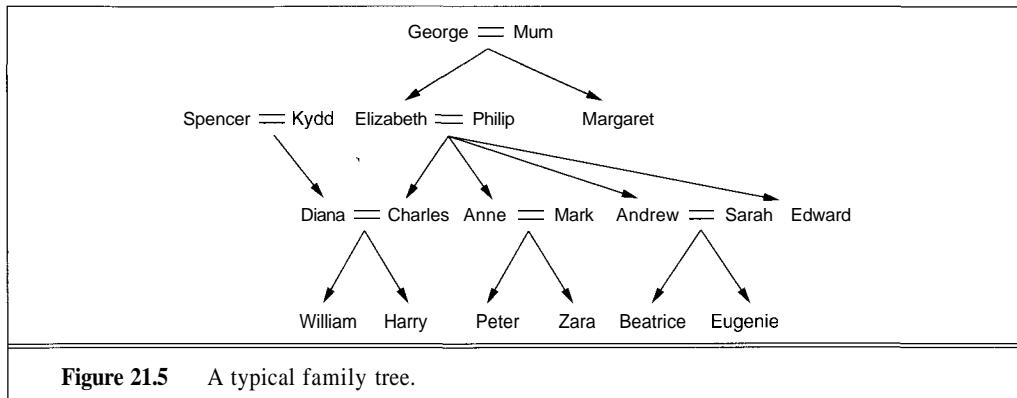
The sentences in *Classifications* depend on the target concept being learned. If *Q* is *Grandparent*, say, then the sentences in *Classifications* might include the following:

$$\begin{aligned} &\textit{Grandparent(Mum,Charles)} \quad \textit{Grandparent(Elizabeth,Beatrice)} \quad \dots \\ &\neg\textit{Grandparent(Mum,Harry)} \quad \neg\textit{Grandparent(Spencer,Peter)} \end{aligned}$$

The object of an inductive learning program is to come up with a set of sentences for the *Hypothesis* such that the entailment constraint is satisfied. Suppose, for the moment, that the agent has no background knowledge: *Background* is empty. Then one possible solution for *Hypothesis* is the following:

$$\begin{aligned} \textit{Grandparent}(x, >*) &\Leftrightarrow [\exists z \textit{ Mother}(x,z) \wedge \textit{Mother}(z,y)] \\ &\vee [\exists z \textit{ Mother}(x,z) \wedge \textit{Father}(z,y)] \\ &\vee [\exists z \textit{ Father}(x,z) \wedge \textit{Mother}(z,y)] \\ &\vee [\exists z \textit{ Father}(x,z) \wedge \textit{Father}(z,y)] \end{aligned}$$

³ We suggest that it might be appropriate at this point for the reader to refer back to Chapter 9 for some of the underlying concepts, including Horn clauses, conjunctive normal form, unification, and resolution.



Notice that an attribute-based learning algorithm such as DECISION-TREE-LEARNING will get nowhere in solving this problem. In order to express *Grandparent* as an attribute (i.e., a unary predicate), we would need to *make pairs of people into objects*:

$\text{Grandparent}(\langle \text{Mum}, \text{Charles} \rangle) \dots$

Then we get stuck in trying to represent the example descriptions. The only possible attributes are horrible things such as

$\text{FirstElementIsMotherOfElizabeth}(\text{Mum}, \langle \text{Charles} \rangle)$

The definition of *Grandparent* in terms of these attributes simply becomes a large disjunction of specific cases that does not generalize to new examples at all. *Attribute-based learning algorithms are incapable of learning relational predicates*. Thus, one of the principal advantages of ILP algorithms is their applicability to a much wider range of problems.

The reader will certainly have noticed that a little bit of background knowledge would help in the representation of the *Grandparent* definition. For example, if *Background* included the sentence

$$\text{Parent}(x, y) \Leftrightarrow [\text{Mother}(x, y) \vee \text{Father}(x, y)]$$

then the definition of *Grandparent* would be reduced to

$$\text{Grandparent}(x, y) \Leftrightarrow [\exists z \text{ Parent}(x, z) \wedge \text{Parent}(z, y)]$$

This shows how background knowledge can dramatically reduce the size of hypothesis required to explain the observations.

It is also possible for ILP algorithms to *create* new predicates in order to facilitate the expression of explanatory hypotheses. Given the example data shown earlier, it is entirely reasonable to propose an additional predicate, which we would call “*Parent*,” in order to simplify the definitions of the target predicates. Algorithms that can generate new predicates are called **constructive induction** algorithms. Clearly, constructive induction is a necessary part of the picture of cumulative learning sketched in the introduction. It has been one of the hardest problems in machine learning, but some ILP techniques provide effective mechanisms for achieving it.

In the rest of this chapter, we will study the two principal approaches to ILP. The first uses techniques based on inverting a resolution proof, and the second uses a generalization of decision-tree methods.

INVERSE
RESOLUTION

Inverse resolution

Inverse resolution is based on the observation that if the example *Classifications* follow from *Background A Hypothesis A Descriptions*, then one must be able to prove this fact by resolution (because resolution is complete). If we can "run the proof backwards," then we can find a *Hypothesis* such that the proof goes through. The key, then, is to find a way to invert the resolution process so that we can run the proof backwards.

Generating inverse proofs

The backward proof process consists of individual backward steps. An ordinary resolution step takes two clauses C_1 and C_2 and resolves them to produce the **resolvent** C . An inverse resolution step takes a resolvent C and produces two clauses C_1 and C_2 , such that C is the result of resolving C_1 and C_2 ; or it takes C and C_1 and produces a suitable C_2 .

The early steps in an inverse resolution process are shown in Figure 21.6, where we focus on the positive example *Grandparent(George, Anne)*. The process begins at the end of the proof, that is, at the contradiction, and works backwards. The negated goal clause is $\neg\text{Grandparent}(\text{George}, \text{Anne})$, which is $\text{Grandparent}(\text{George}, \text{Anne}) \Rightarrow \text{False}$ in implicative normal form. The first inverse step takes this and the contradictory clause $\text{True} \Rightarrow \text{False}$, and generates *Grandparent(George, Anne)*. The next step takes this clause and the known clause *Parent(Elizabeth, Anne)*, and generates the clause

$$\text{Parent}(\text{Elizabeth}, y) \Rightarrow \text{Grandparent}(\text{George}, y)$$

With one further step, the inverse resolution process will generate the correct hypothesis.

Clearly, inverse resolution involves a search. Each inverse resolution step is nondeterministic, because for any C and C_1 , there can be several or even an infinite number of clauses C_2 that satisfy the requirement that when resolved with C_1 it generates C . For example, in-

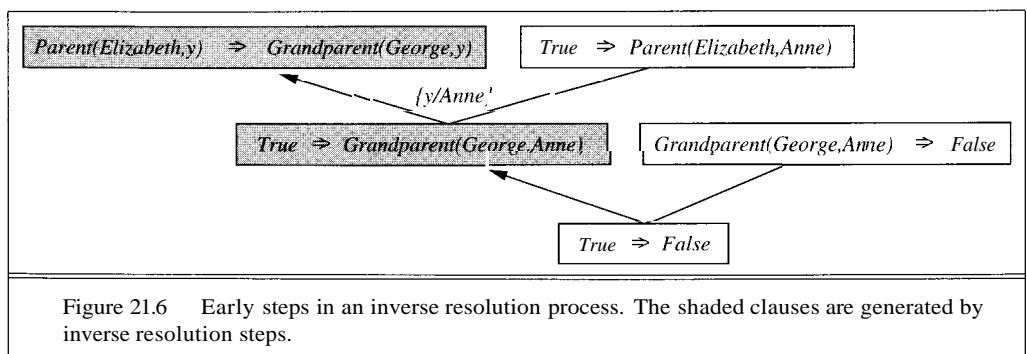


Figure 21.6 Early steps in an inverse resolution process. The shaded clauses are generated by inverse resolution steps.

stead of $\text{Parent}(\text{Elizabeth}, y) \Rightarrow \text{Grandparent}(\text{George}, y)$, the inverse resolution step might have generated the following sentences:

$$\text{Parent}(\text{Elizabeth}, \text{Anne}) \Rightarrow \text{Grandparent}(\text{George}, \text{Anne})$$

$$\text{Parent}(z, \text{Anne}) \Rightarrow \text{Grandparent}(\text{George}, \text{Anne})$$

$$\text{Parent}(z, y) \Rightarrow \text{Grandparent}(\text{George}, y)$$

⋮

(See Exercises 21.4 and 21.5.) Furthermore, the clauses C_1 (and perhaps also C_2) that participate in each step can be chosen from the *Background* knowledge, from the example *Descriptions*, from the negated *Classifications*, or from hypothesized clauses that have already been generated in the inverse resolution tree.

An exhaustive search process for inverse resolution would be extremely inefficient. ILP systems use a number of restrictions to make the process more manageable, including the elimination of function symbols, generating only the most specific hypotheses possible, and the use of Horn clauses. One can also consider inverting the *restricted* resolution strategies that were introduced in Chapter 9. With a restricted but complete strategy, such as linear resolution, the inverse resolution process will be more efficient because certain clauses will be ruled out as candidates for C_1 and C_2 . Other useful constraints include the fact that all the hypothesized clauses must be consistent with each other, and that each hypothesized clause must agree with the observations. This last criterion would rule out the clause $\text{Parent}(z, y) \Rightarrow \text{Grandparent}(\text{George}, y)$ listed before.

Discovering new predicates and new knowledge

An inverse resolution procedure that inverts a complete resolution strategy is, in principle, a complete algorithm for learning first-order theories. That is, if some unknown *Hypothesis* generates a set of examples, then an inverse resolution procedure can generate *Hypothesis* from the examples. This observation suggests an interesting possibility. Suppose, for example, that the available examples include a variety of trajectories of falling bodies. Would an inverse resolution program be theoretically capable of inferring the law of gravity? The answer is clearly yes, because the law of gravity allows one to explain the examples, given suitable background mathematics. Similarly, one can imagine that electromagnetism, quantum mechanics, and the theory of relativity are also within the scope of ILP programs. However, such imaginings are on a par with the proverbial monkey with a typewriter, at least until we find ways to overcome the very large branching factors and the lack of structure in the search space that characterize current systems.

One thing that inverse resolution systems *will* do for you is invent new predicates. This ability is often seen as somewhat magical, because computers are often thought of as "merely working with what they are given." In fact, new predicates fall directly out of the inverse resolution step. The simplest case arises when hypothesizing two new clauses C_1 and C_2 , given a clause C . The resolution of C_1 and C_2 eliminates a literal that the two clauses share, hence it is quite possible that the eliminated literal contained a predicate that does not appear in C . Thus, when working backwards, one possibility is to generate a new predicate from which to reconstruct the missing literal.

Figure 21.7 shows an example in which the new predicate P is generated in the process of learning a definition for *Ancestor*. Once generated, P can be used in later inverse resolution steps. For example, a later step might hypothesize that $Mother(x,y) \Rightarrow P(x,y)$. Thus, the new predicate P has its meaning constrained by the generation of hypotheses that involve it. Another example might lead to the constraint $Father(x,y) \Rightarrow P(x,y)$. In other words, the predicate P is what we usually think of as the *Parent* relationship. As we mentioned earlier, the invention of new predicates can significantly reduce the size of the definition of the goal predicate. Hence, by including the ability to invent new predicates, inverse resolution systems can often solve learning problems that are infeasible with other techniques.

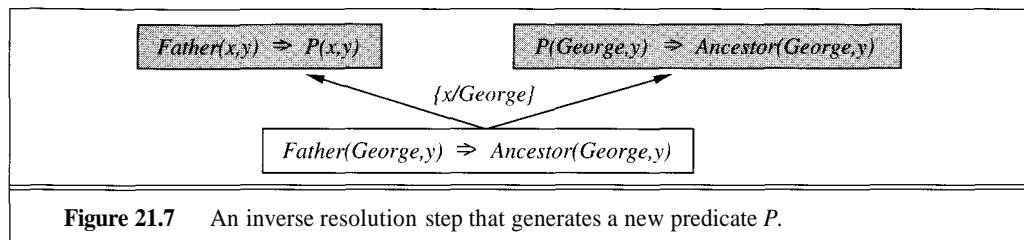


Figure 21.7 An inverse resolution step that generates a new predicate P .

Some of the deepest revolutions in science come from the invention of new predicates and functions—for example, Galileo's invention of acceleration or Joule's invention of thermal energy. Once these terms are available, the discovery of new laws becomes (relatively) easy. The difficult part lies in realizing that some new entity, with a specific relationship to existing entities, will allow an entire body of observations to be explained with a much simpler and more elegant theory than previously existed.

As yet, ILP systems have not been applied to such difficult tasks. It does appear, however, that the ability to use background knowledge provides significant advantages. In several applications, ILP techniques have outperformed knowledge-free methods. For example, in molecular biology, it is useful to have background knowledge about typical molecular bonding patterns, valences of atoms, bond strengths, and so on. Using such knowledge, Stephen Muggleton's GOLEM system has been able to generate high-quality predictions of both protein structure from sequence information (Muggleton *et al.*, 1992) and the therapeutic efficacy of various drugs based on their molecular structures (King *et al.*, 1992). These results, like Meta-DENDRAL's, were considered sufficiently interesting in their own right to be published in leading scientific journals. The differences between GOLEM'S and Meta-DENDRAL's performance are that (1) the new domains are much more difficult, and (2) GOLEM is a completely general-purpose program that is able to make use of background knowledge about any domain whatsoever.

Top-down learning methods

The second approach to ILP is essentially a generalization of the techniques of decision-tree learning to the first-order case. Rather than starting from the observations and working backwards, we start with a very general rule and gradually specialize it so that it fits the data. This is essentially what happens in decision-tree learning, where a decision tree is gradually grown until

it is consistent with the observations. In the first-order case, we use first-order literals instead of attributes, and the hypothesis is a set of clauses instead of a decision tree. This section describes FOIL (Quinlan, 1990), one of the first programs to use this approach.

Suppose we are trying to learn a definition of the $\text{Grandfather}(x, y)$ predicate, using the same family data as before. As with decision-tree learning, we can divide the examples into positive and negative examples. Positive examples are

$\langle \text{George}, \text{Anne} \rangle, \langle \text{Philip}, \text{Peter} \rangle, \langle \text{Spencer}, \text{Harry} \rangle, \dots$

and negative examples are

$\langle \text{George}, \text{Elizabeth} \rangle, \langle \text{Harry}, \text{Zara} \rangle, \langle \text{Charles}, \text{Philip} \rangle, \dots$

Notice that each example is a *pair* of objects, because Grandfather is a binary predicate. In all, there are 12 positive examples in the family tree, and 388 negative examples (all the other pairs of people).

FOIL constructs a set of Horn clauses with $\text{Grandfather}(x, y)$ as the head, such that the 12 positive examples are classified as instances of the $\text{Grandfather}(x, y)$ relationship, whereas the other 388 examples are ruled out because no clause succeeds with those bindings for x and y . We begin with a clause with an empty body:

$\Rightarrow \text{Grandfather}(xy)$

This classifies every example as positive, so it needs to be specialized. This is done by adding literals one at a time to the left-hand side. Here are three potential additions:

$\text{Father}(x, y) \Rightarrow \text{Grandfather}(x, y)$

$\text{Parent}(x, z) \Rightarrow \text{Grandfather}(x, y)$

$\text{Father}(x, z) \Rightarrow \text{Grandfather}(x, y)$

(Notice that we are assuming that a clause defining Parent is already present as part of the background knowledge.) The first of these three incorrectly classifies all of the 12 positive examples as negative, and therefore can be ruled out. The second and third agree with all of the positive examples, but the second is incorrect on a larger fraction of the negative examples—twice as many, in fact, because it allows mothers as well as fathers. Hence, we prefer the third clause.

Now we need to specialize this clause further, to rule out the cases in which x is the father of some z but z is not a parent of y . Adding the single literal $\text{Parent}(z, y)$ gives the clause

$\text{Father}(x, z) \wedge \text{Parent}(z, y) \Rightarrow \text{Grandfather}(xy)$

which correctly classifies all the examples. FOIL will find and choose this literal, thereby solving the learning task.

The preceding example is a very simple illustration of how FOIL operates. A sketch of the complete algorithm is shown in Figure 21.8. Essentially, the algorithm repeatedly constructs a clause, literal by literal, until it agrees with some subset of the positive examples and none of the negative examples. Then the positive examples covered by the clause are removed from the training set, and the process continues until no positive examples remain. The two main components to be explained are NEW-LITERALS, which constructs all possible new literals to add to the clause, and CHOOSE-LITERAL, which selects a literal to add.

```

function FOIL(examples,target)returns a set of Horn clauses
  inputs: examples, set of examples
           target, a literal for the goal predicate
  local variables: clauses, set of clauses, initially empty

  while examples contains positive examples do
    clause — NEW-CLAUSE(examples)
    remove examples covered by clause from examples
    clauses — {clause|clauses}
  return clauses

function NEW-CLAUSE(example,target) returns a Horn clause
  local variables: clause, a clause with target as head and an empty body
                  l, a literal to be added to the clause
                  extended-examples, a set of examples with values for new variables

  extended-examples — examples
  while extended-examples contains negative examples do
    l ← CHOOSE-LITERAL(NEW-LITERALS(clause),extended-examples)
    append l to the body of clause
    extended-examples ← set of examples created by applying EXTEND-EXAMPLE
      to each example in extended-examples
  return clause

function EXTEND-EXAMPLE(example,literal) returns
  if example satisfies literal
    then return the set of examples created by extending example with
      each possible constant value for each new variable in literal
  else return the empty set

```

Figure 21.8 Sketch of the FOIL algorithm for learning sets of first-order Horn clauses from examples. NEW-LITERAL and CHOOSE-EITERAL are explained in the text.

NEW-LITERALS takes a clause and constructs all possible "useful" literals that could be added to the clause. Let us use as an example the clause

$$\text{Father}(x, z) \Rightarrow \text{Grandfather}(x, y)$$

There are three kinds of literals that can be added.

1. Literals using predicates: the literal can be negated or unnegated, any existing predicate (including the goal predicate) can be used, and the arguments must all be variables. Any variables can be used for any argument of the predicate, with one restriction: each literal must include *at least one* variable from an earlier literal or from the head of the clause. Literals such as *Mother(z, u)*, *Married(z,z)*, $\neg\text{Male}(y)$, and *Grandfather(vx)* are allowed, whereas *Married(u, v)* is not. Notice that the use of the predicate from the head of the clause allows FOIL to learn *recursive* definitions.

2. Equality and inequality literals: these relate variables already appearing in the clause. For example, we might add $z \neq x$. These literals can also include user-specified constants. In the family domain, there will not usually be any such "special" constants, whereas in learning arithmetic, we might use 0 and 1, and in list functions, the empty list [].
3. Arithmetic comparisons: when dealing with functions of continuous variables, literals such as $x > y$ and $y < z$ can be added. As in decision-tree learning, one can also use constant threshold values that are chosen to maximize the discriminatory power of the test.

All this adds up to a very large branching factor in the search space (see Exercise 21.6). Implementations of FOIL may also use type information to restrict the hypothesis space. For example, if the domain included numbers as well as people, type restrictions would prevent NEW-LITERALS from generating literals such as $\text{Parent}(x, n)$, where x is a person and n is a number.

CHOOSE-LITERAL uses a heuristic somewhat similar to information gain (see page 541) to decide which literal to add. The exact details are not so important here, particularly as a number of different variations are currently being tried out. One interesting additional feature of FOIL is the use of Ockham's razor to eliminate some hypotheses. If a clause becomes longer (according to some metric) than the total length of the positive examples that the clause explains, that clause is not considered as a potential hypothesis. This technique provides a way to avoid overcomplex clauses that fit noise in the data. For an explanation of the connection between noise and clause length, see Section 19.6.

FOIL and its relatives have been used to learn a wide variety of definitions. One of the most impressive demonstrations (Quinlan and Cameron-Jones, 1993) involved solving a long sequences of exercises on list-processing functions from Bratko's (1986) Prolog textbook. In each case, the program was able to learn a correct definition of the function from a small set of examples, using the previously learned functions as background knowledge.

21.5 SUMMARY

This chapter has investigated various ways in which prior knowledge can help an agent to learn from new experiences.

- The use of prior knowledge in learning leads to a picture of **cumulative learning**, in which learning agents improve their learning ability as they acquire more knowledge.
- Prior knowledge helps learning by eliminating otherwise consistent hypotheses and by "filling in" the explanation of examples, thereby allowing for shorter hypotheses. These contributions improve both the sample complexity and computation complexity of learning.
- Understanding the different logical roles played by prior knowledge, as expressed by **entailment constraints**, helps to define a variety of learning techniques.
- **Explanation-based learning** (EBL) extracts general rules from single examples by *explaining* the examples and generalizing the explanation. It provides a deductive method to turn first-principles knowledge into useful, efficient, special-purpose expertise.

- **Relevance-based learning** (RBL) uses prior knowledge in the form of determinations to identify the relevant attributes, thereby generating a reduced hypothesis space and speeding up learning. RBL also allows deductive generalizations from single examples.
- **Knowledge-based inductive learning** (KBIL) finds inductive hypotheses that explain sets of observations with the help of background knowledge.
- **Inductive logic programming** (ILP) techniques perform KBIL using knowledge expressed in first-order logic. ILP methods can learn relational knowledge that is not expressible in attribute-based systems.
- ILP methods naturally generate new predicates with which concise new theories can be expressed, and show promise as general-purpose scientific theory formation systems.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

GRUE

The use of prior knowledge in learning from experience has had a surprisingly brief period of intensive study. *Fact, Fiction, and Forecast*, by the philosopher Nelson Goodman (1954), refuted the earlier supposition that induction was simply a matter of seeing enough examples of some universally quantified proposition and then adopting it as a hypothesis. Consider, for example, the hypothesis "All emeralds are grue," where **grue** means "green if observed before time t , but blue if observed thereafter." At any time up to t , we might have observed millions of instances confirming the rule that emeralds are grue, and no disconfirming instances, and yet we are unwilling to adopt the rule. This can only be explained by appeal to the role of relevant prior knowledge in the induction process. Goodman proposes a variety of different kinds of prior knowledge that might be useful, including a version of determinations called **overhypotheses**. Unfortunately, Goodman's work was never taken up in early studies of machine learning.

EBL had its roots in the techniques used by the STRIPS planner (Fikes *et al.*, 1972). When a plan was constructed, a generalized version of it was saved in a plan library and used in later planning as a **macro-operator**. Similar ideas appeared in Anderson's ACT* architecture, under the heading **of knowledge compilation** (Anderson, 1983); and in the SOAR architecture as **chunking** (Laird *et al.*, 1986). **Schema acquisition** (DeJong, 1981), **analytical generalization** (Mitchell, 1982), and **constraint-based generalization** (Minton, 1984) were immediate precursors of the rapid growth of interest in EBL stimulated by the publication of (Mitchell *et al.*, 1986; DeJong and Mooney, 1986). Hirsh (1987) introduced the EBL algorithm described in the text, showing how it could be incorporated directly into a logic programming system. Van Harmelen and Bundy (1988) explain EBL as a variant of the **partial evaluation** method used in program analysis systems (Jones *et al.*, 1993).

More recently, rigorous analysis and experimental work has led to a better understanding of the potential costs and benefits of EBL in terms of problem-solving speed. Minton (1988) showed that without extensive extra work, EBL could easily slow down a program significantly. Tambe *et al.* (1990) found a similar problem with chunking, and proposed a reduction in the expressive power of the rule language in order to minimize the cost of matching rules against working memory. This work bears strong parallels with recent results on the complexity of

inference in restricted versions of first-order logic (see Chapter 10). Formal probabilistic analysis of the expected payoff of EBL can be found in (Greiner, 1989; Subramanian and Feldman, 1990). An excellent survey appears in (Dietterich, 1990).

Instead of using examples as foci for generalization, one can use them directly to solve new problems in a process known as **analogical reasoning**. This form of reasoning ranges from a form of plausible reasoning based on degree of similarity (Gentner, 1983), through a form of deductive inference based on determinations (Davies and Russell, 1987) but requiring the participation of the example, to a form of "lazy" EBL that tailors the direction of generalization of the old example to fit the needs of the new problem. This latter form of analogy reasoning is found most commonly in **case-based reasoning** (Kolodner, 1993) and **derivational analogy** (Veloso and Carbonell, 1993).

Relevance information in the form of functional dependencies was first developed in the database community, where it is used to structure large sets of attributes into manageable subsets. Functional dependencies were used for analogical reasoning by Carbonell and Collins (1973), and given a more logical flavor by Bobrow and Raphael (1974). Dependencies were independently rediscovered, and given a full logical analysis, by Davies (1985) and Russell (1986a), for the problem of analogical inference (see also (Davies and Russell, 1987)). They were used for declarative bias by Russell and Grosof (1987). The equivalence of determinations to a restricted-vocabulary hypothesis space was proved in (Russell, 1988). Learning algorithms for determinations, and the improved performance obtained by RBDTL, were first shown in the FOCUS algorithm in (Almuallim and Dietterich, 1991). Tadepalli (1993) describes an ingenious algorithm for learning with determinations that shows large improvements in learning speed.

The study of methods for learning first-order logical sentences began with the remarkable Ph.D. thesis by Gordon Plotkin (1971) at Edinburgh. Although Plotkin developed many of the theorems and methods that are in current use in ILP, he was discouraged by some undecidability results for certain subproblems in induction. MIS (Shapiro, 1981) reintroduced the problem of learning logic programs, but was mainly seen as a contribution to the theory of automated debugging. The field was reinvigorated by Muggleton and Buntine (1988), whose CIGOL program incorporated a slightly incomplete version of inverse resolution and was capable of generating new predicates.⁴ More recent systems include GOLEM (Muggleton and Cao, 1990), ITOU (Rouveiro and Puget, 1989) and CUNT (De Raedt, 1992). A second thread of ILP research began with Quinlan's FOIL system, described in this chapter (Quinlan, 1990). A formal analysis of ILP methods appears in (Muggleton, 1991), and a large collection of papers in (Muggleton, 1992).

Early complexity results by Haussler (1989) suggested that learning first-order sentences was hopelessly complex. However, with better understanding of the importance of various kinds of syntactic restrictions on clauses, positive results have been obtained even for clauses with recursion (Dzeroski *et al.*, 1992). A recent paper by Kietz and Dzeroski (1994) provides an excellent survey of complexity results in ILP.

Although ILP now seems to be the dominant approach to constructive induction, it has not been the only approach taken. So-called **discovery systems** aim to model the process of scientific discovery of new concepts, usually by a direct search in the space of concept definitions. Doug Lenat's AM (Automated Mathematician) (Davis and Lenat, 1982) used discovery heuristics

⁴ The inverse resolution method also appears in (Russell, 1986b), where a complete algorithm is mentioned in a footnote.

expressed as expert system rules to guide its search for concepts and conjectures in elementary number theory. In sharp contrast with most systems designed for mathematical reasoning, AM lacked a concept of proof and could only make conjectures. It rediscovered Goldbach's Conjecture and the Unique Prime Factorization Theorem. AM's architecture was generalized in the EURISKO system (Lenat, 1983) by adding a mechanism capable of rewriting the system's own discovery heuristics. EURISKO was applied in a number of areas other than mathematical discovery, although with less success than AM. The methodology of AM and EURISKO has been controversial (Ritchie and Hanna, 1984; Lenat and Brown, 1984).

Another class of discovery systems aims to operate with real scientific data to find new laws. The systems DALTON, GLAUBER, and STAHL (Langley *et al.*, 1987) are rule-based systems that look for quantitative relationships in experimental data from physical systems; in each case, the system has been able to recapitulate a well-known discovery from the history of science. AUTOCLASS (Cheeseman *et al.*, 1988) takes a more theoretically grounded approach to concept discovery: it uses Bayesian probabilistic reasoning to partition given data into the "most likely" collection of classes. AUTOCLASS has been applied to a number of real-world scientific classification tasks, including the discovery of new types of stars from spectral data and the analysis of protein structure (Hunter and States, 1992).

EXERCISES

21.1 Show, by translating into conjunctive normal form and applying resolution, that the conclusion drawn on page 633 concerning Brazilians is sound.

21.2 For each of the following determinations, write down the logical representation and explain why the determination is true (if it is):

- a. Zip code determines the state (U.S.).
- b. Design and denomination determine the mass of a coin.
- c. For a given program, input determines output.
- d. Climate, food intake, exercise, and metabolism determine weight gain/loss.
- e. Baldness is determined by the baldness (or lack thereof) of one's maternal grandfather.

21.3 Would a probabilistic version of determinations be useful? Suggest a definition.

21.4 Fill in the missing values for the clauses $C\backslash$ and/or C_2 in the following sets of clauses, given that C is the resolvent of $C\backslash$ and C_2 .

- a. $C = \text{True} \Rightarrow P(A,B), C_1 = P(x,y) \Rightarrow Q(x,y), C_2 = ??.$
- b. $C = \text{True} \Rightarrow P(A,B), C_1 = II, C_2 = ??.$
- c. $C = P(x,y) \Rightarrow P(x,f(y))C_1 = II, C_2 = II.$

If there is more than one possible solution, provide one example of each different kind.



21.5 Suppose one writes a logic program that carries out a resolution inference step. That is, let $\text{Resolve}(c_1, c_2, c)$ succeed if c is the result of resolving c_1 and c_2 . Normally, Resolve would be used as part of a theorem prover by calling it with c_1 and c_2 instantiated to particular clauses, thereby generating the resolvent c . Now suppose instead that we call it with c instantiated and c_1 and c_2 uninstantiated. Will this succeed in generating the appropriate results of an inverse resolution step? Would you need any special modifications to the logic programming system for this to work?

21.6 Suppose that FOIL is considering adding a literal to a clause using a binary predicate P , and that previous literals (including the head of the clause) contain five different variables.

- a. How many functionally different literals can be generated? Notice that two literals are functionally identical if they differ only in the names of the *new* variables that they contain.
- b. Can you find a general formula for the number of different literals with a predicate of arity r when there are n variables previously used?
- c. Why does FOIL not allow literals that contain no previously used variables?

21.7 Using the data from the family tree in Figure 21.5, or a subset thereof, apply the FOIL algorithm to learn a definition for the *Ancestor* predicate.

Part VII

COMMUNICATING, PERCEIVING, AND ACTING

So far we have been concerned with what happens inside an agent—from the time it receives a percept to the time it decides on an action. In this part, we concentrate on the interface between the agent and the environment. On one end, we have *perception*: vision, hearing, touch, and possibly other senses. On the other end, we have *action*: the movement of a robot arm, for example.

Also covered in this part is *communication*. A group of agents can be more successful—individually and collectively—if they communicate their beliefs and goals to each other. We look most closely at human language and how it can be used as a communication tool.

22

AGENTS THAT COMMUNICATE

In which we see why agents might want to exchange information-carrying messages with each other, and how they can do so.

It is dusk in the savanna woodlands of Amboseli National Park near the base of Kilimanjaro. A group of vervet monkeys are foraging for food. One vervet lets out a loud barking call and the group responds by scrambling for the trees, neatly avoiding the leopard that the first vervet had seen hiding in the bush. The vervet has successfully communicated with the group.

Communication is such a widespread phenomenon that it is hard to pin down an exact definition. In general, *communication is the intentional exchange of information brought about by the production and perception of signs drawn from a shared system of conventional signs*. Most animals employ a fixed set of signs to represent messages that are important to their survival: food here, predator nearby, approach, withdraw, let's mate. The vervet is unusual in having a variety of calls for different predators: a loud bark for leopards, a short cough for eagles, and a chutter for snakes. They use one kind of grunt in exchanges with dominant members of their own social group, another kind with subordinate members, and yet another with vervets in other social groups.

The leopard alarm (the loud bark) is a conventional sign that both alerts others that there is danger in the bush nearby, and signals the action of escaping to the trees. But consider a vervet that is too far away to hear the call, but can see the others heading for the trees. He too may climb the nearest tree, but he has not participated in communication because he did not perceive any intentionally conveyed signs, but rather used his general powers of perception and reasoning.

Humans use a limited number of conventional signs (smiling, shaking hands) to communicate in much the same way as other animals. Humans have also developed a complex, structured system of signs known as **language** that enables them to communicate most of what they know about the world. Although chimpanzees, dolphins, and other mammals have shown vocabularies of hundreds of signs and some aptitude for stringing them together, humans are the only species that can reliably communicate an unbounded number of qualitatively different messages.¹

¹ The bee's tail-wagging dance specifies the distance and angle from the sun at which food can be found, so in one sense the bee can convey an infinite number of messages, but we do not count this as unbounded variation.

Of course, there are other attributes that are uniquely human: no other species wears clothes, creates representational art, or watches four hours of television a day. But when Turing proposed his test (see Section 1.1), he based it on language because language is intimately tied to thinking in a way that, say, clothing is not. In this chapter, we will explain how a communicating agent works and present a simplified version of English that is sufficient to illustrate the workings of the agent's major components.

22.1 COMMUNICATION AS ACTION

SPEECH ACT

One of the actions that is available to an agent is to produce language. This is called a **speech act**. "Speech" is used in the same sense as in "free speech," not "talking," so typing, skywriting, and using sign language all count as speech acts. English has no neutral word for an agent that produces language, either by speaking or writing or anything else. We will use **speaker**, **hearer**, and **utterance** as generic terms referring to any mode of communication. We will also use the term **words** to refer to any kind of conventional communicative sign.

Why would an agent bother to perform a speech act when it could be doing a "regular" action? Imagine a group of agents are exploring the wumpus world together. The group gains an advantage (collectively and individually) by being able to do the following:

INDIRECT SPEECH ACT

- **Inform** each other about the part of the world each has explored, so that each agent has less exploring to do. This is done by making statements: *There's a breeze here in 3 4.*
- **Query** other agents about particular aspects of the world. This is typically done by asking questions: *Have you smelled the wumpus anywhere?*
- **Answer** questions. This is a kind of informing. *Yes, I smelled the wumpus in 2 5.*
- **Request or command** other agents to perform actions: *Please help me carry the gold.* It can be seen as impolite to make a direct request, so often an **indirect speech act** (a request in the form of a statement or question) is used instead: */ could use some help carrying this* or *Could you help me carry this?*
- **Promise** to do things or **offer** deals: */// shoot the wumpus if you let me share the gold.*
- **Acknowledge** requests and offers: *OK.*
- **Share** feelings and experiences with each other: *You know, old chap, when I get in a spot like that, I usually go back to the start and head out in another direction, or Man, that wumpus sure needs some deodorant!*

These examples show that speech acts can be quite useful and versatile. Some kinds of speech acts (informing, answering, acknowledging, sharing) have the intended effect of transferring information to the hearer. Others (requesting, commanding, querying, leopard alarm) have the intended effect of making the hearer take some action. A dual purpose of communication is to establish trust and build social ties (just as primates groom each other both to remove fleas and to build relationships). This helps to explain what is communicated in exchanges such as "Hello, how are you? Fine, how are you? Not bad."

THE EVOLUTION OF LANGUAGE

In 1866, the Société de Linguistique de Paris passed a by-law banning all debate on the origin of language. Outside the halls of that body, the debate goes on. One proposal (Chomsky, 1980; Fodor, 1983) is that there is a language module that exists only in the brain of humans. There is now considerable evidence (Dingwell, 1988) that language use is made possible by a large repertoire of different skills that did not develop in isolation of general cognitive capabilities, and that many of the precursors of human language use can be seen in the other primates and in the fossil record.

At least seven researchers claim to have taught primates over 100 words. Koko the gorilla is the vocabulary queen with over a thousand. Although some argue that the researchers become too attached to the animals and attribute abilities to them that are not really there, it seems safe to say that primates can learn words spontaneously (from human trainers or from each other) and can use them to inform others of what they know and what they want. They can produce and understand sentences involving abstract concepts in a limited way, such as referring to objects that are not present. For example, chimpanzees can correctly respond to "get the ball that is outside" or "bring the red ball over to the blue ball." They can even tell lies: a vervet who is losing a fight may give the leopard alarm, causing the fight to be called off while everyone heads for the trees. Some have claimed that such behavior is evidence that animals form models of the world, including models of how other agents act. Unfortunately, present psychological methodologies do not allow us to distinguish between behaviors that are mediated by internal models and those that are merely stimulus-response patterns. Also, although primates clearly learn some rules for ordering words, there appear to be limitations in the way they use syntax to produce unbounded sequences. Despite some impressive accomplishments, no primate has duplicated the explosion of language use that all normal human children accomplish by age four.

Language and thought reinforce each other, but it is not known if humans evolved to use language well because they are smart, or if they are smart because they use language well. One theory (Jerison, 1991) is that human language stems primarily from a need for better cognitive maps of the territory. Canines and other social carnivores rely heavily on scent marking and their olfactory system to decide both where they are and what other animals have been there—just as the wumpus world agents use the presence of a smell or breeze to help map out that world. But the early hominids (monkeys and apes of 30 million years ago) did not have a well enough developed olfactory system to map out the world this way, so they substituted vocal sounds for scent marking. Thus, the rest of this chapter, in Jerison's view, is devoted to the way we humans compensate for our inadequate noses.



The hard part for an agent is to decide *when* a speech act of some kind is called for, and to decide *which* speech act, out of all the possibilities, is the right one. At one level, this is just the familiar **planning** problem—an agent has a set of possible actions to choose from, and must somehow try to choose actions that achieve the goal of communicating some information to another agent. *All the difficulties that make planning hard (see Chapter 12) apply to planning speech acts.* It would be impossibly complex to plan English conversations at the level of individual movements of the mouth and tongue, so we need to plan with several levels of hierarchical abstraction—words, phrases and sentences, at least. Another problem is nondeterminism. Whereas most actions in the wumpus world are deterministic, speech acts are not. Consider the action *Speak("TurnRight!")*. If another agent perceives the words, and if the agent interprets it as a command to turn right, then that agent may do so. Or then again, the agent may ignore the command and choose another action. The nondeterminism means that we will need conditional plans. Instead of planning a conversation from beginning to end, the best we can do is construct a general plan or policy for the conversation, generate the first sentence, stop to perceive the reply, and react to it.

UNDERSTANDING

The problem of understanding speech acts is much like other **understanding** problems, such as image understanding or medical diagnosis. We are given a set of ambiguous inputs, and from them we have to work backwards to decide what state of the world could have created the inputs. Part of the speech act understanding problem is specific to language. We need to know something about the syntax and semantics of a language to determine why another agent performed a given speech act. The understanding problem also includes the more general problem of **plan recognition**. If we observe an agent turning and moving toward the gold, we can understand the actions by forming a model of the agent's beliefs that says the agent has the goal of getting the gold. A similar kind of mental model building is required to understand the agent who turns toward the gold and says, "I'm going to grab it." Even though there may be other objects nearby, it is fairly clear that "it" refers to the gold.

PLAN RECOGNITION

Part of the understanding problem can be handled by logical reasoning. We will see that logical implications are a good way of describing the ways that words and phrases combine to form larger phrases. Another part of the understanding problem can only be handled by uncertain reasoning techniques. Usually, there will be several states of the world that could all lead to the same speech act, so the understander has to decide which one is more probable.

Now that we have seen how communication fits into our general agent design, we can turn our attention to language itself. As we focus more and more on the way language *actually is*, rather than on the general properties of communication methods, we will find ourselves moving into the realm of natural science—that is, science that works by finding out things about the real world rather than about programs or other artifacts. Natural language understanding is one of the few areas of AI that has this property.

Fundamentals of language

STRINGS	
TERMINAL SYMBOLS	
PHRASE STRUCTURE	
NOUN PHRASE	
VERB PHRASE	
SENTENCE	
NONTERMINAL SYMBOLS	
REWRITE RULES	

languages, we will make use of all the tools of formal language theory, starting with the Backus-Naur form (BNF) notation, which is described in Appendix B on page 854.

A formal language is defined as a set of strings, where each string is a sequence of symbols taken from a finite set called the **terminal symbols**. For English, the terminal symbols include words like *a, aardvark, aback, abacus*, and about 400,000 more.

One of the confusing things in working with both formal and natural languages is that there are so many different formalisms and notations for writing grammars (see the Historical Notes section for this chapter). However, most of them are similar in that they are based on the idea of **phrase structure**—that strings are composed of substrings called **phrases**, which come in different categories. For example, the phrases "the wumpus," "the king," and "the agent in the corner" are all examples of the category **noun phrase** (or *NP* for short). There are two reasons for identifying phrases in this way. First, phrases are convenient handles on which we can attach semantics. Second, categorizing phrases helps us to describe the allowable strings of the language. We can say that any of the noun phrases can combine with a **verb phrase** (or *VP*) such as "is dead" to form a phrase of category **sentence** (or *S*). Without the intermediate notions of noun phrase and verb phrase, it would be difficult to explain why "the wumpus is dead" is a sentence whereas "wumpus the dead is" is not. Grammatical categories are essentially posited as part of a scientific theory of language that attempts to account for the difference between grammatical and ungrammatical categories. It is theoretically possible, although perhaps unlikely, that in some future theory of the English language, the *NP* and *VP* categories may not exist.

Categories such as *NP*, *VP*, and *S* are called **nonterminal symbols**. In the BNF notation, **rewrite rules** consist of a single nonterminal symbol on the left-hand side, and a sequence of terminals or nonterminals on the right-hand side. The meaning of a rule such as

$$S \rightarrow NP \ VP$$

is that we can take any phrase categorized as a *NP*, append to it any phrase categorized as a *VP*, and the result will be a phrase categorized as an *S*.

The component steps of communication

A typical communication episode, in which speaker *S* wants to convey proposition *P* to hearer *H* using words *W*, is composed of seven processes. Three take place in the speaker:

Intention: *S* wants *H* to believe *P* (where *S* typically believes *P*)

Generation: *S* chooses the words *W* (because they express the meaning *P*)

Synthesis: *S* utters the words *W* (usually addressing them to *H*)

Four take place in the hearer:

Perception: *H* perceives *W* (ideally *W'* = *W*, but misperception is possible)

Analysis: *H* infers that *W* has possible meanings *P*₁, ..., *P*_{*n*} (words and phrases can have several meanings)

Disambiguation: *H* infers that *S* intended to convey *P*_{*i*} (where ideally *Pj* = *P*, but misinterpretation is possible)

Incorporation: *H* decides to believe *P*_{*i*} (or rejects it if it is out of line with what *H* already believes)

GENERATIVE CAPACITY

Grammatical formalisms can be classified by their **generative capacity**: the set of languages they can represent. Chomsky (1957) describes four classes of grammatical formalisms that differ only in the form of the rewrite rules. The classes can be arranged in a hierarchy, where each class can be used to describe all the languages that can be described by a less powerful class, as well as some additional languages. Here we list the hierarchy, most powerful class first:

Recursively enumerable grammars use unrestricted rules: both sides of the rewrite rules can have any number of terminal and nonterminal symbols. These grammars are equivalent to Turing machines in their expressive power.

Context-sensitive grammars are restricted only in that the right-hand hand side must contain at least as many symbols as the left-hand side. The name context-sensitive comes from the fact that a rule such as $ASB \rightarrow AXB$ says that an S can be rewritten as an X in the context of a preceding A and a following B .

In **context-free grammars** (or CFGs), the left-hand side consists of a single nonterminal symbol. Thus, each rule licenses rewriting the nonterminal as the right-hand side in *any* context. CFGs are popular for natural language grammars, although it is now widely accepted that at least some natural languages are not context-free (Pullum, 1991).

Regular grammars are the most restricted class. Every rule has a single nonterminal on the left-hand side, and a terminal symbol optionally followed by a nonterminal on the right-hand side. Regular grammars are equivalent in power to finite-state machines. They are poorly suited for programming languages because, for example, they cannot represent constructs such as balanced opening and closing parentheses.

To give you an idea of which languages can be handled by which classes, the language $a^n b^n$ (a sequence of n copies of a followed by the same number of b) can be generated by a context-free grammar, but not a regular grammar. The language $a^n b^n c^n$ requires a context-sensitive grammar, whereas the language $a^* b^*$ (a sequence of any number of a followed by any number of b) can be described by any of the four classes. A summary of the four classes follows.

Class	Sample Rule	Sample Language
Recursively enumerable	$A \rightarrow C$	any
Context-sensitive	$A \rightarrow BA$	$a^n b^n c^n$
Context-free	$S \rightarrow a S b$	$a^n b^n$
Regular	$S \rightarrow a S$	$a^* b^*$

INTENTION

Let us look at these seven processes in the context of the example shown in Figure 22.1.

GENERATION

Intention. Somehow, the speaker decides that there is something that is worth saying to the hearer. This often involves reasoning about the beliefs and goals of the hearer, so that the utterance will have the desired effect. For our example, the speaker has the intention of having the hearer know that the wumpus is no longer alive.

SYNTHESIS

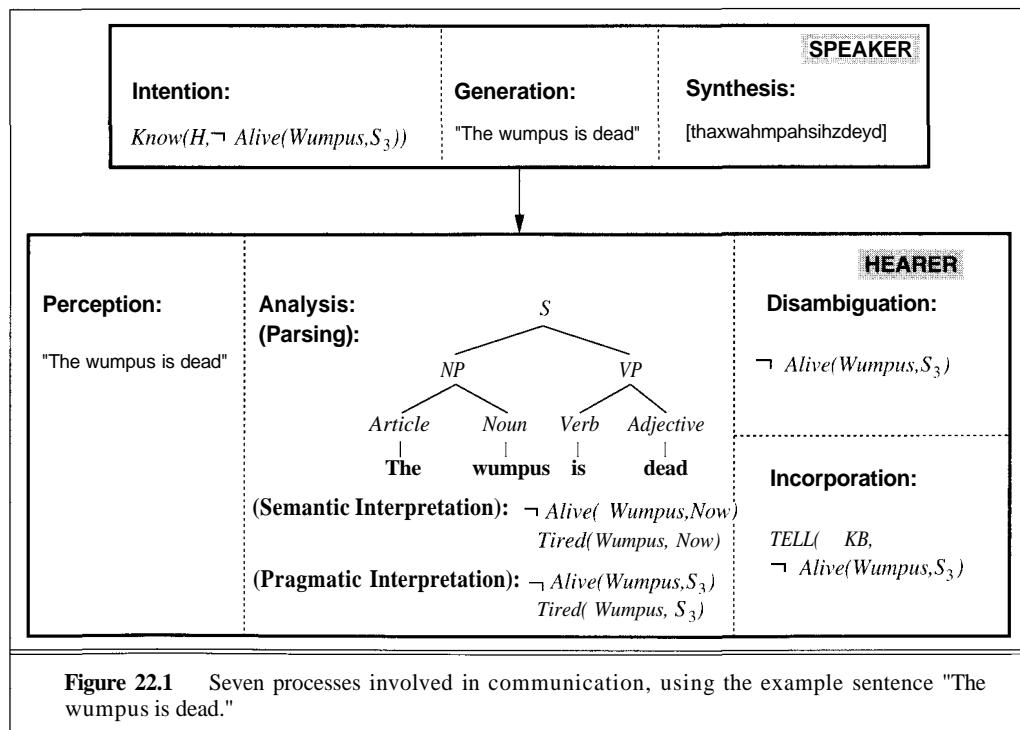
Generation. The speaker uses knowledge about language to decide what to say. In many ways, this is harder than the inverse problem of understanding (i.e., analysis and disambiguation). Generation has not been stressed as much as understanding in AI, mainly because we humans are anxious to talk to machines, but are not as excited about them talking back. For now, we just assume the hearer is able to choose the words "The wumpus is dead."

PERCEPTION

Synthesis. Most language-based AI systems synthesize typed output on a screen or paper, which is a trivial task. Speech synthesis has been growing in popularity, and some systems are beginning to sound human. In Figure 22.1, we show the agent synthesizing a string of sounds written in the phonetic alphabet defined on page 758: "[thaxwahmpahsihzdeyd]." The details of this notation are unimportant; the point is that the sounds that get synthesized are different from the words that the agent generates. Also note that the words are run together; this is typical of quickly spoken speech.

Perception. When the medium is speech, the perception step is called **speech recognition**;

when it is printing, it is called **optical character recognition**. Both have moved from being



ANALYSIS

esoteric to being commonplace within the last five years. For the example, let us assume that the hearer perceives the sounds and recovers the spoken words perfectly. (In Chapter 24 we see how this might be done.)

PARSING

Analysis. We divide analysis into two main parts: syntactic interpretation (or parsing) and semantic interpretation. Semantic interpretation includes both understanding the meanings of words and incorporating knowledge of the current situation (also called pragmatic interpretation).

PARSE TREE

The word **parsing** is derived from the Latin phrase *pars orationis*, or "part of speech," and refers to the process of assigning a part of speech (noun, verb, and so on) to each word in a sentence and grouping the words into phrases. One way of displaying the result of a syntactic analysis is with a **parse tree**, as shown in Figure 22.1. A parse tree is a tree in which interior nodes represent phrases, links represent applications of grammar rules, and leaf nodes represent words. We define the **yield** of a node as the list of all the leaves below the node, in left-to-right order, then we can say that the meaning of a parse tree is that each node with label X asserts that the yield of the node is a phrase of category X .

SEMANTIC INTERPRETATION

Semantic interpretation is the process of extracting the meaning of an utterance as an expression in some representation language. In Figure 22.1 we show two possible semantic interpretations: that the wumpus is not alive, and that it is tired (a colloquial meaning of *dead*). Utterances with several possible interpretations are said to be **ambiguous**. We use logic as the representation language, but other representations could be used. **Pragmatic interpretation** is the part of semantic interpretation that takes the current situation into account.² In the example, all that pragmatics does is replace the constant *Now* with the constant S_3 , which stands for the current situation.

PRAGMATIC INTERPRETATION

Disambiguation. Most speakers are not intentionally ambiguous, but most utterances have several legal interpretations. Communication works because the hearer does the work of figuring out which interpretation is the one the speaker probably meant to convey. Notice that this is the first time we have used the *word probably*, and disambiguation is the first process that depends heavily on uncertain reasoning. Analysis generates possible interpretations; if more than one interpretation is found, then disambiguation chooses the one that is best.

DISAMBIGUATION

Incorporation. A totally naive agent might believe everything it hears, but a sophisticated agent treats the words W and the derived interpretation P_i as additional pieces of evidence that get considered along with all other evidence for and against P_i .

INCORPORATION

Note that it only makes sense to use language when there are agents to communicate with who (a) understand a common language, (b) have a shared context on which to base the conversation, and (c) are at least somewhat rational. Communication does not work when agents are completely irrational, because there is no way to predict how an irrational agent will react to a speech act. Interestingly, communication can work when agents are uncooperative. Even if you believe that another wumpus world explorer would lead you astray in order to get the gold all for itself, you can still communicate to help each other kill the wumpus or perform some other task that is helpful to both agents. Returning to Africa for another example, when an antelope sees a

² Thus, pragmatic interpretation associates meanings with *utterances made* in specific contexts, whereas the rest of semantic interpretation associates meanings with *strings* in isolation. This is controversial, and other authors draw the line between semantics and pragmatics in different places, or just group them together. Also, some authors use the term **parsing** to encompass all of what we call analysis.

predator at a safe distance, it will stot, or leap high into the air. This not only communicates to other antelopes that danger is near, but also communicates to the predator "I see you, and I am healthy enough to run away, so don't even bother chasing me." So even though the two animals are enemies with no common goal, the communication saves both of them from wasting time and energy on a fruitless chase.

Two models of communication

ENCODED MESSAGE

Our study of communication centers on the way that an agent's beliefs are turned into words and back into beliefs in another agent's knowledge base (or head). There are two ways of looking at the process. The **encoded message** model says that the speaker has a definite proposition P in mind, and encodes the proposition into the words (or signs) W . The hearer then tries to decode the message W to retrieve the original proposition P (cf. Morse code). Under this model, the meaning in the speaker's head, the message that gets transmitted, and the interpretation that the hearer arrives at all ideally carry the same content. When they differ, it is because of noise in the communication channel or an error in encoding or decoding.

SITUATED LANGUAGE

Limitations of the encoded message model led to the **situated language** model, which says that the meaning of a message depends on both the words and the **situation** in which the words are uttered. In this model, just as in situation calculus, the encoding and decoding functions take an extra argument representing the current situation. This accounts for the fact that the same words can have very different meanings in different situations. "I am here now" represents one fact when spoken by Peter in Boston on Monday, and quite another fact when spoken by Stuart in Berkeley on Tuesday. More subtly, "You must read this book" is a suggestion when written by a critic in the newspaper, and an assignment when spoken by an instructor to the class. "Diamond" means one thing when the subject is jewelry, and another when the subject is baseball.

The situated language model points out a possible source of communication failure: if the speaker and hearer have different ideas of what the current situation is, then the message may not get through as intended. For example, suppose agents X , Y , and Z are exploring the wumpus world together. X and Y secretly meet and agree that when they smell the wumpus they will both shoot it, and if they see the gold, they will grab it and run, and try to keep Z from sharing it. Now suppose X smells the wumpus, while Y in the adjacent square smells nothing, but sees the gold. X yells "Now!", intending it to mean that now is the time to shoot the wumpus, but Y interprets it as meaning now is the time to grab the gold and run.

22.2 TYPES OF COMMUNICATING AGENTS

In this chapter, we consider agents that communicate in two different ways. First are agents who share a common internal representation language; they can communicate without any external language at all. Then come agents that make no assumptions about each other's internal language, but share a communication language that is a subset of English.

TELEPATHIC
COMMUNICATION

Communicating using Tell and Ask

In this section, we study a form of communication in which agents share the same internal representation language and have direct access to each other's knowledge bases through the TELL and ASK interface. That is, agent A can communicate proposition P to agent B with $\text{TELL}(KB_B, "P")$, just as A would add P to its own knowledge base with $\text{TELL}(KB_A, "P")$. Similarly, agent A can find out if B knows Q with $\text{ASK}(KB_B, "Q")$. We will call this **telepathic communication**. Figure 22.2 shows a schematic diagram in which each agent is modified to have an input/output port to its knowledge base, in addition to the perception and action ports.

Humans are lacking in telepathy powers, so they cannot make use of this kind of communication, but it is feasible to program a group of robots with a common internal representation language and equip them with radio or infrared links to transmit internal representations directly to each other's knowledge bases. Then if agent A wanted to tell agent B that there is a pit in location [2,3], all A would have to do is execute:

$\text{TELL}(KB_B, "Pit(P_{A1})A\ At(P_{A1}, [2, 3], S_{A9}))$

where S_{A9} is the current situation, and P_{A1} is A's symbol for the pit. For this to work, the agents have to agree not just on the format of the internal representation language, but also on a great many symbols. Some of the symbols are static: they have a fixed denotation that can be easily agreed on ahead of time. Examples are the predicates *At* and *Pit*, the constants *A* and *B* representing the agents, and the numbering scheme for locations.

Other symbols are dynamic: they are created after the agents start exploring the world. Examples include the constant P_{A1} representing a pit, and S_{A9} representing the current situation. The hard part is synchronizing the use of these dynamic symbols. There are three difficulties:

1. There has to be a naming policy so that A and B do not simultaneously introduce the same symbol to mean different things. We have adopted the policy that each agent includes its own name as part of the subscript to each symbol it introduces.
2. There has to be some way of relating symbols introduced by different agents, so that an agent can tell whether P_{A1} and, say, P_{B2} denote the same pit or not. In part, this is the same problem that a single agent faces. An agent that detects a breeze in two different squares

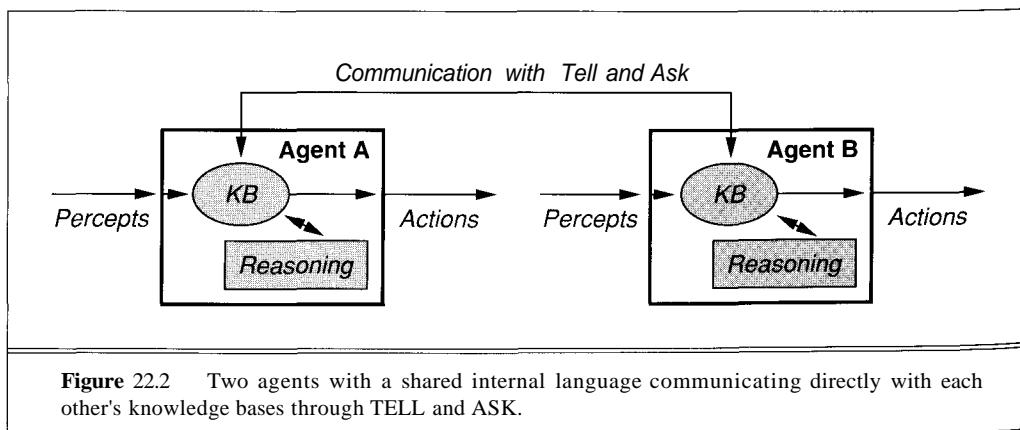


Figure 22.2 Two agents with a shared internal language communicating directly with each other's knowledge bases through TELL and ASK.

might introduce the symbols P_1 and P_2 to denote the two pits that caused the breezes, and must then do some reasoning to decide if $P_1 = P_2$. But the problem is harder for multiple agents because they share fewer symbols to begin with, and thus have less common ground to reason with. In particular, agent B has the problem of deciding how the situation S_{A9} relates to its own situation symbols. This problem can be lessened by minimizing the number of new symbols. For example, A could tell B that there is a pit in [2,3] without introducing any new dynamic symbols with the following action:

TELL(KB_B , “ $\exists p, s \ Pit(p) \wedge At(p, [2, 3], s)$ ”)

This sentence is weaker than the one containing P_{A1} and S_{A9} because it does not say which pit is in [2,3], nor when it was there, but it turns out that these facts do not really matter for the wumpus world. Note that this sentence is similar to the English sentence "There is a pit in 2,3," which also fails to uniquely identify the pit and the time.

3. The final difficulty is in reconciling the differences between different agents' knowledge bases. If communication is free and instantaneous, then all agents can adopt the policy of broadcasting each new fact to everyone else as soon as they learn it. That way everyone will have all the same knowledge. But in most applications, the bandwidth between agents is limited, and they are often completely out of touch with each other for periods of time. When they come back into contact, they have the problem of deciding what new information is worth communicating, and of discovering what interesting facts the other agent knows.

Another problem with telepathic agents as we have described them is that they are vulnerable to sabotage. Another agent could TELL lies directly into the knowledge base and make our naive telepathic agent believe anything.

Communicating using formal language

Because of the sabotage problem, and because it is infeasible for everyone to have the same internal language, most agents communicate through language rather than through direct access to knowledge bases. Figure 22.3 shows a diagram of this type of communication. Agents can perform actions that produce language, which other agents can perceive. The external communication language can be different from the internal representation language, and the agents can each have different internal languages. They need not agree on any internal symbols at all as long as each one can map reliably from the external language to its own internal symbols.

An external communication language brings with it the problems of generation and analysis, and much of the effort in natural language processing (NLP) has gone into devising algorithms for these two processes. But the hardest part of communication with language is still problem (3) from the previous section: reconciling the differences between different agents' knowledge bases. What agent A says, and how agent B interprets A 's statement depends crucially on what A and B already believe (including what they believe about each other's beliefs). This means that agents that have the same internal and external language would have an easy time with generation and analysis, but they would still find it challenging to decide what to say to each other.

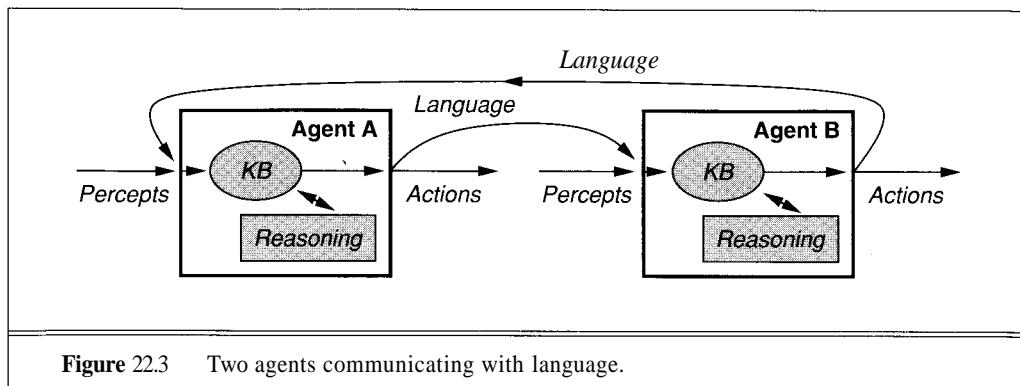


Figure 22.3 Two agents communicating with language.

An agent that communicates

We will now look at an example of communication, in the form of a wumpus world agent that acts as a robot slave that can be commanded by a master. On each turn, the slave describes its percepts in English (actually in a restricted subset), waits for a command from the master, and then interprets the command and executes it. Here is a fragment of a typical dialogue:

ROBOT SLAVE	MASTER
I feel a breeze.	Go to 1 2.
Nothing is here.	Go north.
I feel a breeze and I smell a stench and I see a glitter.	Grab it.
⋮	⋮

The agent program is shown in Figure 22.4. The rest of this chapter fills in the missing routines in the agent program. First, Section 22.3 defines the subset of English in which the agents communicate. Then, Section 22.4 shows how to implement the PARSE function to syntactically analyze strings of the language. Section 22.7 shows how to recover the meaning of a string (the function SEMANTICS), and Section 22.8 shows what to do when there are several possible meanings (the function DISAMBIGUATE). The function GENERATE-DESCRIPTION is not covered in depth, but in general much the same knowledge that is used for understanding can also be used for producing language.

22.3 A FORMAL GRAMMAR FOR A SUBSET OF ENGLISH

In this section, we define a formal grammar for a small subset of English that is suitable for making statements about the wumpus world. We will call this language \mathcal{E}_0 . In defining the language this way, we are implicitly claiming that formal language techniques are appropriate for dealing with natural language. In many ways, they *are* appropriate: natural languages make use

```
function SIMPLE-COMMUNICATING-AGENT(percept) returns action
  static: KB, a knowledge base
        t, a counter, initially 0, indicating time

  TELL(KB,MAKE-PERCEPT-SENTENCE(KB, t))
  words  $\leftarrow$  SPEECH-PART(percept)
  semantics  $\leftarrow$  DISAMBIGUATE(SEMATICS(PARSE(words)))
  if TYPE[semantics] = Command then
    action  $\leftarrow$  CONTENTS[semantics]
  else if TYPE[semantics] = Statement then
    TELL(KB,CONTENTS[semantics])
    action  $\leftarrow$  ASK(KB,MAKE-ACTION-QUERY(percept, t))
  else if TYPE[semantics] = None then
    action  $\leftarrow$  ASK(KB,MAKE-ACTION-QUERY(percept, t))
    description  $\leftarrow$  GENERATE-DESCRIPTION(percept)
  return COMPOUND-ACTION(SAY(description)DO(action))
```

Figure 22.4 A communicating agent that accepts commands and statements, and returns a compound action that both describes its percepts and does something else.

of a fixed set of letters (for written language) or sounds (for spoken language) that combine into a relatively fixed set of words. Either level can be considered the symbols of a formal language. The symbols are formed into strings, and it is clear that strings such as "This is a sentence" are part of the English language, and strings such as "A is sentence this" are not. We can come up with a set of grammar rules that cover at least part of English, and compose phrases to form arbitrarily complex sentences.

However, there are ways in which natural languages differ from formal ones. It is hard to characterize a natural language as a set of strings for four reasons. First, not all speakers agree on what is in the language. Some Canadians end sentences with "eh?," some U.S. southerners use "y'all," and one can start arguments in some circles by asking whether "This ain't a sentence" is. Second, the language changes over time—the English of Shakespeare's day is quite different from what we speak today. Third, some utterances that are clearly ungrammatical are nevertheless understandable. Fourth, grammaticality judgments are often graded rather than absolute. That means that there are some sentences that do not sound quite right, but are not clearly wrong, either. The following sentences (for most speakers) range from good to bad:

To whom did you send the letter?

Next to whom did you stand?

Of whom did you meet a friend?

Of whom did you see the dealer that bought the picture that Vincent painted?

Even if we could agree on exactly which sentences are English and which are not, that would be only a small part of natural language processing. The really hard parts are semantic interpretation and disambiguation. Speech act interpretation (which takes place across the syntactic, semantic, and pragmatic levels) also complicates the picture. In programming languages, every statement

is a command, but in natural language the hearer has to determine if an utterance is a command, question, statement, promise, or whatever. Formal language theory provides a framework within which we can address these more difficult problems, but it does not answer them on its own. In the remainder of this section, we define a formal language for our English subset, \mathcal{E}_0 .

The Lexicon of \mathcal{E}_0

LEXICON

The first step in defining a grammar is to define a **lexicon**, or list of allowable vocabulary words. The words are grouped into the categories or parts of speech familiar to dictionary users: nouns, pronouns, and names to denote things, verbs to denote events, adjectives to modify nouns, and adverbs to modify verbs. Categories that may be less familiar to some readers are **articles** (such as *the*), **prepositions** (*in*), and **conjunctions** (*and*).³ Figure 22.5 shows a small lexicon.

ARTICLES

PREPOSITIONS

CONJUNCTIONS

OPEN CLASSES

CLOSED CLASSES

Each of the categories ends in ... to indicate that there are other words in the category. However, it should be noted that there are two distinct reasons for the missing words. For nouns, verbs, adjectives, and adverbs, it is in principle infeasible to list them all. Not only are there thousands or tens of thousands of members in each class, but new ones are constantly being added. For example, "fax" is now a very common noun and verb, but it was coined only a few years ago. These four categories are called **open classes**. The other categories (pronoun, article, preposition, and conjunction) are called **closed classes**. They have a small number of words (a few to a few dozen) that could in principle be enumerated. Closed classes change over the course of centuries, not months. For example, "thee" and "thou" were commonly used pronouns in the seventeenth century, on the decline in the nineteenth, and are seen only in poetry and regional dialects today.

The Grammar of \mathcal{E}_0

The next step is to combine the words into phrases. We will use five nonterminal symbols to define the different kinds of phrases: sentence (*S*), noun phrase (*NP*), verb phrase (*VP*), prepositional phrase (*PP*), and relative clause (*RelClause*).⁴ Figure 22.6 shows a grammar for \mathcal{E}_0 with an example for each rewrite rule.

22.4 SYNTACTIC ANALYSIS (PARSING)

PARSE FOREST

There are many algorithms for **parsing**—recovering the phrase structure of an utterance, given a grammar. In Figure 22.7, we give a very simple algorithm that nondeterministically chooses one possible parse tree, if one exists. It treats the list of words as a **parse forest**: an ordered list of parse trees. On each step through the main loop, it finds some subsequence of elements in the

³ The term **conjunction** here means something that joins two phrases together. In addition to *and*, it includes *but*, *since*, *while*, *or*, and *because*. Do not be confused by the fact that *or* is logically a disjunction but syntactically a conjunction.

⁴ A relative clause follows and modifies a noun phrase. It consists of a relative pronoun (such as "who" or "that") followed by a verb phrase (and sometimes a whole sentence). An example of a relative clause is *that gave me the gold* in "The agent *that gave me the gold* is in 2,2."

```

Noun — stench | breeze | glitter | nothing  

    | wumpus | pit | pits | gold | east | ...  

Verb → is | see | smell | shoot | feel | stinks  

    | go | grab | carry | kill | turn | ...  

Adjective → right | left | east | south | back | smelly | ...  

Adverb → here | there | nearby | ahead  

    | right | left | east | south | back | ...  

Pronoun → me | you | I | it | ...  

Name → John | Mary | Boston | Aristotle ...  

Article → the | a | an | ...  

Preposition → to | in | on | near | ...  

Conjunction → and | or | but | ...  

Digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

Figure 22.5 The lexicon for \mathcal{E}_0 .

$S \rightarrow NP VP$ $S Conjunction S$	I + feel a breeze I feel a breeze + and + I smell a wumpus
$NP \quad — Pronoun$ <i>Noun</i> <i>Article Noun</i> <i>Digit Digit</i> <i>NP PP</i> <i>NP RelClause</i>	 pits the + wumpus 34 the wumpus + to the east the wumpus + that is smelly
$VP \rightarrow Verb$ <i>VP NP</i> <i>VP Adjective</i> <i>VP PP</i> <i>VP Adverb</i>	stinks feel + a breeze is + smelly turn + to the east go + ahead
$PP \rightarrow Preposition NP$ <i>RelClause</i> → <i>that VP</i>	to + the east that + is smelly

Figure 22.6 The grammar for \mathcal{E}_0 , with example phrases for each rule.

forest that match the right-hand side of one of the grammar rules. It then replaces the subsequence with a single parse tree whose category is the left-hand side of the rule, and whose children are the nodes in the original subsequence. In Figure 22.8, we show a trace of the execution in parsing the string "the wumpus is dead." Every choice is a good one, so there is no backtracking.

There are many possible parsing algorithms. Some operate top-down, starting with an S and expanding it according to the grammar rules to match the words in the string. Some use a combination of top-down and bottom-up, and some use dynamic programming techniques to avoid the inefficiencies of backtracking. We cover one efficient algorithm for parsing context-free grammars in Section 23.2. But first, we will show how the parsing problem can be interpreted as a logical inference problem, and thus can be handled by general inference algorithms.

```

function BOTTOM-UP-PARSE(words, grammar) returns a parse tree
  forest — words
  loop do
    if LENGTH(forest) = 1 and CATEGORY(forest[1]) = START(grammar) then
      return forest[1]
    else
      i — choose from {1...LENGTH(forest)}
      rule — choose from RULES(grammar)
      n — LENGTH(RULE-RHS(rule))
      subsequence — SUBSEQUENCE(forest, /, i+n-)
      if MATCH(subsequence, RULE-RHS(rule)) then
        forest[i...i+n-1] — [MAKE-NODE(RULE-LHS(rule), subsequence)]
      else fail
    end
  
```

Figure 22.7 Nondeterministic bottom-up parsing algorithm for context-free grammars. It picks one parse to return. Each node in a parse tree has two fields: CATEGORY and CHILDREN.

<i>forest</i>	<i>subsequence</i>	<i>rule</i>
The wumpus is dead	The	Article — the
Article wumpus is dead	wumpus	Noun — wumpus
Article Noun is dead	Article No/in	NP — Article Noun
NP is dead	is	Verb — is
NP Verb dead	dead	Adjective —> dead
NP Verb Adjective	Verb	VP — Verb
NP VP Adjective	VP Adjective	VP — VP Adjective
NP VP	NP VP	S — NP VP
S		

Figure 22.8 Trace of BOTTOM-UP-PARSE on the string "The wumpus is dead."

22.5 DEFINITE CLAUSE GRAMMAR (DCG)

There are two problems with BNF. First, we are interested in using language for communication, so we need some way of associating a meaning with each string, and BNF only talks about strings, not meanings. Second, we will want to describe grammars that are context-sensitive, whereas BNF is strictly context-free. In this section, we introduce a formalism that can handle both of these problems.

The idea is to use the full power of first-order logic to talk about strings and their meanings. Each nonterminal symbol becomes a one-place predicate that is true of strings that are phrases of that category. For example, *Noun*("stench") is a true logical sentence, whereas *Noun*("the") is false. It is easy to write BNF rules as logical implication sentences in first-order logic:

BNF

$S \rightarrow NP\ VP$

Noun — **stench** | ...

First-Order Logic

$NP(s_1)A\ VP(s_2) \Rightarrow S(Append(s_1, s_2))$

$(s = "stench")V\ ... \Rightarrow Noun(s)$

LOGIC GRAMMAR

DEFINITE CLAUSE
GRAMMAR
DCG

The first of these rules says that if there is a string s_1 that is a noun phrase and a string s_2 that is a verb phrase, then the string formed by appending them together is a sentence. The second rule says that if s is the string "stench" (or one of the other words not shown), then the string s is a noun. A grammar written with logical sentences is called a **logic grammar**. Since unrestricted logical inference is computationally expensive, most logic grammars insist on a restricted format. The most common is **definite clause grammar** or **DCG**, in which every sentence must be a definite clause.⁵

The *DCG formalism* is attractive because it allows us to describe grammars in terms of something we understand well: first-order logic. Unfortunately, it has the disadvantage of being more verbose than BNF notation. We can have the best of both by defining a special DCG *notation* that is an extension of BNF, but retains the well-founded DCG semantics. From now on, when we say "definite clause grammar," we mean a grammar written in this special notation, which is defined as follows:

- The notation $X \rightarrow Y\ Z\ ...$ translates as $Y(s_1)A\ Z(s_2)A\ ... \Rightarrow X(Append(s_1, s_2, ...))$.
- The notation $X \rightarrow \text{word}$ translates as $X(["word"])$.
- The notation $X \rightarrow Y\ | \ Z\ | \ ...$ translates as $Y'(s)V\ Z'(s)V\ ... \Rightarrow X(s)$, where Y' is the translation into logic of the DCG expression Y .

These three rules allow us to translate BNF into definite clauses. We now see how to extend the notation to incorporate grammars that can not be expressed in BNF.

- Nonterminal symbols can be **augmented** with extra arguments. In simple BNF notation, a nonterminal such as NP gets translated as a one-place predicates where the single argument represents a string: $NP(s)$. In the augmented DCG notation, we can write $NP(sem)$, for example, to express "an NP with semantics sem ." This gets translated into logic as the two-place predicate $NP(sem, s)$.

⁵ A definite clause is a type of Horn clause that, when written as an implication, has exactly one atom in its consequent, and a conjunction of zero or more atoms in its antecedent, for example, $A_1\ A\ A_2\ A\ ... \Rightarrow C$.

- A **variable** can appear on the right-hand side of a DCG rule. The variable represents a single symbol in the input string, without saying what it is. We can use this to define the nonterminal category *Double* as the set of strings consisting of a word repeated twice. Here is the definition in both DCG notation and normal first-order logic notation:

DCG*Double* — *w w***First-Order Logic** $(s_1 = [w] \wedge s_2 = [w]) \Rightarrow Double(Append(s_1, s_2))$

- An arbitrary logical test can appear on the right-hand side of a rule. Such tests are enclosed in curly braces in DCG notation.

As an example using all three extensions, here are rules for *Digit* and *Number* in the grammar of arithmetic (see Appendix B). The nonterminals take an extra argument representing their semantic interpretation:

DCG*Digit(sem)* — *sem* { $0 < sem < 9$ }*Number(sem)* — *Digit(sem)**Number(sem)* — *Number(sem₁) Digit(sem₂)*
{ $sem = 10 \times sem_1 + sem_2$ }**First-Order Logic** $(s = [sem]) \Rightarrow Digit(sem, s)$ $Digit(sem, s) \Rightarrow Number(sem, s)$ $Number(sem, s_1) \wedge Digit(sem, s_2) \wedge sem = 10 \times sem_1 + sem_2 \Rightarrow Number(sem, Append(s_1, s_2))$

The first rule can be read as "a phrase of category *Digit* and semantic value *sem* can be formed from a symbol *sem* between 0 and 9." The second rule says that a number can consist of a single digit, and has the same semantics as that digit. The third can be read as saying "a number with semantic value *sem* can be formed from a number with semantic value *sem₁* followed by a digit with semantic value *sem₂*, where $sem = 10 \times sem_1 + sem_2$."

There are now five things that can appear on the right-hand side of a rule in DCG notation: an un-augmented nonterminal (*Digit*), an augmented nonterminal (*Digit(sem)*), a variable representing a terminal (*sem*), a constant terminal (*word*), or a logical test ({ $0 < sem < 9$ }).

The left-hand side must consist of a single nonterminal, with or without augmentation.

22.6 AUGMENTING A GRAMMAR

OVERGENERATES

The simple grammar for \mathcal{E}_0 generates many sentences of English, but it also **overgenerates**—generates sentences that are not grammatical—such as "Me smells a stench." There are two problems with this string: it should be "I," not "me," and it should be "smell," not "smells." To fix these problems, we will first determine what the facts of English are, and then see how we can get the grammar to reflect the facts.

CASES

Many languages divide their nouns into different **cases**, depending on their role in the sentence. Those who have taken Latin are familiar with this, but in English there is no notion of case on regular nouns, only on pronouns. We say that pronouns like "I" are in the subjective (or nominative) case, and pronouns like "me" are in the objective (or accusative) case. Many languages have a dative case for words in the indirect object position. Many languages also require **agreement** between the subject and main verb of a sentence. Here, too, the distinctions

AGREEMENT

are minimal in English compared to most languages; all verbs except "be" have only two forms. One form (e.g., "smells") goes with third person singular subjects such as "he" or "the wumpus." The other form (e.g., "smell") goes with all other subjects, such as "I" or "you" or "wumpuses."

Now we are ready to fix our grammar. We will consider noun cases first. The problem is that with the distinctions we have made so far (i.e., the nonterminals we have defined), English is not context-free. It is not the case that any NP can combine with a VP to form a sentence, for example, the NP "I" can, but "me" cannot. If we want to stick with a context-free grammar we will have to introduce new categories such as NP_S and NP_O , to stand for noun phrases in the subjective and objective case, respectively. We would also need to split the category *Pronoun* into the two categories *Pronouns* (which includes "I") and *Pronoun_O* (which includes "me"). The necessary changes to the grammar are outlined in Figure 22.9. Notice that all the NP rules are duplicated,

$S \rightarrow NP_S VP \mid \dots$
$NP_S \rightarrow Pronouns \mid Noun \mid Article\ Noun$
$NP_O \rightarrow Pronoun_O \mid Noun \mid Article\ Noun$
$VP \rightarrow VP\ NP_O \mid \dots$
$PP \rightarrow Preposition\ NP_O$
$Pronouns \rightarrow I \mid you \mid he \mid she \mid \dots$
$Pronoun_O \rightarrow me \mid you \mid him \mid her \mid \dots$

Figure 22.9 The changes needed in \mathcal{E}_0 to handle subjective and objective cases.

once for NPs and once for NP_O . It would not be too bad if the number of rules doubled once, but it would double again when we changed the grammar to account for subject/verb agreement, and again for the next distinction. Thus, the size of the grammar can grow exponentially with the number of distinctions we need to make.

AUGMENT

An alternative approach is to **augment** the existing rules of the grammar instead of introducing new rules. The result is a concise, compact grammar. We start by parameterizing the categories NP and *Pronoun* so that they take a parameter indicating their case. In the rule for S , the NP must be in the subjective case, whereas in the rules for VP and PP , the NP must be in the objective case. The rule for NP takes a variable as its argument. This use of a variable—avoiding a decision where the distinction is not important—is what keeps the size of the rule set manageable. Figure 22.10 shows the augmented grammar, which defines a language we call \mathcal{E}_1 .

The problem of subject/verb agreement could also be handled with augmentations, but we delay showing this until the next chapter. Instead, we address a slightly harder problem: verb subcategorization.

Verb Subcategorization

The \mathcal{E}_1 language is an improvement over \mathcal{E}_0 , but it still allows ungrammatical sentences. One problem is in the way verb phrases are put together. We want to accept verb phrases like "give me the gold" and "go to 1,2." All these are in \mathcal{E}_1 , but unfortunately so are "go me the gold"

$$\begin{aligned}
 S & \rightarrow NP(\text{Subjective}) VP \mid \dots \\
 NP(\text{case}) & \rightarrow \text{Pronoun(case)} \mid \text{Noun} \mid \text{Article Noun} \mid \dots \\
 VP & \rightarrow VP\ NP(\text{Objective}) \mid \dots \\
 PP & \rightarrow \text{Preposition } NP(\text{Objective}) \\
 \text{Pronoun(Subjective)} & \rightarrow I \mid you \mid he \mid she \mid \dots \\
 \text{Pronoun(Objective)} & \rightarrow me \mid you \mid him \mid her \mid \dots
 \end{aligned}$$

Figure 22.10 The grammar of \mathcal{E}_1 using augmentations to represent noun cases.

SUBCATEGORIZATION
COMPLEMENTS

SUBCATEGORIZATION
LIST

and "give to 1,2." To eliminate these, the grammar must state which verbs can be followed by which other categories. We call this the **subcategorization** information for the verb. Each verb has a list of **complements**—obligatory phrases that follow the verb within the verb phrase. So in "Give the gold to me," the *NP* "the gold" and the *PP* "to me" are complements of "give."⁶

A **subcategorization list** is a list of complement categories that the verb accepts, so "give" has the subcategorization list *[NP, PP]* in this example. It is possible for a verb to have more than one subcategorization list, just as it is possible for a word to have more than one category, or for a pronoun to have more than one case.⁷ In fact, "give" also has the subcategorization list *[NP, NP]*, as in "Give me the gold." We can treat this like any other kind of ambiguity. It is also important to know what subcategorization lists a verb does not take. The fact that "give" does not take the list *[PP]* means that "give to me" is not by itself a valid verb phrase. Figure 22.11 gives some examples of verbs and their subcategorization lists, or **subcats** for short.

Verb	Subcats	Example Verb Phrase
give	<i>[NP, PP]</i> <i>[NP, NP]</i>	give the gold in 3 3 to me give me the gold
smell	<i>[NP]</i> <i>[Adjective]</i> <i>[PP]</i>	smell a wumpus smell awful smell like a wumpus
is	<i>[Adjective]</i> <i>[PP]</i> <i>[NP]</i>	is smelly is in 2 2 is a pit
died	[]	died
believe	<i>[S]</i>	believe the smelly wumpus in 2 2 is dead

Figure 22.11 Examples of verbs with their subcategorization frames.

⁶ This is one definition of *complement*, but other authors have different terminology. Some say that the subject of the verb is also a complement. Others say that only the prepositional phrase is a complement, and the noun phrase should be called an **argument**.

⁷ For example, "you" can be used in either subjective or objective case.

To integrate verb subcategorization into the grammar, we do three things. The first step is to augment the category *VP* to take a subcategorization argument that indicates the complements that are needed to form a complete *VP*. For example, "give" can be made into a complete *VP* by adding *[NP, PP]*, "give the gold" can be made complete by adding *[PP]*, and "give the gold to me" is already complete; its subcategorization list is *[]*. That gives us these rules:

$$\begin{aligned} VP(subcat) \quad - & \quad VP([NP|subcat]NP(Objective)) \\ | & \quad VP([Adjective|subcat]) Adjective \\ | & \quad VP([PP|subcat]) PP \\ | & \quad Verb(subcat) \end{aligned}$$

The first line can be read as "A *VP* with a given subcat list, *subcat*, can be formed by a *VP* followed by a *NP* in the objective case, as long as that *VP* has a subcat list that starts with the symbol *NP* and is followed by the elements of the list *subcat*."

The second step is to change the rule for *S* to say that it requires a verb phrase that has all its complements, and thus has a subcat list of *[]*. This means that "He died" is a legal sentence, but "You give" is not. The new rule,

$$5 \quad - \quad NP(Subjective) VP([])$$

can be read as "A sentence can be composed of a *NP* in the subjective case, followed by a *VP* which has a null subcat list." Figure 22.12 shows a parse tree using this grammar.

ADJUNCTS

The third step is to remember that in addition to complements, verb phrases (and other phrases) can also take **adjuncts**, which are phrases that are not licensed by the individual verb but rather may appear in any verb phrase. Phrases representing time and place are adjuncts, because almost any action or event can have a time or place. For example, the adverb "now" in "I smell a wumpus now" and the *PP* "on Tuesday" in "give me the gold on Tuesday" are adjuncts. Here are two rules to allow adjuncts:

$$\begin{aligned} VP(subcat) \quad - & \quad VP(subcat) PP \\ | & \quad VP(subcat)Adverb \end{aligned}$$

Notice that we now have two rules with *VP PP* on the right-hand side, one as an adjunct and one as a complement. This can lead to ambiguities. For example, "I walked Sunday" is usually interpreted as an intransitive *VP* followed by a time adjunct meaning "I moved myself with my feet on Sunday." But if Sunday is the name of a dog, then Sunday is an *NP* complement, and the meaning is that I took the dog for a walk at some unspecified time.

RULE SCHEMA

Generative Capacity of Augmented Grammars

The generative capacity of augmented grammars depends on the number of values for the augmentations. If there is a finite number, then the augmented grammar is equivalent to a context-free grammar. To see this, consider each augmented rule as a **rule schema**, which stands for a set of rules, one for each possible combination of values for the augmented constituents. If we replace each rule schema with the complete set of rules, we end up with a finite (although perhaps exponentially large) set of context-free rules. But in the general case, augmented grammars go

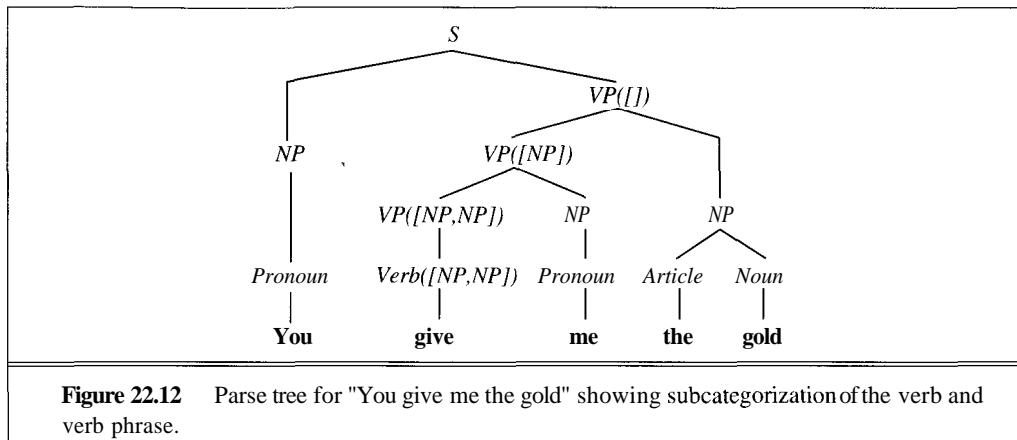


Figure 22.12 Parse tree for "You give me the gold" showing subcategorization of the verb and verb phrase.

beyond context-free. For example, the context-sensitive language $a^n b^n c^n$ can be represented with the following augmented grammar (where ϵ represents the empty string):

$$\begin{array}{ll}
 S(n) \rightarrow A(n) B(n) C(n) & \\
 A(0) \rightarrow \epsilon & A(n+1) \rightarrow a A(n) \\
 B(0) \rightarrow \epsilon & B(n+1) \rightarrow b B(n) \\
 C(0) \rightarrow \epsilon & C(n+1) \rightarrow c C(n)
 \end{array}$$

22.7 SEMANTIC INTERPRETATION

Throughout Part III, we gained experience in translating between English and first-order logic. There it was done informally, to get a feeling for what statements of first-order logic mean. Here we will do the same thing in a more carefully controlled way, to define what statements of \mathcal{E}_1 mean. Later on we will investigate the meaning of other types of speech acts besides sentences.

Before moving on to natural languages, we will consider the semantics of formal languages. It is easy to deal with meanings of expressions like " $X + Y$ " in arithmetic or $X A Y$ in logic because they have a **compositional semantics**. This means that the semantics of any phrase is a function of the semantics of its subphrases; it does not depend on any other phrase before, after, or encompassing the given phrase. So if we know the meaning of X and Y (and $+$), then we know the meaning of the whole phrase.

Things get more complicated when we go beyond the simple language of arithmetic. For example, the programming language expression "10+10" might have the semantic interpretation 4 when it appears in the larger string "BASE(2); x ← 10+10." However, we can still salvage a compositional semantics out of this if we say that the semantic function associated with " $x + y$ " is to add x and y in the current base. The great advantage of compositional semantics is the same as the advantage of context-free grammars: it lets us handle an infinite grammar with a finite (and often small) set of rules.

At first glance, natural languages appear to have a noncompositional semantics. In "The batter hit the ball," we expect the semantic interpretation of "batter" to be *one who swings a bat* and of "ball" to be *spherical sporting equipment*. But in "The chef mixed the batter to be served at the ball," we expect the two words to have different meanings. This suggests that the meaning of "batter" and "ball" depends noncompositionally on the surrounding context. However, these semantic interpretations are only the expected, preferred ones, not the only possible ones. It is *possible* that "The batter hit the ball" refers to a cake mixture making a grand appearance at a formal dance. If you work hard enough, you can invent a story where that is the preferred reading. In short, *semantic interpretation alone cannot be certain of the right interpretation of a phrase or sentence*. So we divide the work—semantic interpretation is responsible for combining meanings compositionally to get a set of possible interpretations, and disambiguation is responsible for choosing the best one.

There are other constructions in natural language that pose problems for the compositional approach. The problem of quantifier scope (page 678) is one example. But overall, the compositional approach is the one most favored by modern natural language systems.

Semantics as DCG Augmentations

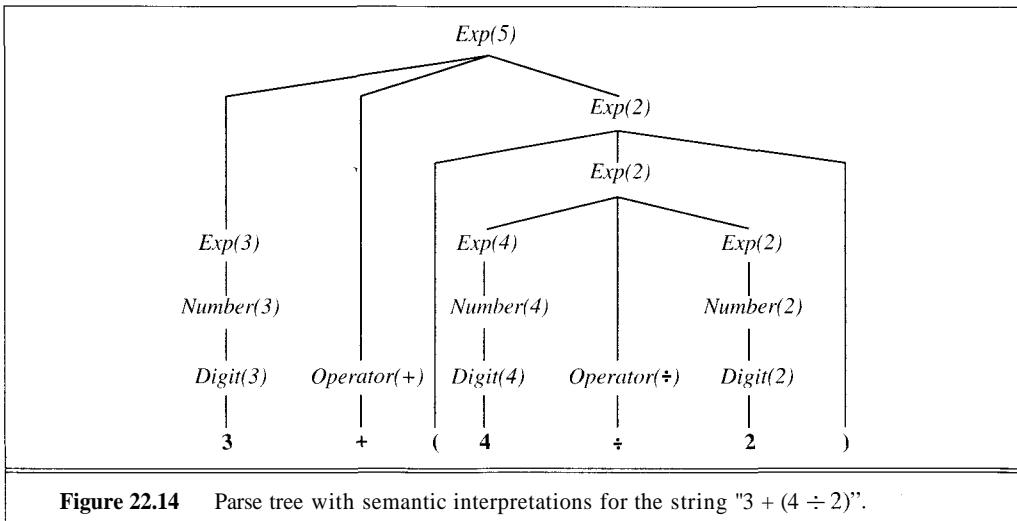
On page 668 we saw how augmentations could be used to specify the semantics of numbers and digits. In fact, it is not difficult to use the same idea to specify the semantics of the complete language of arithmetic, as we do in Figure 22.13. Figure 22.14 shows the parse tree for $3 + (4 - 2)$ according to this grammar, with the semantic augmentations. The string is analyzed as $Exp(5)$, an expression whose semantic interpretation is 5.

```
Exp(sem) — Exp(sem1) Operator(op)Exp(sem2) {sem = Apply(op, sem1, sem2)}
Exp(sem) → ( Exp(sem) )
Exp(sem) → Number(sem)
Digit(sem) → sem {0 < sem < 9}
Number(sem) → Digit(sem)
Number(sem) → Number(sem1)Digit(sem2) {sem = 10 × sem1 + sem2}
Operator(sem) → sem {sem G {+, -, ÷, ×}}
```

Figure 22.13 A grammar for arithmetic expressions, with semantics.

The semantics of "John loves Mary"

We are now ready to write a grammar with semantics for a very small subset of English. As usual, the first step is to determine what the facts are—what semantic representations we want to associate with what phrases. We will look at the simple sentence "John loves Mary" and associate with it the semantic interpretation *Loves(John, Mary)*. It is trivial to see which parts of the semantic interpretation come from which words in the sentence. The complicated part is



deciding how the parts fit together, particularly for intermediate phrases such as the VP "loves Mary." Note that the semantic interpretation of this phrase is neither a logical term nor a complete logical sentence. Intuitively, "loves Mary" is a description that may or may not apply to a particular person (in this case, it applies to John). This means that "loves Mary" is a **predicate** that, when combined with a term that represents a person (the person doing the loving), yields a complete logical sentence. Using the A-notation (see page 195), we can represent "loves Mary" as the predicate

$$\lambda x \text{ Loves}(x, \text{Mary})$$

The *NP* "Mary" can be represented by the logical constant *Mary*. That sets us up to define a rule that says "an *NP* with semantics *obj* followed by a *VP* with semantics *rel* yields a sentence whose semantics is the result of applying the relation *rel* to the object *obj*:"

$$S(\text{rel}(\text{obj})) \rightarrow \text{NP}(\text{obj}) \text{ VP}(\text{rel})$$

The rule tells us that the semantic interpretation of "John loves Mary" is

$$(\lambda x \text{ Loves}(x, \text{Mary}))(John)$$

which is equivalent to *Loves(John, Mary)*.

The rest of the semantics follows in a straightforward way from the choices we have made so far. Because *VPs* are represented as predicates, it is a good idea to be consistent and represent verbs as predicates as well. The verb "loves" is represented as $\lambda y \lambda x \text{ Loves}(x, y)$, the predicate that, when given an argument such as *Mary*, returns the predicate $\lambda x \text{ Loves}(x, \text{Mary})$.

The *VP* \rightarrow *Verb NP* rule applies the predicate that is the semantic interpretation of the verb to the object that is the semantic interpretation of the *NP* to get the semantic interpretation of the whole *VP*. We end up with the grammar shown in Figure 22.15 and the parse tree shown in Figure 22.16.

$$\begin{aligned} S(\text{rel}(obj)) &\rightarrow NP(obj) \quad VP(\text{rel}) \\ VP(\text{rel}(obj)) &\rightarrow \text{Verb}(\text{rel}) \quad NP(obj) \\ NP(obj) &\rightarrow \text{Name}(obj) \end{aligned}$$

$$\begin{aligned} \text{Name}(John) &\rightarrow \textbf{John} \\ \text{Name}(Mary) &\rightarrow \textbf{Mary} \\ \text{Verb}(\lambda x \lambda y \text{Loves}(x, y)) &\rightarrow \textbf{loves} \end{aligned}$$

Figure 22.15 A grammar that can derive a parse tree and semantic interpretation for "John loves Mary" (and three other sentences).

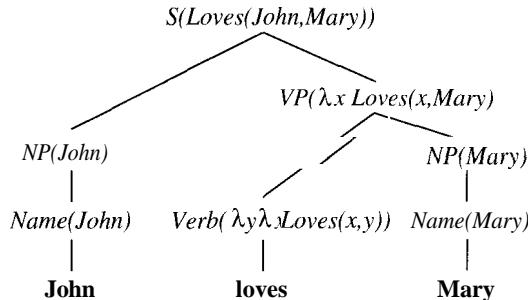


Figure 22.16 A parse tree with semantic interpretations for the string "John loves Mary".

The semantics of \mathcal{E}_1

We had no problem with "John loves Mary," but things get more complicated when we consider all of \mathcal{E}_1 . Immediately we are faced with all the choices of Chapter 8 for our semantic representation; for example, how do we represent time, events, and substances? Our first choice will be to use the event calculus notation of Section 8.4. In this notation, the sentence "Every agent smells a wumpus" can be expressed as:

$$\forall a \text{ Agent}(a) \Rightarrow \exists w \text{ Wumpus}(w) \wedge \exists e \ e \in \text{Perceive}(a, w, \text{Nose}) \wedge \text{During}(\text{Now}, e)$$

We could have used a *Smell* predicate instead of *Perceive*, but we wanted to be able to emphasize the similarities between smelling, hearing, feeling, touching, and seeing.

Our task is to build up our desired representation from the constituents of the sentence. We first break the sentence into *NP* and *VP* phrases, to which we can assign the following semantics:

$$\begin{aligned} \text{Every agent } \quad NP(\forall a \text{ Agent}(a)) &\Rightarrow \dots \\ \text{smells a wumpus } \quad VP(\exists w \text{ Wumpus}(w)) \wedge & \\ &\exists e \ (e \in \text{Perceive}(\dots, w, \text{Nose}) \wedge \text{During}(\text{Now}, e)) \end{aligned}$$

Right away there are two problems. First, the semantics of the entire sentence appears to be the semantics of the *NP* with the semantics of the *VP* filling in the \dots part. That means that we cannot form the semantics of the sentence with *rel(obj)*. We could do it with *obj(rel)*, which seems a

little odd (at least at first glance). The second problem is that we need to get the variable *a* as an argument to the relation *Perceive*. In other words, the semantics of the sentence is formed by plugging the semantics of the *VP* into the right argument slot of the *NP*, while also plugging the variable *a* from the *NP* into the right argument slot of the semantics of the *VP*. It looks as if we need two functional compositions, and promises to be rather confusing. The complexity stems from the fact that the semantic structure is very different from the syntactic structure.

To avoid this confusion, many modern grammars take a different tack. They define an **intermediate form** to mediate between syntax and semantics. The intermediate form has two key properties. First, it is structurally similar to the syntax of the sentence, and thus can be easily constructed through compositional means. Second, it contains enough information so that it can be translated into a regular first-order logical sentence. Because it sits between the syntactic and logical forms, it is sometimes called a **quasi-logical form**.⁸ In this chapter, we will use a quasi-logical form that includes all of first-order logic and is augmented by lambda expressions and one new construction, which we will call a **quantified term**. The quantified term that is the semantic interpretation of "every agent" is written

$[\forall a \text{ Agent}(a)]$

This looks like a logical sentence, but it is used in the same way that a logical term is used. In the following example, we see quantified terms as arguments to the relation *Perceive* in the interpretation of "Every agent smells a wumpus":

$3 e (e 6 \text{ Perceive}([\forall a \text{ Agent}(a)], [3 w \text{ Wumpus}(w)]) \text{ Nose}) \text{ A During}(Now, e))$

We will write our grammar so that it generates this quasi-logical form. In Section 22.7 we will see how to translate this into regular first-order logic.

It can be difficult to write a complex grammar that always comes up with the right semantic interpretation, and everyone has their own way of attacking the problem. We suggest a methodology based on these steps:

1. Decide on the logical or quasi-logical form you want to generate. Write down some example sentences and their corresponding logical forms. One such example sentence is at the top of Figure 22.17.
2. Make one-word-at-a-time modifications to your example sentences, and study the corresponding logical forms. For example, the semantics of "Every agent smelled a wumpus" is the same as our example sentence, except that *During(Now, e)* is replaced with *After(Now, e)*. This suggests that *During* is part of the semantics of "smells" and *After* is part of the semantics of "smelled." Similarly, changing the word "every" to "an" might result in a change of \forall to \exists in the logical form. This gives you a hint about the semantic interpretation of "an" and "every."
3. Eventually you should be able to write down the basic logical type of each lexical category (noun, verb, and so on), along with some word/logical form pairs. This is motivated in part by example sentences and in part by your intuitions. For example, it seems clear enough that the pronoun "I" should denote the object *Speaker* (which happens to be a fluent, dependent on the situation). Once we decide that one word in a category is of a

⁸ Some quasi-logical forms have the third property that they can succinctly represent ambiguities that could only be represented in logical form by a long disjunction.

certain semantic type, then we know that everything in the category is of the same type. Otherwise, the compositionality would not work out right. See the middle of Figure 22.17 for types and examples of all the lexical categories.

4. Now consider phrase-at-a-time modifications to your example sentences (e.g., substituting "every stinking wumpus" for "I"). You should be able to determine examples and types for constituent phrases, as in the bottom of Figure 22.17. In Figure 22.18, we see the complete parse tree for a sentence.
5. Once you know the type of each category, it is not too hard to attach semantic interpretation augmentations to the grammar rules. Some of the rules have only one right-hand side constituent and only need to copy up the semantics of that constituent:

$$NP(sem) \rightarrow Pronoun(sem)$$

6. Other times, the right-hand side of a rule will contain a semantic interpretation that is a predicate (or function), and one or more that are objects. To get the semantics of the whole phrase, just apply the relation (or function) to the object(s):

$$S(rel(obj)) \rightarrow NP(obj) VP(rel)$$

7. Sometimes the semantics is built up by concatenating the semantics of the constituents, possibly with some connectors wrapped around them:

$$NP([sem_1, sem_2]) \rightarrow Digit(sem_1) Digit(sem_2)$$

8. Finally, sometimes you need to take apart one of the constituents before putting the semantics of the whole phrase back together. Here is a complex example:

$$VP(Xx rel_1(x) A rel_2(EVENT\text{-}VAR(rel_1))) \rightarrow VP(rel_1)Adverb(rel_2)$$

The intent here is that the function EVENT-VAR picks out the event variable from the intermediate form expression rel_1 . The end result is that a verb phrase such as "saw me yesterday" gets the interpretation:

$$\exists x \exists e \ e \in Sees(x, Speaker) A After(Now\#e) A During(e, Yesterday)$$

By following these steps, we arrive at the grammar in Figure 22.19. To actually use this grammar, we would augment it further with the case and subcategorization information that we worked out previously. There is no difficulty in combining such things with semantics, but the grammar is easier to understand when we look at one type of augmentation at a time.

Converting quasi-logical form to logical form

The final step of semantic interpretation is to convert the quasi-logical form into real first-order logic. For our quasi-logical form, that means turning quantified terms into real terms. This is done by a simple rule: For each quantified term $[qx P(x)]$ within a quasi-logical form QLF , replace the quantified term with x , and replace QLF with $q x P(x) op QLF$, where op is \Rightarrow when q is V, and is A when q is 3 or $\exists!$. For example, the sentence "Every dog has a day" has the quasi-logical form:

$$\exists e \ e \in Has([\forall d Dog(d)], [\exists a Day(a)], Now)$$

Category	Type	Example	Quasi-Logical Form
<i>S</i>	Sentence	I sleep.	$\exists e \ e \in (\text{Sleep}, \text{Speaker})$ A During(Now, <i>e</i>)
<i>Adjective</i>	$object \rightarrow sentence$	smelly	$\exists x \ Smelly(x)$
<i>Adverb</i>	$event \rightarrow sentence$	today	$\exists e \ During(e, \text{Today})$
<i>Article</i>	Quantifier	the	$3!$
<i>Conjunction</i>	$sentence^2 \rightarrow sentence$	and	$\lambda p, q \ (p \wedge q)$
<i>Digit</i>	Number	7	I
<i>Noun</i>	$object \rightarrow sentence$	wumpus	$\exists x \ Wumpus(x)$
<i>Preposition</i>	$object^2 \rightarrow sentence$	in	$\exists x \ \lambda y In(x, y)$
<i>Pronoun</i>	Object	I	<i>Speaker</i>
<i>Verb</i>	$object^n \rightarrow sentence$	eats	$\exists y \ Xx \ \exists e \ e \in Eats(x, y)$ A During(Now, <i>e</i>)
<i>NP</i>	Object	a dog	$\exists d \ Dog(d)$
<i>PP</i>	$object \rightarrow sentence$	in [2,2]	$\exists x \ In(x, [2, 2])$
<i>RelClause</i>	$object \rightarrow sentence$	that sees me	$\exists x \ \exists e \ e \in Sees(x, \text{Speaker})$ A During(Now, <i>e</i>)
<i>VP</i>	$object^n \rightarrow sentence$	sees me	$\exists x \ \exists e \ e \in Sees(x, \text{Speaker})$ A During(Now, <i>e</i>)

Figure 22.17 Table showing the type of quasi-logical form expression for each syntactic category. The notation $t \rightarrow r$ denotes a function that takes an argument of type t and returns a result of type r .

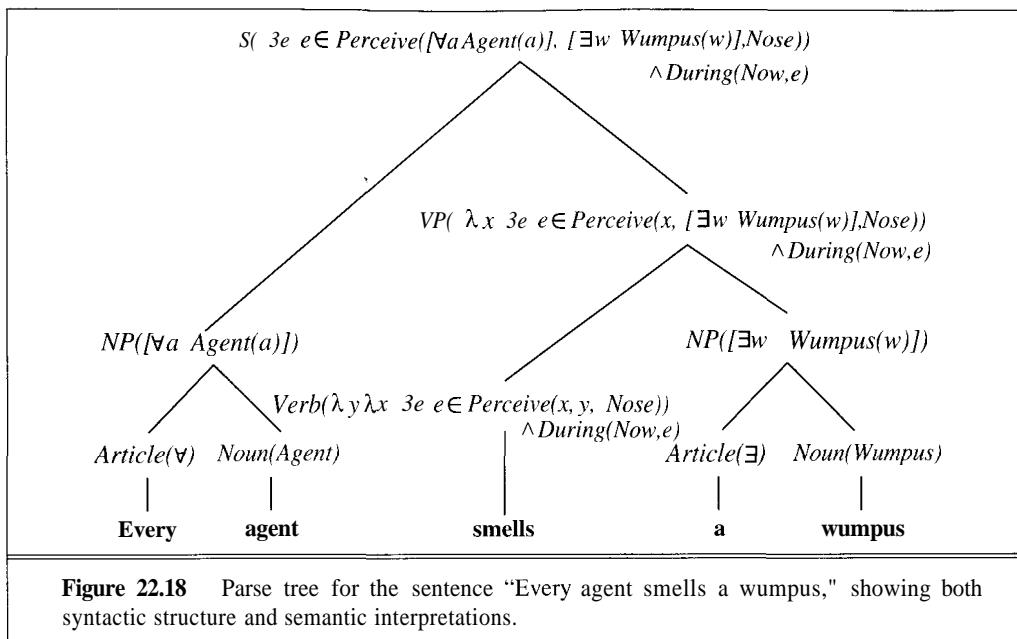
We did not specify which of the two quantified terms gets pulled out first, so there are actually two possible interpretations:

$$\begin{aligned} \forall d \ Dog(d) &\Rightarrow \exists a \ Day(a) \wedge \exists e \ e \in Has(d, a, \text{Now}) \\ \exists a \ Day(a) \wedge \forall d \ Dog(d) &\Rightarrow \exists e \ e \in Has(d, a, \text{Now}) \end{aligned}$$

The first one says that each dog has his own day, while the second says there is a special day that all dogs share. Choosing between them is a job for disambiguation. Often the left-to-right order of the quantified terms matches the left-to-right order of the quantifiers, but other factors come into play. The advantage of quasi-logical form is that it succinctly represents all the possibilities. The disadvantage is that it doesn't help you choose between them; for that we need the full power of disambiguation using all sources of evidence.

Pragmatic Interpretation

We have shown how an agent can perceive a string of words and use a grammar to derive a set of possible semantic interpretations. Now we address the problem of completing the interpretation by adding information about the current situation, information that is noncompositional and context-dependent.



INDEXICALS

The most obvious need for pragmatic information is in resolving the meaning of **indexicals**, which are phrases that refer directly to the current situation. For example, in the sentence "I am in Boston today," the interpretation of the indexicals "I" and "today" depend on who uttered the sentence when. We represent indexicals by Skolem constants (such as *Speaker*), which are interpreted as fluents. The hearer who perceives a speech act should also perceive who the speaker is, and use this information to resolve the indexical. For example, the hearer might know $T(\text{Speaker} = \text{Agent}_B, \text{Now})$.

ANAPHORA

Another important concern is **anaphora**, the occurrence of phrases referring to objects that have been mentioned previously. Consider the passage:

"John was hungry. He entered a restaurant."

To understand that "he" in the second sentence refers to John, we need to have processed the first sentence and used it as part of the situational knowledge in interpreting the second sentence. Anaphoric reference can also be made with definite noun phrases like "the man." In fact, the pattern of reference can be rather complicated, requiring a thorough understanding of the discourse. Consider the following sentence:

"After John proposed to Marsha, they found a preacher and got married. For the honeymoon, they went to Hawaii."

Here the definite noun phrase "the honeymoon" refers to something that was only implicitly alluded to by the verb "married." The pronoun "they" refers to a group that was not explicitly mentioned before: John and Marsha (but *not* the preacher).

When pronouns are used to refer to things within the same sentence, we at least get some help from syntax. For example, in "He saw him in the mirror" the two pronouns must refer to

```

 $S(rel(obj)) \rightarrow NP(obj) VP(rel)$ 
 $S(conj(sem_1, sem_2)) \rightarrow S(sem_1) Conjunction(conj)S(sem_2)$ 

 $NP(sem) \leftarrow \neg Pronoun(sem)$ 
 $NP(sem) \leftarrow \neg Name(sem)$ 
 $NP([qx sem(x)]) \rightarrow Article(q) Noun(sem)$ 
 $NP([qx obj A rel(x)]) \rightarrow NP([qx obj]) PP(rel)$ 
 $NP([qx obj A rel(x)]) \rightarrow NP([qx obj]) RelClause(rel)$ 
 $NP([sem_1, sem_2]) \rightarrow Digit(sem_1) Digit(sem_2)$ 

/* VP rules for subcategorization: * /
 $VP(sem) \rightarrow Verb(sem)$ 
 $VP(rel(obj)) \leftarrow VP(rel) NP(obj)$ 
 $VP(sem_1(sem_2)) \rightarrow VP(sem_1) Adjective(sem_2)$ 
 $VP(sem_1(sem_2)) \rightarrow VP(sem_1) PP(sem_2)$ 
/* VP rules for adjuncts: */
 $VP(\langle x sem_1(x) A sem_2 (\text{EVENT-VAR}(sem_1))) \rightarrow VP(sem_1) PP(sem_2)$ 
 $VP(\langle x sem_1(x) A sem_2 (\text{EVENT-VAR}(sem_1))) \rightarrow VP(sem_1) Adverb(sem_2)$ 

 $RelClause(sem) — that VP(sem)$ 

 $PP(Xx rel(x, obj)) \rightarrow Preposition(rel) NP(obj)$ 

```

Figure 22.19 A grammar for £2 with semantics.

different people, whereas in "He saw himself," they refer to the same person. But most of the time, there are no strict rules on anaphoric reference. So deciding which reference is the right one is a part of disambiguation, although the disambiguation is certainly guided by pragmatic (i.e., context-dependent) information.

22.8 AMBIGUITY AND DISAMBIGUATION

In the ideal communicative exchange, the speaker has a proposition P in mind and performs a speech act that may have several interpretations, but which in the current situation can best be interpreted as communicating P . The hearer realizes this, and so arrives at P as the interpretation. We say that the hearer has disambiguated or resolved the ambiguity. Occasionally, the hearer may be confused and need to ask for clarification, but it would be tiresome if this happened too often, or if the hearer asked for clarification on the clarification. Unfortunately, there are many ways in which communication can break down. A speaker who does not speak loudly enough



will not be heard. But *the biggest problem is that most utterances are ambiguous*. Here are some examples taken from newspaper headlines:

- Squad helps dog bite victim.
- Red-hot star to wed astronomer.
- Helicopter powered by human flies.
- Once-sagging cloth diaper industry saved by full dumps.

and the World War II favorite:

- American pushes bottle up Germans.

LEXICAL AMBIGUITY

The simplest type of ambiguity is **lexical ambiguity**, where a word has more than one meaning. For example, the adjective "hot" can mean warm or spicy or electrified or radioactive or vehement or sexy or popular or stolen. Lexical ambiguity can cut across categories: "back" is an adverb in "go back," an adjective in "back door," a noun in "the back of the room," and a verb in "back up your files."

SYNTACTIC AMBIGUITY

Syntactic ambiguity (also known as **structural ambiguity**) can occur with or without lexical ambiguity. For example, the string "I smelled a wumpus in 2,2" has two parses: one where the propositional phrase modifies the noun, and one where it modifies the verb. The syntactic ambiguity leads to a **semantic ambiguity**, because one parse means the wumpus is in 2,2 and the other means that a stench is in 2,2. In this case, getting the wrong interpretation could be a deadly mistake. The lexical ambiguities of the previous paragraph also lead to semantic ambiguities. On the other hand, semantic ambiguity can occur even in phrases with no lexical or syntactic ambiguity. For example, the noun phrase "cat person" can be someone who likes felines or the lead of the movie *Attack of the Cat People*. A "coast road" can be a road that follows the coast, or a road that leads to the coast.

REFERENTIAL AMBIGUITY

One pervasive form of semantic ambiguity is **referential ambiguity**. Anaphoric expressions such as "it" can refer to almost anything. Referential ambiguity occurs because natural languages consist almost entirely of words for categories, not for individual objects. There is no word for *the-apple-I-had-for-lunch-today*, just categories like *apple*.

PRAGMATIC AMBIGUITY

One type of **pragmatic ambiguity** occurs when the speaker and hearer disagree on what the current situation is. If the speaker says "I'll meet you next Friday" thinking that they're talking about the 17th, and the hearer thinks that they are talking about the 24th, then there is miscommunication. The example on page 659 about "Now!" also involves pragmatic ambiguity.

LOCAL AMBIGUITY

Sometimes a phrase or sentence has **local ambiguity**, where a substring can be parsed several ways, but only one of those ways fits into the larger context of the whole string. For example, in the C programming language, the string `*c` means "pointer to c" when it appears in the declaration `char *c;` but it means "multiply by c" when it appears in the expression `2 *c`. In English, "the radio broadcasts" is a noun phrase in "the radio broadcasts inform" and a noun phrase followed by a verb in "the radio broadcasts information." It is possible for a phrase or sentence to be syntactically ambiguous but semantically unambiguous. For example, " S_1 and S_2 and S_3 " has two different parses. But they both have the same meaning (at least in £2), because conjunction is associative.

VAGUE

Natural languages are also **vague**. When we say, "It's hot outside," it says something about the temperature, but it is open to a wide range of interpretation because "hot" is a vague term. To some it might mean the temperature is above 75°F, and to others it might mean 90°F.

Finally, there can be ambiguity about what speech act has been performed. A hearer who says, "yes" when asked, "Do you know what time it is?" has successfully interpreted the sentence as if it were question, but most likely it was actually intended as a request for information.

Disambiguation



As we said before, *disambiguation is a question of diagnosis*. The hearer maintains a model of the world and, upon hearing a new speech act, adds the possible interpretations of the speech act to the model as hypotheses. The uncertain reasoning techniques of Part V can then be used to decide which interpretation is best. To do this well, the model must include a lot of information about the world. For example, to correctly resolve the syntactic ambiguity in "Chris saw the Grand Canyon flying to New York," one needs to know that it is more likely that Chris is doing the flying than that the Grand Canyon is. Similarly, to understand "Donald keeps his money in the bank," it helps to know that money is kept in savings institutions more often than in snowbanks.

One also needs a good model of the beliefs of speaker and hearer, in order to decide what the speaker will bother to say. For example, the normal interpretation of the statement "I am not a crook" is that the speaker is not a criminal. This is true even though an alternative interpretation—that the speaker is not a hooked shepherd's staff—has a higher probability of being true. Similarly, "Howard doesn't keep his money in the bank" probably refers to saving institutions, because it would not be worth remarking that he did not keep his money in a snowbank. In general, disambiguation requires the combination of four models:

1. **The world model:** the probability that a fact occurs in the world.
2. **The mental model:** the probability that the speaker forms the intention of communicating this fact to the hearer, given that it occurs.⁹ (This combines models of what the speaker believes, what the speaker believes the hearer believes, and so on.)
3. **The language model:** the probability that a certain string of words will be chosen, given that the speaker has the intention of communicating a certain fact.
4. **The acoustic model:** the probability that a particular sequence of sounds will be generated, given that the speaker has chosen a given string of words. This will be taken up when we consider perception in Chapter 24.

The final reason why it is hard to pick the right interpretation is that there may be several right ones. Jokes rely on the fact that the hearer will entertain two interpretations simultaneously. In "She criticized his apartment so he knocked her flat," we have three lexical and one syntactic ambiguity. But the joke would be lost on a hearer who simply accepted the best interpretation and ignored the other. Poetry, advertising, political rhetoric, and murder mysteries are other genres that make use of deliberate ambiguity. Most language understanding programs ignore this possibility, just as many diagnosis systems ignore the possibility of multiple causes.

Context-free grammars do not provide a very useful language model (even when augmentations are included). The problem is that the grammar does not say which strings are more probable than others—it simply divides the strings into two classes: grammatical and ungrammatical.

⁹ We should also consider the possibility that the speaker intends to convey some information given that it did *not* occur, that is, that the speaker is mistaken or lying.

The simplest way to provide a probability distribution is to use a **probabilistic context-free grammar** or PCFG.¹⁰ In the PCFG language model, each rewrite rule has a probability associated with it, such that the sum for all rules with the same left-hand side is 1—for example,

$$\begin{aligned} S \rightarrow NP\ VP & \quad (0.9) \\ S \rightarrow S\ Conjunction\ S & \quad (0.1) \end{aligned}$$

In the PCFG model, the probability of a string, $P(\text{words})$, is just the sum of the probabilities of its parse trees—one such tree for an unambiguous string, no trees for an ungrammatical string, and several trees for an ambiguous string. The probability of a given tree is the product of the probabilities of all the rules that make up the nodes of the tree.

The problem with PCFGs is that they are context-free. That means that the difference between $P(\text{"I ate a banana"})$ and $P(\text{"I ate a bandana"})$ depends only on $P(\text{"banana"})$ versus $P(\text{"bandana"})$, and not on the relation between "ate" and the respective nouns. To get at that kind of relationship, we will need some kind of context-sensitive model. Other probabilistic language models that include context sensitivity have been proposed (see the Historical Notes section). The problem of combining the four models into one is taken up when we discuss speech recognition in Section 24.7.

22.9 A COMMUNICATING AGENT

We have now seen how to go all the way from strings to meanings using syntactic and semantic analysis and disambiguation. The final step is to show how this fits in to an agent that can communicate. We start with the simple wumpus world robot slave described on page 662.

The first step in building the communicating agent is to extend the grammar to accept commands such as "Go east." So far, the language \mathcal{E}_2 has only one type of speech act: the statement. We will extend it with commands and acknowledgments to yield the language \mathcal{E}_3 .

The new words and grammar rules are not complicated. A command can be formed from a VP , where the subject is implicitly the hearer. For example, "Go to 2 2" is a command, and it already is a VP according to the \mathcal{E}_2 grammar. The semantics of the command is derived by applying the semantics of the VP to the object *Hearer*. Now that we have several kinds of speech acts, we will identify the kind (i.e., command or statement) as part of the quasi-logical form. Here are the rules for commands and statements:

$$\begin{aligned} S(Command(rel(Hearer))) & \rightarrow VP(rel) \\ S(Statement(rel(obj))) & \rightarrow NP(obj)\ VP(rel) \end{aligned}$$

So the quasi-logical form for "Go to 2 2" is:¹¹

$$Command(\exists e\ e \in Go(Hearer,[2,2]))$$

¹⁰ PCFGs are also known as stochastic context-free grammars or SCFGs.

¹¹ Note that the quasi-logical form for a command does not include the time of the event (e.g., *During(Now, e)*). That is because commands are tenseless. We can't tell that by looking at commands with "go," but consider that the correct form of a command is "[you] be good," (using the untensed form "be") not "[you] are good."

Acknowledgments are even simpler—they consist of a single word: "yes" or "OK" to positively acknowledge, and "no" to negatively acknowledge. Here are the rules:

$$S(\text{Acknowledge}(\text{sem})) \rightarrow \text{Ack}(\text{sem})$$

$$\text{Ack}(\text{True}) \rightarrow \text{yes}$$

$$\text{Ack}(\text{True}) \rightarrow \text{OK}$$

$$\text{Ack}(\text{False}) \rightarrow \text{no}$$

It is up to the master (or whoever hears an acknowledgment) to do pragmatic interpretation to realize that "OK," which gets the interpretation $\text{Ack}(\text{True})$, means that the agent has agreed to follow out a command (usually the most recent command). The agent program for the robot slave was shown in Figure 22.4 (page 663). But in a sense we do not need a new agent program; all we need to do is use an existing agent (such as a logical planning agent) and give it the goals of understanding the input and responding to it properly.

22.10 SUMMARY

We have seen why it is useful to communicate, and how language can be interpreted by agents in a situation. Natural language processing is difficult for three reasons. First, one has to have a lot of specific knowledge about the words and grammar rules of the language. Second, one must be able to integrate this knowledge with other knowledge about the world. Third, language involves an additional complication that we have not dealt with so far: that there are other agents in the world who have their own beliefs, goals, and plans. This chapter makes the following points in addressing these difficulties:

- Agents send signals to each other to achieve certain purposes: to inform, to warn, to elicit help, to share knowledge, or to promise something. Sending a signal in this way is called a **speech act**. Ultimately, all speech acts are an attempt to get another agent to believe something or do something.
- All animals use some conventional signs to communicate, but humans use language in a more sophisticated way that enables them to communicate much more.
- Formal language theory and **phrase structure** grammars (and in particular, **context-free** grammar) are useful tools for dealing with some aspects of natural language.
- Communication involves three steps by the speaker: the intention to convey an idea, the mental generation of words, and their physical synthesis. The hearer then has four steps: perception, analysis, disambiguation, and incorporation of the meaning.
- The **encoded message** model of communication states that a speaker encodes a representation of a proposition into language, and the hearer then decodes the message to uncover the proposition. The **situated language** model states that the meaning of a message is a function of both the message and the situation in which it occurs.
- It is convenient to **augment** a grammar to handle such problems as subject/verb agreement, pronoun case, and semantics. Definite Clause Grammar (DCG) is an extension of BNF that allows for augmentations.

- There are many algorithms for parsing strings. We showed a simple one. It is also possible to feed DCG rules directly to a logic programming system or theorem prover.
- **Pragmatic** interpretation takes the current situation into account to determine the effect of an utterance in context.
- **Disambiguation** is the process of deciding which of the possible interpretations is the one that the speaker intended to convey.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The idea of language as action stems from twentieth-century linguistically oriented philosophy (Wittgenstein, 1953; Grice, 1957; Austin, 1962) and particularly from the book *Speech Acts* (Searle, 1969). A precursor to the idea of speech acts was Protagoras's distinction of four types of sentence: prayer, question, answer, and injunction. Hobbs *et al.* (1987) describe a more practical application of the situated model.

Like semantic networks, context-free grammars (also known as phrase structure grammars) are a reinvention of a technique first used by the ancient Indian grammarians (especially Panini, c. 350 B.C.) studying Shastric Sanskrit (Ingerman, 1967). In modern times, they were reinvented by Noam Chomsky for the analysis of English syntax (Chomsky, 1956) and independently by John Backus for the analysis of Algol-60 syntax. Naur (Naur, 1963) extended Backus's notation, and is now credited with the "N" in BNF, which originally stood for "Backus Normal Form." Knuth (1968) defined a kind of augmented grammar called **attribute grammar**.

There have been many attempts to write formal grammars of natural languages, both in "pure" linguistics and in computational linguistics. The Linguistic String Project at New York University (Sager, 1981) produced a large grammar for the machine parsing of English, using essentially context-free rewrite rules with some restrictions based on subcategorization. A good example of a modern system using unification grammar is the Core Language Engine (Alshawi, 1992). There are several comprehensive but informal grammars of English (Quirk *et al.*, 1985; Huddleston, 1988). Good textbooks on linguistics include Baker (1989) and Chierchia and McConnell-Ginet (1990). McCawley's (1993) text concentrates on logic for linguists. Definite clause grammars were introduced by Colmerauer (1975) and developed and popularized by Pereira and Warren (1980).

Formal semantic interpretation of natural languages originates within philosophy and formal logic and is especially closely related to Alfred Tarski's (1935) work on the semantics of formal languages. Bar-Hillel was the first to consider the problems of pragmatics and propose that they could be handled by formal logic. For example, he introduced C.S. Peirce's (1902) term *indexical* into linguistics (Bar-Hillel, 1954). Richard Montague's essay "English as a formal language" (1970) is a kind of manifesto for the logical analysis of language, but the book by Dowty, Wall, and Peters (1991) and the article by Lewis (1972) are more readable. A complete collection of Montague's contributions has been edited by Thomason (1974). In artificial intelligence, the work of McAllester and Givan (1992) continues the Montagovian tradition, adding many new technical insights.

The idea of an intermediate or quasi-logical form to handle problems such as quantifier scoping goes back to Woods (1978), and is present in many recent systems (Alshawi, 1992; Hwang and Schubert, 1993). Van Lehn (1978) gives a survey of human preferences for quantifier scope disambiguation.

Linguists have dozens of different formalisms; hundreds if you count all the minor modifications and notational variants. We have stuck to a single approach—definite clause grammar with BNF-style notation—but the history of the others is interesting. Sells (1985) offers a good comparison of some current formalisms, but Gerald Gazdar's (1989) analysis is more succinct:

Here is the history of linguistics in one sentence: once upon a time linguists (i.e., syntacticians) used augmented phrase structure grammars, then they went over to transformational grammars, and then some of them started using augmented phrase structure grammars again, *<space for moral>*. Whilst we are in this careful scholarly mode, let us do the same service for computational linguistics: once upon a time computational linguistics (i.e., builders of parsers) used augmented phrase structure grammars, then they went over to augmented transition networks, and then many of them started using augmented phrase structure grammars again, *<space for moral>*.

We can characterize the different formalisms according to four dichotomies: transformational versus monostratal, unification versus assignment, lexical versus grammatical, and syntactic categories versus semantic categories.

TRANSFORMATIONAL
GRAMMAR

DEEP STRUCTURE

SURFACE
STRUCTURE

AUGMENTED
TRANSITION
NETWORK

The dominant formalism for the quarter century starting in 1956 was **transformational grammar** (Chomsky, 1957; Chomsky, 1965). In this approach, the commonality in meaning between sentences like "Man bites dog" and "Dog is bitten by man" is captured by a context-free grammar that generates proto-sentences in a canonical form called the **deep structure**. "Man bites dog" and "Dog is bitten by man" would have the same deep structure, but different **surface structure**. A separate set of rules called **transformations** map between deep and surface structure. The fact that there are two distinct levels of analysis and two sets of rules makes transformational grammar a **multistratal** theory. Computational linguists have turned away from transformational grammar, because it is difficult to write parsers that can invert the transformations to recover the deep structure.

The **augmented transition network** (ATN) grammars mentioned in the Gazdar quote were invented as a way to go beyond context-free grammar while maintaining a monostratal approach that is computationally tractable. They were invented by Thorne (1968) but are mostly associated with the work of Woods (1970). The rules in an ATN grammar are represented as a directed graph, but one can easily transliterate between transition networks and context-free grammars, so choosing one over the other is mostly a matter of taste. The other big difference is that DCGs are augmented with **unification** assertions, whereas ATNs are augmented with **assignment** statements. This makes DCGs closer to standard first-order logic, but more importantly, it allows a DCG grammar to be processed by a variety of algorithms. Any order of application of the rules will arrive at the same answer, because unification is commutative. Assignment, of course, is not commutative. GPSG or Generalized Phrase Structure Grammar (Gazdar *et al.*, 1985) and HPSG or Head-driven Phrase Structure Grammar (Pollard and Sag, 1994) are two important examples of unification-based grammars. Shieber (1986) surveys them and others.

Since the mid-1980s, there has also been a trend toward putting more information in the lexicon and less in the grammar. For example, rather than having a grammar rule to transform

active sentences into passive, many modern grammars place the burden of passives on the lexicon. The grammar would have one or more rules saying a verb phrase can be a verb optionally preceded by an auxiliary verb and followed by complements. The lexical entry for "bitten" would say that it is preceded by a form of the auxiliary verb "be" and followed by a prepositional phrase with the preposition "by."

In the 1970s it was felt that putting this kind of information in the lexicon would be missing an important generality—that most transitive verbs have passive forms.¹² The current view is that if we can account for the way the passives of new verbs are learned, then we have not lost any generalities. Putting the information in the lexicon rather than the grammar is just a kind of compilation—it can make the parser's job easier at run time. LFG or lexical-functional grammar (Bresnan, 1982) was the first major grammar of English and formalism to be highly lexicalized. If we carry lexicalization to an extreme, we end up with **categorial grammar**, in which there can be as few as two grammar rules, or **dependency grammar** (Melčuk and Polguere, 1988), in which there are no phrases, only words. TAG or Tree-Adjoining Grammar (Joshi, 1985) is not strictly lexical, but it is gaining popularity in its lexicalized form (Schabes *et al.*, 1988).

A major barrier to the widespread use of natural language processing is the difficulty of tuning an NLP system to perform well in a new domain, and the amount of specialized training (in linguistics and computer science) needed to do the tuning. One way to lower this barrier is to throw out the specialized terminology and methodology of linguistics and base the system's grammar more directly on the problem domain. This is achieved by replacing abstract syntactic categories with domain-specific semantic categories. Such a grammar is called a **semantic grammar**. For example, an interface to an airline reservation system could have categories like *Location* and *Fly-To* instead of *NP* and *VP*. See Birnbaum and Selfridge (1981) for an implementation of a system based on semantic grammars.

There are two main drawbacks to semantic grammars. First, they are specific to a particular domain. Very little of the work that goes into building a system can be transferred to a different domain. Second, they make it hard to add syntactic generalizations. Handling constructions such as passive sentences means adding not just one new rule, but one rule for each verb-like category. Getting it right is time-consuming and error-prone. Semantic grammars can be used to get a small application working quickly in a limited domain, but they do not scale up well.

The other approach to knowledge acquisition for NLP is to use machine learning. Gold (1967) set the groundwork for this field, and Fu and Booth (1986a; 1986b) give a tutorial of recent work. Stolcke (1993) gives an algorithm for learning probabilistic context-free grammars, and Black *et al.* (1992) and Magerman (1993) show how to learn more complex grammars.

Research on language learning by humans is surveyed by Wanner and Gleitman (1982) and by Bloom (1994). Pinker (1989) gives his take on the field. A variety of machine learning experiments have tried to duplicate human language learning (Clark, 1992; Siskind, 1994).

Disambiguation has always been one of the hardest parts of NLP. In part, this is because of a lack of help from other fields. Linguistics considers disambiguation to be largely outside its domain, and literary criticism (Empson, 1953; Hobbs, 1990) is ambiguous about whether

¹² It is now known that passivity is a feature of sentences, not verbs. For example, "This bed was slept in by George Washington" is a good sentence, but "The stars were slept under by Fred" is not (even though the two corresponding active sentences are perfectly good).

ambiguity is something to be resolved or to be cherished. Some of the earliest work on disambiguation was Wilks' (1975) theory of **preference semantics**, which tried to find interpretations that minimize the number of semantic anomalies. Hirst (1987) describes a system with similar aims that is closer to the compositional semantics described in this chapter. Some problems with multiple interpretations are addressed by Norvig (1988).

Probabilistic techniques for disambiguation have been predominant in recent years, partly because of the availability of large corpora of text from which to gather statistics, and partly because the field is evolving towards a more scientific methodology. Research involving large corpora of text is described in the special issue of *Computational Linguistics* (Volume 19, Numbers 1 and 2, 1993) and the book by Garside *et al.* (1987). The statistical approach to language is covered in a book by Charniak (1993). This subfield started when NLP researchers noticed the success of probabilistic models in information retrieval (Salton, 1989) and speech recognition (Rabiner, 1990). This lead to the development of probabilistic models for word sense disambiguation (Yarowsky, 1992; Resnik, 1993) and eventually to the full parsing task, as in the work by Church (1988) and by Chitrao and Grishman (1990). Some recent work casts the disambiguation problem as belief network evaluation (Charniak and Goldman, 1992; Goldman and Charniak, 1992; Wu, 1993).

The Association for Computational Linguistics (ACL) holds regular conferences; much current research on natural language processing is published in their proceedings, and in the ACL's journal *Computational Linguistics. Readings in Natural Language Processing* (Grosz *et al.*, 1986) is an anthology containing many important papers in the field. The leading textbook is *Natural Language Understanding* (Alien, 1995). Pereira and Sheiber (1987) and Covington (1994) offer concise overviews based on implementations in Prolog. The *Encyclopedia of AI* has many useful articles on the field; see especially "Computational Linguistics" and "Natural Language Understanding."

EXERCISES

22.1 Outline the major differences between Pascal (or any other computer language with which you are familiar) and English, and the "understanding" problem in each case. Think about such things as grammar, syntax, semantics, pragmatics, compositionality, context-dependence, lexical ambiguity, syntactic ambiguity, reference-finding (including pronouns), background knowledge, and what it means to "understand" in the first place.

22.2 Which of the following are reasons for introducing a quasi-logical form?

- a. To make it easier to write simple compositional grammar rules.
- b. To extend the expressiveness of the semantic representation language.
- c. To be able to represent quantifier scoping ambiguities (among others) in a succinct form.
- d. To make it easier to do semantic disambiguation.

22.3 Determine what semantic interpretation would be given to the following sentences by the grammar in this chapter:

- a. It is a wumpus.
- b. The wumpus is dead.
- c. The wumpus is in 2,2.

Would it be a good idea to have the semantic interpretation for "It is a wumpus" be simply $\exists x \text{ Wumpus}(x)$? Consider alternative sentences such as "It was a wumpus."

22.4 Augment the grammar from this chapter so that it handles the following:

- a. Pronoun case.
- b. Subject/verb agreement.
- c. Article/noun agreement: "agents" is an *NP* but "agent" is not. In general, only plural nouns can appear without an article.

22.5 This exercise concerns grammars for very simple languages.

- a. Write a context-free grammar for the language $a^n b^n$.
- b. Write a context-free grammar for the palindrome language: the set of all strings whose second half is the reverse of the first half.
- c. Write a context-sensitive grammar for the language $a^n b^n c^n$.
- d. Write a context-sensitive grammar for the duplicate language: the set of all strings whose second half is the same as the first half.



22.6 This exercise continues the example of Section 22.9 by making the slave more intelligent. On each turn, the slave describes its percepts as before, but it also says where it is (e.g., "I am in 1,1") and reports any relevant facts it has deduced about the neighboring squares (e.g., "There is a pit in 1,2" or "2,1 is safe"). You need not do any fancy language generation, but you do have to address the intention problem: deciding which facts are worth mentioning. In addition, you should give your slave a sense of self-preservation. If it is commanded to enter a deadly square, it should politely refuse. If commanded to enter an unsafe square, it can ask for confirmation, but if commanded again, it should obey. Run this slave in the wumpus environment a few times. How much easier is it to work with this slave than the simple one from Section 22.9?

22.7 Consider the sentence "Someone walked slowly to the supermarket" and the following set of context-free rewrite rules which give the grammatical categories of the words of the sentence:

$$\begin{array}{ll} \text{Pronoun} \rightarrow \text{someone} & \text{V} \rightarrow \text{walked} \\ \text{Adv} \rightarrow \text{slowly} & \text{Prep} \rightarrow \text{to} \\ \text{Det} \rightarrow \text{the} & \text{Noun} \rightarrow \text{supermarket} \end{array}$$

Which of the following three sets of rewrite rules, when added to the preceding rules, yield context-free grammars that can generate the above sentence?

(A):	(B):	(C):
$S \rightarrow NP VP$	$S \rightarrow NP VP$	$S \rightarrow NP VP$
$NP \rightarrow Pronoun$	$NP \rightarrow Pronoun$	$NP \rightarrow Pronoun$
$NP \rightarrow Det Noun$	$NP \rightarrow Noun$	$NP \rightarrow Det NP$
$VP \rightarrow VP PP$	$NP \rightarrow DetNP$	$VP \rightarrow VAdv$
$VP \rightarrow VP Adv Adv$	$VP \rightarrow V Vmod$	$Adv \rightarrow AdvAdv$
$VP \rightarrow V$	$Vmod \rightarrow Adv Vmod$	$Adv \rightarrow PP$
$PP \rightarrow Prep NP$	$Vmod \rightarrow Adv$	$PP \rightarrow Prep NP$
$NP \rightarrow Noun$	$Adv \rightarrow PP$	$NP \rightarrow Noun$
	$PP \rightarrow PrepNP$	

Write down at least one other English sentence generated by Grammar (B). It should be significantly different from the above sentence, and should be at least six words long. Do not use any of the words from the preceding sentence; instead, add grammatical rules of your own, for instance, $Noun \rightarrow \text{bottle}$. Show the parse tree for your sentence.

22.8 This exercise concerns a language we call *Buffalo*" which is very much like English except the only word in its lexicon is *buffalo*. (The language is due to Barton, Berwick, and Ristad.) Here are two sentences from the language:

- Buffalo buffalo buffalo Buffalo buffalo.
- Buffalo Buffalo buffalo buffalo Buffalo buffalo.

In case you don't believe these are sentences, here are two English sentences with corresponding syntactic structure:

- Dallas cattle bewilder Denver cattle.
- Chefs London critics admire cook French food.

Write a grammar for *Buffalo*". The lexical categories are adjective, noun, and (transitive) verb, and there should be one grammar rule for sentence, one for verb phrase, and three rules for noun phrase: raw noun, adjective modifier, and reduced relative clause (i.e., a relative clause without the word "that"). Tabulate the number of possible parses for *Buffalo*" for n up to 10.

23

PRACTICAL NATURAL LANGUAGE PROCESSING

In which we see how to scale up from toy domains like the wumpus world to practical systems that perform useful tasks with language.

In Chapter 22, we saw that agents can gain by communicating with each other. We also saw some techniques for interpreting sentences from simple subsets of English. In this chapter, we show how far beyond the wumpus world one can go by elaborating on those techniques. The topics covered are as follows:

- 0 **Practical applications:** tasks where natural language has proved useful.
 - ◇ **Discourse processing:** the problem of handling more than one sentence.
 - 0 **Efficient parsing:** algorithms for parsing and interpreting sentences quickly.
 - 0 **Scaling up the lexicon:** dealing with unusual and even unknown words.
 - ◇ **Scaling up the grammar:** dealing with complicated syntax.
 - ◇ **Semantic interpretation:** some problems that make semantic interpretation more than just a matter of composing simple functions.
 - 0 **Disambiguation:** how to choose the right interpretation.

23.1 PRACTICAL APPLICATIONS

We start by surveying successful systems that put natural language to practical use. The successful systems share two properties: they are focused on a particular *domain* rather than allowing discussion of any topic, and they are focused on a particular *task* rather than attempting to understand language completely. We will look at five tasks.

Machine translation

In the early 1960s, there was great hope that computers would be able to translate from one natural language to another, just as Turing's project "translated" coded messages into intelligible

German. But by 1966, it became clear that translation requires an understanding of the meaning of the message (and hence detailed knowledge about the world), whereas code breaking depends only on the syntactic properties of the messages.

Although there has been no fundamental breakthrough in machine translation, there has been real progress, to the point that there are now dozens of machine translation systems in everyday use that save money over fully manual techniques. One of the most successful is the TAUM-METEO system, developed by the University of Montreal, which translates weather reports from English to French. It works because the language used in these government weather reports is highly stylized and regular.

In more open domains, the results are less impressive. A representative system is SPANAM (Vasconcellos and León, 1985), which can translate a Spanish passage into English of this quality:

The extension of the coverage of the health services to the underserved or not served population of the countries of the region was the central goal of the Ten-Year Plan and probably that of greater scope and transcendence. Almost all the countries formulated the purpose of extending the coverage although could be appreciated a diversity of approaches for its attack, which is understandable in view of the different national policies that had acted in the configuration of the health systems of each one of the countries.

This is mostly understandable, but not always grammatical and rarely fluent. Standing on its own, unrestricted machine translation is still inadequate. But when a human translator is given a text like this as an initial guideline, the human is able to work two to four times faster. Sometimes a monolingual human can post-edit the output without having to read the original. This saves money because such editors can be paid less than bilingual translators.

Another possibility is to invest the human effort on pre-editing the original document. If the original document can be made to conform to a restricted subset of English (or whatever the original language is), then it can sometimes be translated without the need for post-editing. This approach is particularly cost-effective when there is a need to translate one document into many languages, as is the case for legal documents in the European Community, or for companies that sell the same product internationally. Restricted languages are sometimes called "Caterpillar English," because Caterpillar was the first firm to try writing their manuals in this form. The first really successful use of this approach was made by Xerox. They defined a language for their maintenance manuals that was simple enough that it could be translated by the SYSTRAN system into all the languages Xerox deals with. As an added benefit, the original English manuals became clearer as well.

There is a substantial start-up cost to any machine translation effort. To achieve broad coverage, translation systems have lexicons of 20,000 to 100,000 words and grammars of 100 to 10,000 rules, the numbers varying greatly depending on the choice of formalism.

Translation is difficult because, in the general case, it requires in-depth understanding of the text, and that requires in-depth understanding of the situation that is being communicated. This is true even for very simple texts—even "texts" of one word. Consider the word "Open" on the door of a store.¹ It communicates the idea that the store is accepting customers at the moment. Now consider the same word "Open" on a large banner outside a newly constructed store. It means that the store is now in daily operation, but readers of this sign would not feel

¹ This example is due to Martin Kay.

misled if the store closed at night without removing the banner. The two signs use the identical word to convey different meanings. In some other languages, the same word or phrase would be used in both cases, but in German, the sign on the door would be "Offen" while the banner would read "Neu Eröffnet."

The problem is that different languages categorize the world differently. A majority of the situations that are covered by the English word "open" are also covered by the German word "offen," but the boundaries of the category differ across languages. In English, we extend the basic meaning of "open" to cover open markets, open questions, and open job offerings. In German, the extensions are different. Job offerings are "freie," not open, but the concepts of loose ice, private firms, and blank checks all use a form of "offen."

To do translation well, a translator (human or machine) must read the original text, understand the situation to which it is referring, and find a corresponding text in the target language that does a good job of describing the same or a similar situation. Often this involves a choice. For example, the English word "you" can be translated into French as either the formal "vous" or the informal "tu." There is just no way that one can refer to the concept of "you" in French without also making a choice of formal or informal. Translators (both machine and human) sometimes find it difficult to make this choice.

Database access

The first major success for natural language processing (NLP) was in the area of database access. Circa 1970, there were many databases on mainframe computers, but they could be accessed only by writing complicated programs in obscure programming languages. The staff in charge of the mainframes could not keep up with all the requests of users who needed to get at this data, and the users understandably did not want to learn how to program their own requests. Natural language interfaces provided a solution to this dilemma.

The first such interface was the LUNAR system, a prototype built by William Woods (1973) and his team for the NASA Manned Spacecraft Center. It enabled a geologist to ask questions about the chemical analysis data of lunar rock and soil samples brought back by the Apollo missions. The system was not put into real operational use, but in one test it successfully answered 78% of queries such as

What is the average modal plagioclase concentration for lunar samples that contain rubidium?

Fernando Pereira's CHAT system (Pereira, 1983) is at a similar level of complexity. It generates the following answers to questions about a geographical database:

Q: Which countries are bordered by two seas?

A: Egypt, Iran, Israel, Saudi Arabia and Turkey

Q: What are the countries from which a river flows into the Black sea?

A: Romania, Soviet Union

Q: What is the total area of countries south of the equator and not in Australasia?

A: 10,228,000 square miles

Q: What is the ocean that borders African countries and that borders Asian countries?

A: Indian Ocean

The advantages of systems like this are obvious. The disadvantage is that the user never knows which wordings of a query will succeed and which are outside the system's competence. For example, CHAT handles "south of the equator" and "with latitude less than zero," but not "in the southern hemisphere." There is no principled reason why this last paraphrase should not work; it just happens that "hemisphere" is not in the dictionary (nor is this sense of "in"). Similarly, the final sample question could not be phrased as "What ocean borders both African countries and Asian?" because the grammar does not allow that kind of conjunction.

Over the last decade, some commercial systems have built up large enough grammars and lexicons to handle a fairly wide variety of inputs. The main challenge for current systems is to follow the context of an interaction. The user should be able to ask a series of questions where some of them implicitly refer to earlier questions or answers:

What countries are north of the equator?

How about south?

Show only the ones outside Australasia.

What is their total area?

Some systems (e.g., TEAM (Grosz *et al.*, 1987)) handle problems like this to a limited degree. We return to the problem in Section 23.6.

In the 1990s, companies such as Natural Language Inc. and Symantec are still selling database access tools that use natural language, but customers are less likely to make their buying decisions based on the strength of the natural language component than on the graphical user interface or the degree of integration of the database with spreadsheets and word processing. Natural language is not always the most natural way to communicate: sometimes it is easier to point and click with a mouse to express an idea (e.g., "sum *that* column of the spreadsheet").

The emphasis in practical NLP has now shifted away from database access to the broad field of **text interpretation**. In part, this is a reflection of a change in the computer industry. In the early 1980s, most online information was stored in databases or spreadsheets. Now the majority of online information is text: email, news, journal articles, reports, books, encyclopedias. Most computer users find there is too much information available, and not enough time to sort through it. Text interpretation programs help to retrieve, categorize, filter, and extract information from text. Text interpretation systems can be split into three types: information retrieval, text categorization, and data extraction.

Information retrieval

In information retrieval (IR), the task is to choose from a set of documents the ones that are relevant to a query. Sometimes a document is represented by a surrogate, such as the title and a list of keywords and/or an abstract. Now that so much text is online, it is more common to use the full text, possibly subdivided into sections that each serve as a separate document for retrieval purposes. The query is normally a list of words typed by the user. In early information retrieval systems, the query was a Boolean combination of keywords. For example, the query "(natural and language) or (computational and linguistics)" would be a reasonable query to find documents related to this chapter. However, users found it difficult to get good results with Boolean queries. When a query finds no documents, for example, it is not clear how to relax the query to find

VECTOR-SPACE

some. Changing an "and" to an "or" is one possibility; adding another disjunction is another, but users found there were too many possibilities and not enough guidance.

Most modern IR systems have switched from the Boolean model to a **vector-space** model, in which every list of words (both document and query) is treated as a vector in n -dimensional space, where n is the number of distinct tokens in the document collection. In this model, the query would simply be "natural language computational linguistics," which would be treated as a vector with the value 1 for these four words (or **terms**, as they are called in IR) and the value 0 for all the other terms. Finding documents is then a matter of comparing this vector against a collection of other vectors and reporting the ones that are close. The vector model is more flexible than the Boolean model because the documents can be ranked by their distance to the query, and the closest ones can be reported first.

There are many variations on this model. Some systems are equipped with morphological analyzers that match "linguistic computation" with "computational linguistics." Some allow the query to state that two words must appear near each other to count as a match, and others use a thesaurus to automatically augment the words in the query with their synonyms. Only the most naive systems count all the terms in the vectors equally. Most systems ignore common words like "the" and "a," and many systems weight each term differently. A good way to do this is to give a term a larger weight if it is a good discriminator: if it appears in a small number of documents rather than in many of them.

This model of information retrieval is almost entirely at the word level. It admits a minuscule amount of syntax in that words can be required to be near each other, and allows a similarly tiny role for semantic classes in the form of synonym lists. You might think that IR would perform much better if it used some more sophisticated natural language processing techniques. Many people have thought just that, but surprisingly, none has been able to show a significant improvement on a wide range of IR tasks. It is possible to tune NLP techniques to a particular subject domain, but nobody has been able to successfully apply NLP to an unrestricted range of texts.

The moral is that most of the information in a text is contained in the words. The IR approach does a good job of applying statistical techniques to capture most of this information. It is as if we took all the words in a document, sorted them alphabetically, and then very carefully compared that list to another sorted list. While the sort loses a lot of information about the original document, it often maintains enough to decide if two sorted lists are on similar topics. In contrast, the NLP technology we have today can sometimes pick out additional information—disambiguating words and determining the relations between phrases—but it often fails to recover anything at all. We are just beginning to see hybrid IR/NLP systems that combine the two approaches.

Text categorization

NLP techniques have proven successful in a related task: sorting text into fixed topic categories. There are several commercial services that provide access to news wire stories in this manner. A subscriber can ask for all the news on a particular industry, company, or geographic area, for example. The providers of these services have traditionally used human experts to assign

The advantages of systems like this are obvious. The disadvantage is that the user never knows which wordings of a query will succeed and which are outside the system's competence. For example, CHAT handles "south of the equator" and "with latitude less than zero," but not "in the southern hemisphere." There is no principled reason why this last paraphrase should not work; it just happens that "hemisphere" is not in the dictionary (nor is this sense of "in"). Similarly, the final sample question could not be phrased as "What ocean borders both African countries and Asian?" because the grammar does not allow that kind of conjunction.

Over the last decade, some commercial systems have built up large enough grammars and lexicons to handle a fairly wide variety of inputs. The main challenge for current systems is to follow the context of an interaction. The user should be able to ask a series of questions where some of them implicitly refer to earlier questions or answers:

What countries are north of the equator?

How about south?

Show only the ones outside Australasia.

What is their total area?

Some systems (e.g., TEAM (Grosz *et al.*, 1987)) handle problems like this to a limited degree. We return to the problem in Section 23.6.

In the 1990s, companies such as Natural Language Inc. and Symantec are still selling database access tools that use natural language, but customers are less likely to make their buying decisions based on the strength of the natural language component than on the graphical user interface or the degree of integration of the database with spreadsheets and word processing. Natural language is not always the most natural way to communicate: sometimes it is easier to point and click with a mouse to express an idea (e.g., "sum *that* column of the spreadsheet").

The emphasis in practical NLP has now shifted away from database access to the broad field of **text interpretation**. In part, this is a reflection of a change in the computer industry. In the early 1980s, most online information was stored in databases or spreadsheets. Now the majority of online information is text: email, news, journal articles, reports, books, encyclopedias. Most computer users find there is too much information available, and not enough time to sort through it. Text interpretation programs help to retrieve, categorize, filter, and extract information from text. Text interpretation systems can be split into three types: information retrieval, text categorization, and data extraction.

TEXT INTERPRETATION

Information retrieval

In information retrieval (IR), the task is to choose from a set of documents the ones that are relevant to a query. Sometimes a document is represented by a surrogate, such as the title and a list of keywords and/or an abstract. Now that so much text is online, it is more common to use the full text, possibly subdivided into sections that each serve as a separate document for retrieval purposes. The query is normally a list of words typed by the user. In early information retrieval systems, the query was a Boolean combination of keywords. For example, the query "(natural and language) or (computational and linguistics)" would be a reasonable query to find documents related to this chapter. However, users found it difficult to get good results with Boolean queries. When a query finds no documents, for example, it is not clear how to relax the query to find

some. Changing an "and" to an "or" is one possibility; adding another disjunction is another, but users found there were too many possibilities and not enough guidance.

Most modern IR systems have switched from the Boolean model to a **vector-space** model, in which every list of words (both document and query) is treated as a vector in n -dimensional space, where n is the number of distinct tokens in the document collection. In this model, the query would simply be "natural language computational linguistics," which would be treated as a vector with the value 1 for these four words (or **terms**, as they are called in IR) and the value 0 for all the other terms. Finding documents is then a matter of comparing this vector against a collection of other vectors and reporting the ones that are close. The vector model is more flexible than the Boolean model because the documents can be ranked by their distance to the query, and the closest ones can be reported first.

There are many variations on this model. Some systems are equipped with morphological analyzers that match "linguistic computation" with "computational linguistics." Some allow the query to state that two words must appear near each other to count as a match, and others use a thesaurus to automatically augment the words in the query with their synonyms. Only the most naive systems count all the terms in the vectors equally. Most systems ignore common words like "the" and "a," and many systems weight each term differently. A good way to do this is to give a term a larger weight if it is a good discriminator: if it appears in a small number of documents rather than in many of them.

This model of information retrieval is almost entirely at the word level. It admits a minuscule amount of syntax in that words can be required to be near each other, and allows a similarly tiny role for semantic classes in the form of synonym lists. You might think that IR would perform much better if it used some more sophisticated natural language processing techniques. Many people have thought just that, but surprisingly, none has been able to show a significant improvement on a wide range of IR tasks. It is possible to tune NLP techniques to a particular subject domain, but nobody has been able to successfully apply NLP to an unrestricted range of texts.

The moral is that most of the information in a text is contained in the words. The IR approach does a good job of applying statistical techniques to capture most of this information. It is as if we took all the words in a document, sorted them alphabetically, and then very carefully compared that list to another sorted list. While the sort loses a lot of information about the original document, it often maintains enough to decide if two sorted lists are on similar topics. In contrast, the NLP technology we have today can sometimes pick out additional information—disambiguating words and determining the relations between phrases—but it often fails to recover anything at all. We are just beginning to see hybrid IR/NLP systems that combine the two approaches.

Text categorization

NLP techniques have proven successful in a related task: sorting text into fixed topic categories. There are several commercial services that provide access to news wire stories in this manner. A subscriber can ask for all the news on a particular industry, company, or geographic area, for example. The providers of these services have traditionally used human experts to assign

the categories. In the last few years, NLP systems have proven to be just as accurate, correctly categorizing over 90% of the news stories. They are also far faster and more consistent, so there has been a switch from humans to automated systems.

Text categorization is amenable to NLP techniques where IR is not because the categories are fixed, and thus the system builders can spend the time tuning their program to the problem. For example, in a dictionary, the primary definition of the word "crude" is vulgar, but in a large sample of the *Wall Street Journal*, "crude" refers to oil 100% of the time.

Extracting data from text

The task of data extraction is to take on-line text and derive from it some assertions that can be put into a structured database. For example, the SCISOR system (Jacobs and Rau, 1990) is able to take the following Dow Jones News Service story:

PILLSBURY SURGED 3 3-4 TO 62 IN BIG BOARD COMPOSITE TRADING OF 3.1 MILLION SHARES AFTER BRITAIN'S GRAND METROPOLITAN RAISED ITS HOSTILE TENDER OFFER BY \$3 A SHARE TO \$63. THE COMPANY PROMPTLY REJECTED THE SWEETENED BID, WHICH CAME AFTER THE TWO SIDES COULDN'T AGREE TO A HIGHER OFFER ON FRIENDLY TERMS OVER THE WEEKEND.

and generate this template to add to a database:

```
Corp-Takeover-Core:  
  Subevent: Increased Offer, Rejected Offer  
  Type: Hostile  
  Target: Pillsbury  
  Suitor: Grand Metropolitan  
  Share-Price: 63  
  Stock-Exchange: NYSE  
  Volume: 3.1M  
  Effect-On-Stock: (Up Increment: 3 3-4, To: 62)
```

23.2 EFFICIENT PARSING

Consider the following two sentences:

Have the students in section 2 of Computer Science 101 take the exam.

Have the students in section 2 of Computer Science 101 taken the exam?

Even though they share the first ten words, these sentences have very different parses, because the first is a command and the second is a question. A left-to-right parsing algorithm like the one in Section 22.4 that nondeterministically tries to build the right structure would have to guess if the first word is part of a command or a question, and will not be able to tell if the guess is correct until at least the eleventh word, "take/taken." If the algorithm guessed wrong, it will have to

backtrack all the way to the first word. This kind of backtracking is inevitable, but if our parsing algorithm is to be efficient, it must avoid reanalyzing "the students in section 2 of Computer Science 101" as an *NP* each time it backtracks.

In this section, we look at efficient parsing algorithms. *At the broadest level, there are three main things we can do to improve efficiency:*

1. Don't do twice what you can do once.
2. Don't do once what you can avoid altogether.
3. Don't represent distinctions that you don't need.

To be more specific, we will design a parsing algorithm that does the following:

1. Once we discover that "the students in section 2 of Computer Science 101" is an *NP*, it is a good idea to record that result in a data structure known as a **chart**. Algorithms that do this are called **chart parsers**. Because we are dealing with context-free grammars, any phrase that was found in the context of one branch of the search space can work just as well in any other branch of the search space. Recording results in the chart is a form of dynamic programming that avoids duplicate work.
2. We will see that our chart-parsing algorithm uses a combination of top-down and bottom-up processing in a way that means it never has to consider certain constituents that could not lead to a complete parse. (This also means it can handle grammars with both left-recursive rules and rules with empty right-hand sides without going into an infinite loop.)
3. The result of our algorithm is a **packed forest** of parse tree constituents rather than an enumeration of all possible trees. We will see later why this is important.

CHART

PACKED FOREST

VERTICES
EDGES

The chart is a data structure for representing partial results of the parsing process in such a way that they can be reused later on. The chart for an *n*-word sentence consists of *n* + 1 **vertices** and a number of edges that connect vertices. Figure 23.1 shows a chart with 6 vertices (circles), and 3 edges (lines). For example, the edge labelled

$$[0,5, S \rightarrow NP VP \bullet]$$

means that an *NP* followed by a *VP* combine to make an *S* that spans the string from 0 to 5. The symbol \bullet in an edge separates what has been found so far from what remains to be found.² Edges with the \bullet at the end are called **complete edges**. The edge

$$[0,2, S \rightarrow NP \bullet VP]$$

says that an *NP* spans the string from 0 to 2, and if we could find a *VP* to follow it, then we would have an *S*. Edges like this with the dot before the end are called incomplete edges,³ and we say that the edge is looking for a *VP*. We have already seen two ways to look at the parsing process. In BOTTOM-UP-PARSE on page 666, we described parsing as a process of building words into trees, backtracking when necessary. With Definite Clause Grammar, we described parsing as a form of logical inference on strings. Backtracking was used when several rules could derive the

² It is because of the \bullet that edges are often called **dotted rules**. We think this term is a little confusing, because there can be many dotted rules corresponding to the same grammar rule.

³ Some authors call these **active** edges. In some papers (Earley, 1970), edges are called **states**; the idea is that an incomplete edge marks an intermediate state in the process of finding a complete constituent.

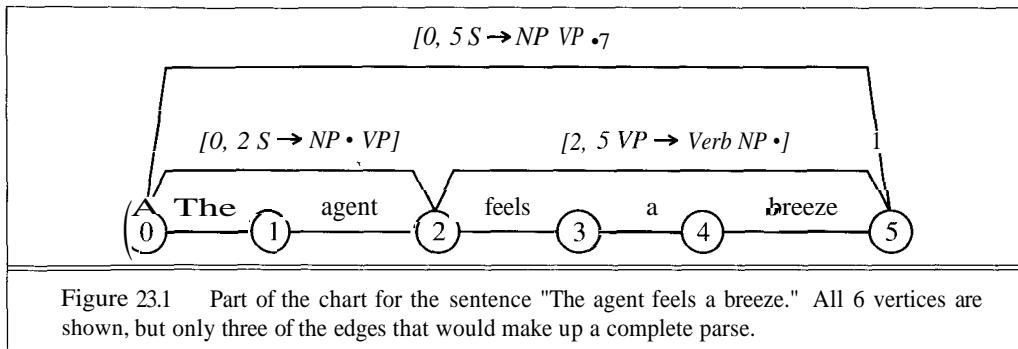


Figure 23.1 Part of the chart for the sentence "The agent feels a breeze." All 6 vertices are shown, but only three of the edges that would make up a complete parse.

same predicate. Now we will see a third approach, chart-parsing. Under this view, the process of parsing an n -word sentence consists of forming a chart with $n + 1$ vertices and adding edges to the chart one at a time, trying to produce a complete edge that spans from vertex 0 to n and is of category S . There is no backtracking; everything that is put in the chart stays there.

INITIALIZER
PREDICTOR
COMPLETER
SCANNER

There are four ways to add an edge to the chart, and we can give each one a name: The **initializer** adds an edge to indicate that we are looking for the start symbol of the grammar, S , starting at position 0, but that we have not found anything yet. The **predictor** takes an incomplete edge that is looking for an X and adds new incomplete edges that, if completed, would build an X in the right place. The **completer** takes an incomplete edge that is looking for an X and ends at vertex j and a complete edge that begins at 7 and has X as the left-hand side, and combines them to make a new edge where the X has been found. Finally, the **scanner** is similar to the completer, except that it uses the input words rather than existing complete edges to generate the X . That is, if there is an edge ending at vertex j that is looking for a *Noun*, and if the j th word in the input string has a *Noun* entry in the lexicon, then the scanner will add a new edge that incorporates the word, and goes to vertex $j + 1$.

We will show two versions of chart-parsing algorithms. Figure 23.2 treats the chart as a set of edges and at each step adds one new edge to the set, nondeterministically choosing between the possible additions. This algorithm uses the operator **pick** rather than **choose** to indicate that it has no backtrack points. Any order of choices leads to the same result in the end. The algorithm terminates when none of the four methods can add a new edge. We use a slight trick to start: we add the edge $[0, 0, S' \rightarrow \bullet S]$ to the chart, where S is the grammar's start symbol, and S' is a new symbol that we just invented. This edge makes the PREDICTOR add an edge for each grammar rule with S on the left-hand side, which is just what we need to start.

Figures 23.3 and 23.4 show a chart and trace of the algorithm parsing the sentence "I feel it." Thirteen edges (labelled a-m) are recorded in the chart, including five complete edges (shown above the vertices of the chart) and eight incomplete ones (below the vertices). Note the cycle of predictor, scanner, and completer actions. For example, the predictor uses the fact that edge (a) is looking for an S to license the prediction of an *NP* (edge b) and a *Pronoun* (edge c). Then the scanner recognizes that there is a *Pronoun* in the right place (edge d), and the completer combines the incomplete edge b with the complete edge d to yield a new edge, e. Note that the name COMPLETER is misleading in that the edges it produces (like e) are not necessarily complete. We use the name because it has a long history, but a better name might have been EXTENDER.

```

function NONDETERMINISTIC-CHART-PARSE(string, grammar) returns chart
  INITIALIZER:
    chart  $\leftarrow [0, 0, S' \rightarrow \cdot S]$ 
    while new edges can still be added do
      edge  $\leftarrow$  choose  $[i, j, A \rightarrow a \rightarrow \dots \rightarrow S \beta]$  in chart
      choose one of the three methods that will succeed:
        PREDICTOR:
          choose  $(B \rightarrow \gamma)$  in RULES[grammar]
          add  $[j, j, B \rightarrow \cdot \gamma]$  to chart
        COMPLETER:
          choose  $[j, k, B \rightarrow F \cdot]$  in chart
          add  $[i, k, A \rightarrow a B \cdot \beta]$  to chart
        SCANNER:
          if string[j+1] is of category B then
            add  $[j, j+1, A \rightarrow a S \cdot \beta]$  to chart
    end
    return chart

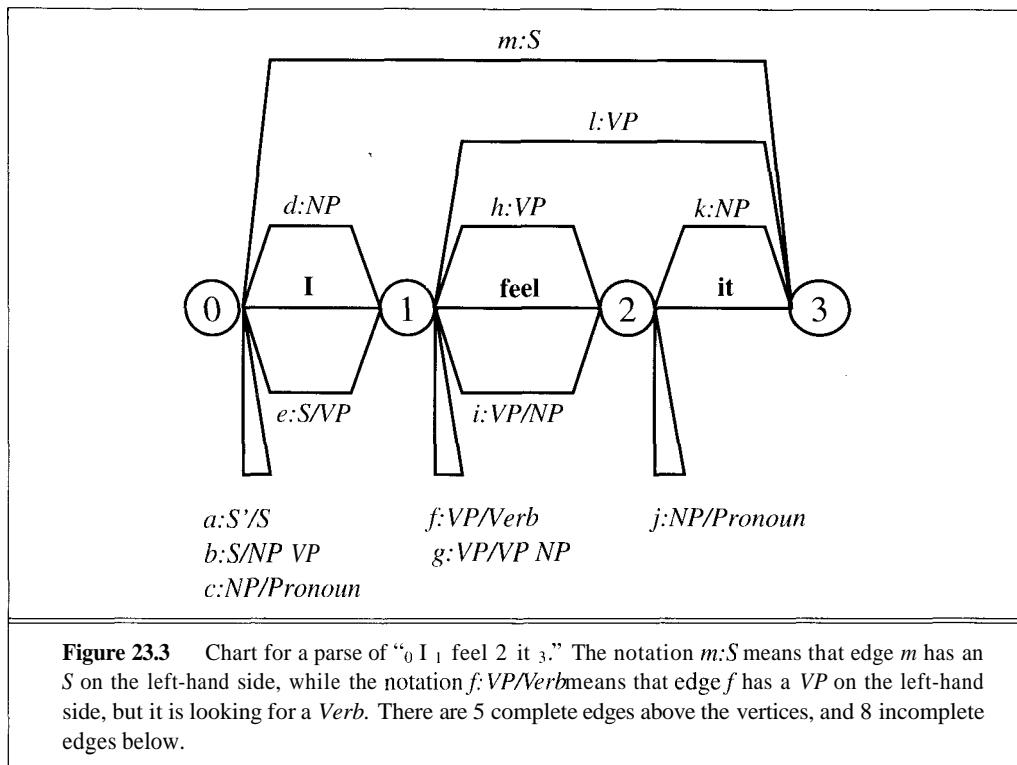
```

Figure 23.2 Nondeterministic chart parsing algorithm. S is the start symbol and S' is a new nonterminal symbol. The Greek letters match a string of zero or more symbols. The variable *edge* is an edge looking for a *B*. The predictor adds an edge that will form a *B*, the completer chooses a complete edge with *B* on the left-hand side and adds a new edge that is just like *edge* except the dot is advanced past *B*. The scanner advances the dot if the next word is of category *B*.

LEFT-CORNER

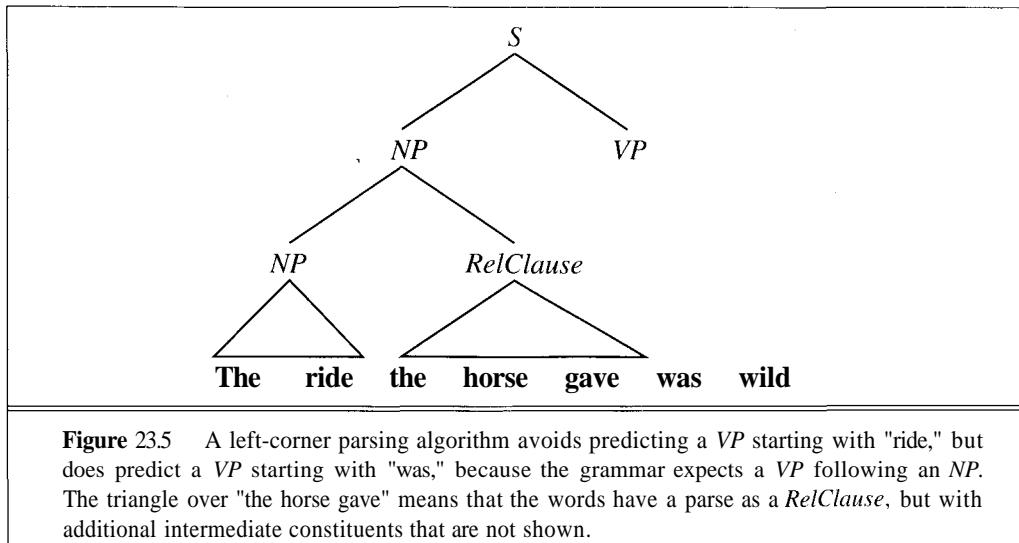
An important feature of our chart-parsing algorithm is that it avoids building some edges that could not possibly be part of an *S* spanning the whole string. Consider the sentence "The ride the horse gave was wild." Some algorithms would parse "ride the horse" as a *VP*, and then discard it when it is found not to fit into a larger *S*. But if we assume that the grammar does not allow a *VP* to follow "the," then the chart-parsing algorithm will never predict a *VP* at that point, and thus will avoid wasting time building the *VP* constituent there. Algorithms that have this property are called **left-corner** parsers, because they build up a parse tree that starts with the grammar's start symbol and extends down to the left-most word in the sentence (the left corner). An edge is added to the chart only if it can serve to extend this parse tree. See Figure 23.5 for an example of this.

Our algorithm has the constraint that the edges are added in left-to-right order. That is, if edge $[i, j, A \rightarrow B]$ is added before $[i', j', C \rightarrow D]$, then it must be that $j < j'$. Figure 23.6 shows a deterministic implementation that obeys this constraint. To get efficiency, we index edges in the chart by their ending vertex number. The notation *chart*[*j*] means the set of edges that end at vertex *j*. Additional indexing of edges may lead to further efficiency: the loop in SCANNER could be eliminated if we indexed edges at a vertex by the terminal symbol they are looking for, and the loop in COMPLETER could be eliminated if we indexed the complete edges at a vertex by their left-hand side. The algorithm also indexes rules so that REWRITES-FOR(*X,G*) returns all rules in *G* whose left-hand side is *X*.



Edge	Procedure	Derivation
a	INITIALIZER	[0,0, $S' \rightarrow \bullet S$]
b	PREDICTOR(a)	[0,0, $S \rightarrow \bullet NP VP$]
c	PREDICTOR(b)	[0,0, $NP \rightarrow \bullet Pronoun$]
d	SCANNER(C)	[0,1, $NP \rightarrow Pronoun \bullet$]
e	COMPLETER(b,d)	[0,1, $S \rightarrow NP \bullet VP$]
f	PREDICTOR(e)	[1,1, $VP \rightarrow \bullet Verb$]
g	PREDICTOR(e)	[1,1, $VP \rightarrow \bullet VNP$]
h	SCANNER(f)	[1,2, $VP \rightarrow Verb \bullet$]
i	COMPLETER(g,h)	[1,2, $VP \rightarrow VP \bullet NP$]
j	PREDICTOR(g)	[2,2, $NP \rightarrow \bullet Pronoun$]
k	SCANNER(j)	[2,3, $NP \rightarrow Pronoun \bullet$]
l	COMPLETER(i,k)	[1,3, $VP \rightarrow VP NP \bullet$]
m	COMPLETER(e,l)	[0,3, $S \rightarrow NP VP \bullet$]

Figure 23.4 Trace of a parse of “₀ I ₁ feel ₂ it ₃. ” For each edge a-m, we show the procedure used to derive the edge from other edges already in the chart.



Extracting parses from the chart: Packing

When the chart-parsing algorithm finishes, it returns the entire chart, but what we really want is a parse tree (or trees). Depending on how the parser is used, we may want to pick out one or all the parse trees that span the entire input, or we may want to look at some subtrees that do not span the whole input. If we have an augmented grammar, we may only want to look at the semantic augmentation, ignoring the syntactic structure. In any case, we need to be able to extract parses from the chart.

The easiest way to do that is to modify COMPLETER so that when it combines two child edges to produce a parent edge, it stores in the parent edge the list of children that comprise it. Then, when we are done with the parse, we need only look in *chart[n]* for an edge that starts at 0, and recursively look at the children lists to reproduce a complete parse tree. The only complication is deciding what to do about ambiguous parses. To see why this is a problem, let us look at an example. The sentence

Fall leaves fall and spring leaves spring

is highly ambiguous because each word (except "and") can be either a noun or a verb, and "fall" and "spring" can be adjectives as well. Altogether the sentence has four parses:⁴

- [S [S [NP Fall leaves] fall] and [S [NP spring leaves] spring]
- [S [S [NP Fall leaves] fall] and [S spring [VP leaves spring]]]
- [S [S Fall [VP leaves fall]] and [S [NP spring leaves] spring]
- [S [S Fall [VP leaves fall]] and [S spring [VP leaves spring]]]

⁴ The parse [S Fall [VP leaves fall]] is equivalent to "Autumn abandons autumn."

```

function CHART-PARSE(string, grammar) returns chart
  chart[0] — {[0, 0,  $S' \rightarrow \bullet S$ ]}
  for v — from 1 to LENGTH(string) do
    SCANNER(v, string[v])
  end
  return chart

procedure ADD-EDGE(edge)
  if edge in chart[END(edge)] then do nothing
  else
    push edge on chart[END(edge)]
    if COMPLETE?(edge) then COMPLETER(edge)
    else PREDICTOR(edge)
  procedure SCANNER(j, word)
    for each [i, j, A —  $a \bullet B \beta$ ] in chart[j] do
      if word is of category B then
        ADD-EDGE([i, j+1, A —  $\alpha B \bullet \beta$ ])
    end

  procedure PREDICTOR([i, j, A —  $a \bullet B \beta$ ])
    for each (B —  $\gamma$ ) in REWRITES-FOR(B, grammar) do
      ADD-EDGE([j, j, B —  $\bullet \gamma$ ])
    end

  procedure COMPLETER([j, k, B —  $\gamma$ ])
    for each [i, j, A —  $a \bullet B' \beta$ ] in chart[j] do
      if B = S' then
        ADD-EDGE([i, k, A —  $\alpha B' \bullet \beta$ ])
    end

```

Figure 23.6 Deterministic version of the chart-parsing algorithm. *S* is the start symbol and *S'* is a new nonterminal symbol. The function ADD-EDGE adds an edge to the chart, and either completes it or predicts from it.

The ambiguity can be divided into two independent parts: each of the two subsentences is ambiguous in two ways. If we had a sentence with n such subsentences joined by conjunctions, then we would get one big sentence with 2^n parses. (There also would be ambiguity in the way the subsentences conjoin with each other, but that is another story, one that is told quite well by Church and Patil (1982).) An exponential number of parses is a bad thing, and one way to avoid the problem is to represent the parses implicitly. Consider the following representation:

$$[S [S \left\{ \begin{array}{l} [\text{NP Fall leaves}] [\text{VP fall}] \\ [\text{NP Fall}] [\text{VP leaves fall}] \end{array} \right\}] \text{ and } [S \left\{ \begin{array}{l} [\text{NP spring leaves}] [\text{VP spring}] \\ [\text{NP spring}] [\text{VP leaves spring}] \end{array} \right\}]]$$

Instead of multiplying out the ambiguity to yield 2^n separate parse trees, we have one big "tree" with ambiguous subparts represented by curly braces. Of course, when $n = 2$, there is not much

difference between 2^n and $2n$, but for large n , this representation offers considerable saving. The representation is called a **packed forest**, because it is the equivalent to a set of trees (a forest), but they are efficiently packed into one structure.

To implement the packed forest representation, we modify COMPLETER to keep track of lists of possible children, and we modify ADD-EDGE so that when we go to add an edge that is already in the chart, we merge its list of possible children with the list that is already there.

We end up with a parsing algorithm that is $O(n^3)$ in the worst case (where n is the number of words in the input). This is the best that can be achieved for a context-free grammar. Note that without the packed forest, the algorithm would be exponential in the worst case, because it is possible for a sentence to have $O(2^n)$ different parse trees. In practice, one can expect a good implementation of the algorithm to parse on the order of 100 words per second, with variation depending on the complexity of the grammar and the input.

23.3 SCALING UP THE LEXICON

TOKENIZATION

In Chapter 22, the input was a sequence of words. In real text-understanding systems, the input is a sequence of characters from which the words must be extracted. Most systems follow a four-step process of tokenization, morphological analysis, dictionary lookup, and error recovery.

MORPHOLOGICAL ANALYSIS

Tokenization is the process of dividing the input into distinct tokens—words and punctuation marks. In Japanese, this is difficult because there are no spaces between words. Languages like English are easier, but not trivial. A hyphen at the end of the line may be an inter- or intraword dash. In some types of text, font changes, underlines, superscripts, and other control sequences must be accounted for. Tokenization routines are designed to be fast, with the idea that as long as they are consistent in breaking up the input text into tokens, any problems can always be handled at some later stage of processing.

INFLECTIONAL MORPHOLOGY

Morphological analysis is the process of describing a word in terms of the prefixes, suffixes, and root forms that comprise it. There are three ways that words can be composed:

DERIVATIONAL MORPHOLOGY

0 **Inflectional morphology** reflects the changes to a word that are needed in a particular grammatical context. For example, most nouns take the suffix “s” when they are plural.

COMPOUNDING

0 **Derivational morphology** derives a new word from another word that is usually of a different category. For example, the noun “shortness” is derived from the adjective “short” together with the suffix “ness.”

◇ **Compounding** takes two words and puts them together. For example, “bookkeeper” is a compound of “book” and “keeper.” (The noun “keeper” is in turn derived from the verb “keep” by derivational morphology.)

Even in a morphologically simple language like English, there can be morphological ambiguities. “Walks” can be either a plural noun or a third-person singular verb. “Unionizable” can be analyzed as “un-ion-izable” or “union-izable,” and “untieable” can be “un-(tie-able)” or “(un-tie)-able.” Many languages make more use of morphology than English. In German, it is not uncommon to see words like “Lebensversicherungsgesellschaftsangestellter” (life insur-

DICTIONARY LOOKUP

ance company employee). Languages such as Finnish, Turkish, Inuit, and Yupik have recursive morphological rules that can generate an infinite number of infinitely long words.

Dictionary lookup is performed on every token (except for special ones such as punctuation). It may be more efficient to store morphologically complex words like "walked" in the dictionary, or it may be better to do morphological analysis first: a morphological rule applies to the input and says that we strip off the "ed" and look up "walk." If we find that it is a verb that is not marked as being irregular, then the rule says that "walked" is the past tense of the root verb. Either way, the task of dictionary lookup is to find a word in the dictionary and return its definition. Thus, any implementation of the *table* abstract data type can serve as a dictionary. Good choices include hash tables, binary trees, b-trees, and tries. The choice depends in part on if there is room to fit the dictionary in primary storage, or if it resides in a file.

ERROR RECOVERY

Error recovery is undertaken when a word is not found in the dictionary. There are at least four types of error recovery. First, morphological rules can guess at the word's syntactic class: "smarply" is not in the dictionary, but it is probably an adverb. Second, capitalization is a clue that a word (or sequence of words) is a proper name. Third, other specialized formats denote dates, times, social security numbers, and so forth. These are often domain-dependent.

Finally, spelling correction routines can be used to find a word in the dictionary that is close to the input word. There are two popular models of "closeness" between words. In the letter-based model, an error consists of inserting or deleting a single letter, transposing two adjacent letters, or replacing one letter with another. Thus, a 10-letter word is one error away from 555 other words: 10 deletions, 9 swaps, 10 x 25 replacements, and 11 x 26 insertions. Exercise 23.11 discusses the implications of this for dictionary implementation. This model is good for correcting slips of the finger, where one key on the keyboard is hit instead of another.

In the sound-based model, words are translated into a canonical form that preserves most of information needed to pronounce the word, but abstracts away some of the details. For example, the word "attention" might be translated into the sequence [a,T,a,N,SH,a,N], where "a" stands for any vowel. The idea is that words such as "attension" and "atennshun" translate to the same sequence. If no other word in the dictionary translates to the same sequence, then we can unambiguously correct the spelling error. Note that the letter-based approach would work just as well for "attension," but not for "atennshun," which is 5 errors away from "attention."

Practical NLP systems have lexicons with from 10,000 to 100,000 root word forms. Building a lexicon of this size is a big investment in time and money, and one that dictionary publishers and companies with NLP programs have not been willing to share. An exception is Wordnet, a freely available dictionary of roughly 100,000 words produced by a group at Princeton led by George Miller. Figure 23.7 gives some of the information on the word "ride" in Wordnet.

As useful as dictionaries like Wordnet are, they do not provide all the lexical information you would like. The two missing pieces are frequency information and semantic restrictions. Frequency information tells us that the teasing sense of ride is unusual, while the other senses are common, or that the female swan sense of "pen" is very rare, while the other senses are common. Semantic restrictions tell us that the direct object of the first sense of ride is a horse, camel, or similar animal, while the direct object of the second kind is a means of conveyance such as a car, bus, skateboard, or airplane. Some frequency information and semantic restrictions can be captured with the help of a large **corpus** of text.

```
Noun: ride
    => mechanical device (device based on mechanical principles)
Noun: drive, ride
    => journey (the act of traveling)
Verb: ride, ride an animal (of animals)
    => travel, go, move, change location, locomote
    *> Somebody rides
    *> Somebody rides something
Verb: ride, travel in a conveyance
    => travel, go, move, change location, locomote
    OP walk, go on foot, foot, leg it, hoof, hoof it
    *> Somebody rides
    *> Somebody rides something
Verb: tease, cod, tantalize, bait, taunt, twit, rally, ride
    => mock, bemock, treat with contempt
    *> Somebody rides somebody
Other words containing "ride":
    rider, joyride, ride piggyback, ride the bench,
    phencyclidine hydrochloride ...
```

Figure 23.7 Part of Wordnet's information for "ride." Wordnet distinguishes two noun and three verb senses, and lists the superclass of each sense. (The user has the option of following superclass links all the way up or down the tree.) There is also one opposite listed (OP), many superclass relations (\Rightarrow), and for each verb, a list of subcategorization frames ($*\Rightarrow$). Finally, a few of the other entries with "ride" are listed to give an idea of the breadth of coverage. Note that expressions like "ride piggyback" are included in addition to individual words. You can get Wordnet by anonymous ftp from `clarity.princeton.edu`.

23.4 SCALING UP THE GRAMMAR

Figure 23.8 shows two examples of real-life language. These examples contrast with our simple \mathcal{E}_2 language in many ways. In tokenization, we have to deal with hyphenation (e.g., `smal- 1est`), unusual punctuation conventions (e.g., `fnctl()`), and formatting (the unindented DESCRIPTION indicating a section header). Lexically, we have novel words like `cmd` and `FD_CLOEXEC`. Syntactically, we have an odd sort of conjunction (read/write); apposition of two noun phrases (the argument `fd`); the "greater than or equal to" construction, which can be treated either as an idiom or an unusual disjunction of post-nominal modifiers; and the fact that "the same file pointer" is the object of "shares," even though there are a dozen words between them. And then there is the 67-word alleged sentence, which is actually a convoluted subordinate clause at best.

To make any sense at all out of these real-life examples requires much more sophistication at every step of the language interpretation process. In this section, we extend \mathcal{E}_2 to yield \mathcal{E}_3 , a language that covers much more, but still has a long way to go.

DESCRIPTION

`fcntl()` performs a variety of functions on open descriptors. The argument `fd` is an open descriptor used by `cmd` as follows:

`F_DUPFD` Returns a new descriptor, which has the smallest value greater than or equal to `arg`. It refers to the same object as the original descriptor, and has the same access mode (read, write or read/write). The new descriptor shares descriptor status flags with `fd`, and if the object was a file, the same file pointer. It is also associated with a `FD_CLOEXEC` (close-on-exec) flag set to remain open across `execve(2V)` system calls.

"And since I was not informed—as a matter of fact, since I did not know that there were excess funds until we, ourselves, in that checkup after the whole thing blew up, and that was, if you'll remember, that was the incident in which the attorney general came to me and told me that he had seen a memo that indicated that there were no more funds."

Figure 23.8 Two examples of real-life language: one from the UNIX manual entry for `fcntl`, and one from a statement made by Ronald Reagan, printed in the May 4, 1987, *Roll Call*.

Nominal compounds and apposition

Technical text is full of nominal compounds: strings of nouns such as "POSTSCRIPT language code input file." The first thing we have to do to handle nominal compounds is realize that we are dealing with nouns, not *NPs*. It is not the case that "the input" and "the file" combine to form "the input the file." Rather, we have two nouns combining to form a larger unit that still can combine with an article to form a *NP*. We will call the larger unit a noun⁵ and thus will need a rule of the form:

Noun → *Noun Noun*

For our example compound, the parse that leads to a semantically sensible interpretation is

[*Noun* [*Noun* [*Noun* POSTSCRIPT language] code] [*Noun* input file]]

The hardest part about nominal compounds is specifying the semantics. Our example can be paraphrased as "a file that is used for input and that consists of code written in a language named POSTSCRIPT." Clearly, there is a wide variety in the meaning associated with a nominal compound. We will use the generic relation *NN* to stand for any of the semantic relations that

⁵ Some grammars introduce intermediate categories between noun and *NP*.

Example Nominal Compound	Relation	Semantic Rule
input file code file language code POSTSCRIPT language	UsedFor ConsistsOf WrittenIn Named	$\forall x, y \text{ } UsedFor(y, x) \Rightarrow NN(x, y)$ $\forall x, y \text{ } ConsistsOf(y, x) \Rightarrow NN(x, y)$ $\forall x, y \text{ } WrittenIn(y, x) \Rightarrow NN(x, y)$ $\forall x, y \text{ } Named(y, x) \Rightarrow NN(x, y)$
basketball shoes baby shoes alligator shoes designer shoes brake shoes	UsedFor UsedBy MadeOf MadeBy PartOf	$\forall x, y \text{ } UsedFor(y, x) \Rightarrow NN(x, y)$ $\forall x, y \text{ } UsedBy(y, x) \Rightarrow NN(x, y)$ $\forall x, y \text{ } MadeOf(y, x) \Rightarrow NN(x, y)$ $\forall x, y \text{ } MadeBy(y, x) \Rightarrow NN(x, y)$ $\forall x, y \text{ } PartOf(y, x) \Rightarrow NN(x, y)$
Figure 23.9 Nominal compounds and the corresponding semantic relationships.		

can hold between two nouns in a nominal compound. We can then write logical rules that say, for example, that when a *UsedFor* relation holds between two objects, we can infer that a *NN* relation holds. Some examples are shown in Figure 23.9.

We would like a noun phrase like "every file input" to get the semantic interpretation:

$$[V/ \exists z \text{ } Input(i) A File(f) A NN(i, f)]$$

Given what we have already decided about representing the semantics of *NPs* and articles, we can get this if we arrange for the semantics of the nominal compound "file input" to be

$$\lambda f \exists z \text{ } Input(i) A File(f) A NN(i, f)$$

And we can get that with the rule

$$Noun(\backslash y \exists x \text{ } sem_1(x) A sem_2(y) A NN(x, y)) \rightarrow Noun(sem_1) Noun(sem_2)$$

APPOSITION

Another complication is that two noun phrases (not two *Nouns*) can be concatenated together in a construction called **apposition**, in which both noun phrases refer to the same thing. Two examples are "[*NP* the argument] [*NP* *fdl*]" and "[*NP* the language] [*NP* POSTSCRIPT]." In these examples, the second *NP* is a name, but it need not always be. In "[*NP*David McDonald] [*NP* the CMU grad] wrote about nominal compounds" we are using "the CMU grad" to distinguish this David McDonald from the other one.⁶ A simplified rule for apposition is

$$NP([qx sem_1] A sem_2) \rightarrow NP([qx sem_1]) NP([q x sem_2])$$

Adjective Phrases

In £2, adjectives were only allowed as complements to a verb, as in "the wumpus is *smelly*" But of course adjectives also appear as modifiers before a noun, as in "a *smelly* wumpus." The semantics of a phrase like this can be formed by a conjunction of the semantics contributed by the adjective and by the noun:

$$\exists w \text{ } Smelly(w) A Wumpus(w)$$

⁶ This is called a **restrictive** apposition because it restricts the set of possible references. Nonrestrictive appositions just add new information. They are often set off by commas or dashes, as in "Tarzan, lord of the jungle."

This is called **intersective semantics**, because the meaning of "smelly wumpus" is the intersection of smelly things and things that are wumpuses. Both nouns and adjectives are represented by predicates that define categories; this was the first scheme for representing categories shown in Section 8.4. If all adjectives had intersective semantics, it would be easy to handle them. We would add a rule for *Noun*, saying it can be composed of an *Adjective* and another *Noun*:

$$\text{Noun}(Xx \text{ sem}_1(x) A \text{ sem}_2(x)) \rightarrow \text{Adjective}(\text{sem}_1)\text{Noun}(\text{sem}_2)$$

Unfortunately, the semantic relation between adjective and noun is often more complicated than just intersection. For example:

- A "red book" has a red cover, but not red pages or lettering. A "red pen" has red ink or a red body. In general, color adjectives refer to a major, salient, visible part of the object.
- "Red hair" is orangish, not red. This implies that the modification is dependent on both the noun and the adjective.
- A "red herring" is an irrelevant distraction, not a fish nor something red. In this case, the phrase has an idiomatic meaning that is completely independent of its parts.
- A "small moon" is bigger than a "large molecule" and is clearly not the intersection of small things and things that are moons.
- A "mere child" is the same as a child: you cannot take a group of children and separate them into the children and the mere children. The adjective "mere" refers to the speaker's attitude about the noun, and not to actual properties of the noun at all.
- In "alleged murderer," the adjective again says something about the attitude of some person (not necessarily the speaker), but the phrase makes no commitment as to whether the referent actually is a murderer.
- "Real leather" is no different from "leather," but the adjective is used when the listener might expect artificial leather.
- A "fake gun" is not a gun at all. Its appearance is similar to a gun's, but it lacks the functional properties.

These kinds of semantic relations can be handled by reifying the categories that were formerly represented as predicates. This is the second scheme for representing categories in Section 8.4. For example, instead of the predicate *Gun*, we will use the object *Guns* to represent the class of all guns. Then "a gun" is represented by $3\ g\ (g\ 6\ \text{Guns})$ and "a fake gun" is $3\ g\ (g\ G\ \text{Fake}(\text{Guns}))$.

Determiners

DETERMINER

In Chapter 22, we showed how articles such as "a" and "the" can be part of noun phrases. Articles are just one type of the more general class of **determiner**, which we abbreviate as *Det*. Determiners can become quite complicated, as in "[*Det*my brother's neighbor's] dog" and "[*Det* all but three of her many] good friends." For our \mathcal{E}_3 language, we allow only one new kind of determiner, a number, as in "three dogs." We will use the quasi-logical form $[3x \text{ Dog}(x)]$ to represent this. This gives us the following grammar rules:

$$\begin{aligned} \text{Det}(q) &\rightarrow \text{Article}(q) \\ \text{Det}(q) &\rightarrow \text{Number}(q) \\ \text{NP}([q\ x\ \text{noun}(x)]) &\rightarrow \text{Det}(q)\ \text{Noun}(\text{noun}) \end{aligned}$$

So far, the rules are simple, and the mapping from strings to quasi-logical form is easy. The hard part is translating from quasi-logical form to logical form. Here are two examples of the logical form we would like to end up with:

Three women carried a piano.

$\exists s \text{ Cardinality}(s) = 3 A \forall w ((w \in s \Rightarrow \text{Woman}(w))$

$A \exists p \text{ Piano}(p) \wedge \exists e (e \in \text{Carry}_1(s, p, \text{Past}))$

(*There is a set of 3 women; this set carried the piano.*)

Three women carried a baby.

$\exists s \text{ Cardinality}(s) = 3 A \forall w ((w \in s \Rightarrow \text{Woman}(w))$

$A \exists b \text{ Baby}(b) A \exists e (e \in \text{Carry}_2(w, p, \text{Past}))$

(*There is a set of 3 women; each woman in the set carried a baby.*)

These examples are ambiguous. In the most likely interpretations, it is the set of women, s , who are carrying the piano together in the first example, whereas in the second example, each woman is separately carrying a different baby. The subscripts on *Carry* indicate different senses. To account for these two different interpretations, we will have to make the rule for translating [$3 w \text{ Woman}(w)$] capable of equating either the variable denoting the set (s) or the variable denoting the elements of the set (w) with the variable representing the subject of the verb phrase.

Noun phrases revisited

Now we look at the rules for noun phrases. The rule for pronouns is unchanged from Chapter 22, and the old rule for *Article* plus *Noun* could be updated simply by changing *Article* to *Det* and including case information and agreement in person and number, yielding the following rule:

$NP(case, Person(3), number, [q \ x \ sem(x)]) \rightarrow Det(number, q) Noun(number, sem)$

We stick to the convention that the semantics is always the last argument. The *case* variable is unbound, indicating that the *NP* can be used in either the subjective or objective case. The *number* variable can take on the values *Singular* or *Plural*, and the rule says that the *Det* and *Noun* must have the same number. For example, "a dog" and "those dogs" are grammatical because they agree in number, but "a dogs" and "those dog" are ungrammatical because they disagree. Note that some determiners (like "the") and some nouns (like "sheep") can be either singular or plural.

Besides playing a role inside the noun phrase, the *number* variable is also important externally: in the $S \rightarrow NP \ VP$ rule, the subject noun phrase must agree with the verb phrase in *number* and *person*. All nouns are in the third person, which we have denoted *Person(3)*. Pronouns have more variety; the pronoun "you," for example, is in the second person and "it" is in the third. Verbs are also marked for person and number. For example, the verb "am" is *Singular* and *Person(1)*, and therefore "I am" is grammatical, while "you am" is not. Here is a rule that enforces subject/verb agreement:

$S(rel(obj)) \rightarrow NP(Subject, person, number, obj) VP(person, number, rel)$

It is also possible to form a noun phrase from a noun with no determiner. There are several ways to do this. One is with a name, such as "John" or "Berkeley." There are several choices for the

semantics of names. The simplest choice is to represent "John" with the constant *John*. Then the representation of "John slept" is $3 \ e \ e \models Sleep(John, Past)$. But this simple approach does not work in a world with more than one thing called John. A better representation is therefore $3 \ e, x \ e \ e \ Sleep([\exists!x Name(x) = John], Past)$. Here are rules that derive that representation:

$$\begin{aligned} NP(case, Person(3), number, [\exists!x Name(x) = name]) &\rightarrow Name(number, name) \\ Name(Singular, John) &\rightarrow John \end{aligned}$$

A noun also needs no determiner if it is a mass noun (e.g., "water") or a generic plural (e.g., "Dogs like bones"). We leave these rules as exercises.

Clausal complements

In \mathcal{E}_2 , all the verbs took only noun phrases and prepositional phrases as complements. But some verbs accept clauses (i.e., sentences or certain types of verb phrases) as complements. For example, in "I believe [he has left]," the object of "believe" is a sentence, and in "I want [to go there]," the object of want is an infinitive verb phrase. We can handle this with the same subcategorization mechanism we have been using (here shown without the other augmentations):

$$\begin{aligned} VP(subcat) &\rightarrow VP([S|subcat])S \\ VP(subcat) &\rightarrow VP([VP|subcat])VP \\ Verb([S]) &\rightarrow believe \\ Verb([VP]) &\rightarrow want \end{aligned}$$

Relative clauses

The grammar of \mathcal{E}_2 allows relative clauses such as "the person [that saw me]," in which the relative clause consists of the word "that" followed by a *VP*, and the interpretation is that the head noun phrase (the person) is the subject of the embedded *VP*. In English, it is also possible to have relative clauses such as "the person [that I saw \sqcup]," in which a sentence follows the word "that," but the sentence is missing an object. The \sqcup symbol, which we call a **gap**⁷ indicates the place where the head noun phrase (the person) logically would appear to complete the sentence. We say that the head noun phrase is the **filler** of the gap. In this example, the gap is in the direct object position, but it could be nested farther into the relative clause. For example, it could be the object of a preposition (e.g., "the person [that I looked [*PP* at \sqcup]]") or the object of a deeply nested clause:

the person [that [5 you said [5 you thought [_s I gave the book to \sqcup]]]]]

So far, all the syntactic relationships we have seen have been at a single level in the parse tree. The filler-gap relationship is called a **long-distance dependency** because it reaches down a potentially unbounded number of nodes into the parse tree. The subscripts (*i*) on parse nodes are used to show that there is an identity relationship—that the recipient of the book giving is the same as "the person" that is the head noun phrase:

[the person]_i [that [_s you said [_s you thought [_s I gave the book to \sqcup_i]]]]]

GAP

FILLER

LONG-DISTANCE
DEPENDENCY

⁷ The gap is called a **trace** in some theories.

To represent filler-gap relationships, we augment most of the category predicates with a *gap* argument. This argument says what is missing from the phrase. For example, the sentence "I saw him" has no gaps, which we represent as $S(Gap(None))$. The phrases " \sqcup saw him" and "I saw \sqcup " are both represented as $S(Gap(NP))$.

To define relative clauses, all we have to do is say that an *NP* can be modified by following it with a relative clause, and that a relative clause consists of a relative pronoun followed by a sentence that contains an *NP* gap. This allows us to handle both "the person that I saw \sqcup " and "the person that \sqcup saw him." Here are the rules with all other augmentations removed:

$$\begin{aligned} NP(gap) &\rightarrow NP(gap) \text{ RelClause} \\ \text{RelClause} &\rightarrow \text{Pronoun(Relative)} S(Gap(NP)) \end{aligned}$$

We also have to say that the empty string, ϵ , comprises an *NP* with an *NP* gap in it:

$$NP(Gap(NP)) \text{--- } 6$$

The rest of the grammar has to cooperate by passing the *gap* argument along; for example:

$$S(Gap(Concat(g_1, g_2))) \text{--- } NP(Gap(g_1)) VP(Gap(g_2))$$

Here *Concat*(g_1, g_2) means g_1 and g_2 together. If g_1 and g_2 are both *Gap(None)*, then the *S* as a whole has no gap. But if, say, g_1 is *Gap(NP)* and g_2 is *Gap(None)*, then the *S* has an *NP* gap.

Questions

In English, there are two main types of questions:

- 0 **Yes/No:** Did you see that?
- ◇ Wh (gapped): What did you see \sqcup ?

SUBJECT-AUX
INVERSION

A yes/no question, as the name implies, expects a yes or no as answer. It is just like a declarative sentence, except that it has an auxiliary verb that appears before the subject *NP*. We call this **subject-aux inversion** and use the category *Sinv* to denote a sentence that has it. If there are several auxiliary verbs, only the first one comes before the subject: "Should you have been seeing that?" Thus, the grammar rules covering simple yes/no sentences are as follows:

$$\begin{aligned} S &\text{--- } Question \\ Question &\rightarrow Sinv \\ Sinv &\text{--- } Aux \text{ } NP \text{ } VP \end{aligned}$$

Wh questions (pronounced "double-U H") expect a noun phrase as an answer. In the simplest case, they start with an interrogative pronoun (who, what, when, where, why, how), which is followed by a gapped *Sinv*:

$$Question \text{ --- } \text{Pronoun(Interrogative)} \text{ } Sinv(Gap(NP))$$

There are also some less common question constructions, as these examples show:

- 0 **Echo:** You saw *what*?
- 0 **Raising intonation:** You see something?
- ◇ **Yes/No with 'be':** Is it safe?
- 0 **Wh subject:** What is the frequency, Kenneth?

- 0 Wh **NP**: [What book] did you read ↗?
- 0 Wh **PP**: [With what] did you see it ↗?
- ◊ Wh **PP**: [Whence] did he come ↗?

Echo questions can also be answered with a noun phrase, but they are normally used rhetorically to express the speaker's amazement at what was just said. For example, "I saw a 30-foot-high purple cow" is answered by "You saw *what*?" Sentences with normal declarative structure can be made into questions with the use of raising intonation at the end of the sentence. This is uncommon in written text. The verb "to be" is the only verb that can stand by itself (without another verb) in an inverted yes/no question. That is, we can say "Is it safe?" but not "Seems it safe?" or "Did they it?" In some dialects, "have" can be used, as in "Have you the time?" or "Have you any wool?" Finally, it is possible to have an *Sinv* with a prepositional phrase gap by prefacing it with a prepositional wh phrase like "from where" or "whence."

Handling agrammatical strings

No matter how thorough the grammar, there will always be strings that fall outside it. It doesn't much matter if this happens because the string is a mistake or because the grammar is missing something. Either way, the speaker is trying to communicate something and the system must process it in some way. Thus, it is the hearer's job to interpret a string somehow, even if it is not completely grammatical. For example, if a character in a mystery novel suddenly keels over but manages to gasp "Poison—Butler—Martini," most people would be able to come up with a good interpretation of these dying words: the butler is the agent of the poisoning action of which the speaker is the victim and the martini is the instrument. We arrive at this interpretation by considering the possible semantic relations that could link each component together, and choosing the best one. It would not do to say "I'm sorry, that's not a grammatical sentence; could you please rephrase it?"

23.5 AMBIGUITY



In this chapter, we have extended the range of syntactic constructions and semantic representations that we can handle. This helps us cover a wider range of language, but it also makes the job of disambiguation harder, because there are more possibilities to choose from. *Finding the right interpretation involves reasoning with uncertainty* using the evidence provided by lexical, syntactic, semantic, and pragmatic sources.

Historically, most approaches to the disambiguation problem have been based on logical inference with no quantitative measures of certainty. In the last few years, the trend has been toward quantitative probabilistic models such as belief networks and hidden Markov models.

Belief networks provide an answer to one hard part of the problem—how to combine evidence from different sources. But we are still left with two more problems: deciding what evidence to put into the network, and deciding what to do with the answers that come back. We



start by looking at several sources of evidence. *It is important to distinguish between sources of evidence and sources of ambiguity.* It is quite possible that a syntactic ambiguity is resolved by semantic evidence. For example, in "I ate spaghetti and meatballs and salad," there is a syntactic ambiguity (is it [_{NP} spaghetti and meatballs] or [_{NP} meatballs and salad]?) that is resolved by nonsyntactic evidence: that spaghetti and meatballs is a common dish, whereas meatballs and salad is not.

Syntactic evidence

Modifiers such as adverbs and prepositional phrases cause a lot of ambiguity, because they can attach to several different heads. For example, in

Lee asked Kim to tell Toby to leave on Saturday.

the adverb "Saturday" can modify the asking, the telling, or the leaving. Psychological studies show that in the absence of other evidence, there is a preference to attach such modifiers to the most recent constituent (in this case, the leaving). So this is a syntactic ambiguity that can be resolved by syntactic evidence.

Lexical evidence

Many words are ambiguous, but not all senses of a word are equally likely. When asked what the word "pen" means, most people will say first that it is a writing instrument. However, it has two other fairly common senses meaning an enclosure for animals and a penitentiary. The senses meaning a female swan and the internal shell of a squid are more obscure, known mostly to specialists in biology (or specialists in ambiguity). As another example, consider the following:

Lee positioned the dress on the rack.

Kim wanted the dress on the rack.

Here "on the rack" is best interpreted as modifying the verb in the first sentence and the dress in the second. That is, Lee is putting the dress on the rack (not moving a dress that is on the rack), and Kim wants to have a dress that is on the rack (and does not want the dress to be on the rack). In both cases, either interpretation is semantically plausible. So this is a syntactic ambiguity that is resolved by lexical evidence—by the preference of the verb for one subcategorization.

Semantic evidence

The a priori probability of a word sense is usually less important than the conditional probability in a given context. Consider the following:

ball, diamond, bat, base

Even without any syntactic structure to speak of, it is easy to pick out the baseball senses of each of these words as being the intended interpretation. This is true even though the a priori probability for "diamond" favors the jewel interpretation. So this is a case of lexical ambiguity resolved by semantic evidence.

As another example, here are five sentences, along with the most likely interpretation of the relation represented by the word "with." Each of the sentences has a lexical ambiguity ("with" has several senses) and a syntactic ambiguity (the *PP* can attach to either "spaghetti" or "ate"), but each is resolved by semantic evidence.

Sentence	Relation
I ate spaghetti with meatballs.	(ingredient of spaghetti)
I ate spaghetti with salad.	(side dish of spaghetti)
I ate spaghetti with abandon.	(manner of eating)
I ate spaghetti with a fork.	(instrument of eating)
I ate spaghetti with a friend.	(accompanier of eating)

It is certainly *possible* to use meatballs as a utensil with which to eat spaghetti, but it is unlikely (not to mention messy), so we prefer interpretations that refer to more likely events. Of course, likeliness of events or situations is not the only factor. In Chapter 22, we saw that the right interpretation of "I am not a crook" is not the most likely situation. It is perfectly coherent to use language to talk about unlikely or even impossible situations: "I ran a mile in one minute" or "This square is a triangle."

Metonymy

METONYMY

A **metonymy** is a figure of speech in which one object is used to stand for another. When we hear "Chrysler announced a new model," we do not interpret it as saying that companies can talk; rather we understand that a spokesperson representing the company made the announcement. Metonymy is common in many kinds of text, and often goes unnoticed by human readers. Unfortunately, our grammar as it is written is not so facile. To handle the semantics of metonymy properly, we need to introduce a whole new level of ambiguity. We do this by providing two objects for the semantic interpretation of every phrase in the sentence: one for the object that the phrase literally refers to (Chrysler), and one for the metonymic reference (the spokesperson). We then have to say that there is a relation between the two. In our current grammar, "Chrysler announced" gets interpreted as

$$\exists x, e \ Chrysl(x) \wedge e \in \text{Announce}(x, \text{Past})$$

We need to change that to

$$\exists m, x, e \ Chrysl(x) \wedge \text{Metonymy}(m, x) \wedge e \in \text{Announce}(m, \text{Past})$$

This gives a representation for the problem, but not a solution. The next step is to define what kinds of metonymy can occur, that is, to define constraints on the *Metonymy* relation. The simplest case is when there is no metonymy at all—the literal object *x* and the metonymic object *m* are identical:

$$\forall m, x \ (m = x) \Rightarrow \text{Metonymy}(m, x)$$

For the Chrysler example, a reasonable generalization is that an organization can be used to stand for a spokesperson of that organization:

$$\forall m, x \ \text{Organization}(x) \wedge \text{Spokesperson}(m, x) \Rightarrow \text{Metonymy}(m, x)$$

Other metonymies include the author for the works (I read Shakespeare) or more generally the producer for the product (I drive a Honda) and the part for the whole (The Red Sox need a strong arm). Other examples of metonymy, such as "The ham sandwich on Table 4 wants another beer," are somewhat harder to classify.

Metaphor

METAPHOR

A **metaphor** is a figure of speech in which a phrase with one literal meaning is used to suggest a different meaning by way of an analogy. Most people think of metaphor as a tool used by poets that does not play a large role in everyday text. However, there are a number of basic metaphors that are so common that we do not even recognize them as such. One such metaphor is the idea that *more is up*. This metaphor allows us to say that prices have risen, climbed, or skyrocketed, that the temperature has dipped or fallen, that one's confidence has plummeted, or that a celebrity's popularity has jumped or soared.

There are two ways to approach metaphors like this. One is to compile all knowledge of the metaphor into the lexicon—to add new senses of the words rise, fall, climb, and so on, that describe them as dealing with quantities on any scale rather than just altitude. This approach suffices for many applications, but it does not capture the generative character of the metaphor that allows humans to use new instances such as "nose-dive" without fear of misunderstanding. The second approach is to include explicit knowledge of common metaphors and use them to interpret new uses as they are read. For example, suppose the system knows the "*more is up*" metaphor. That is, it knows that logical expressions that refer to a point on a vertical scale can be interpreted as being about corresponding points on a quantity scale. Then the expression "sales are high" would get a literal interpretation along the lines of *Altitude(Sales, High)*, which could be interpreted metaphorically as *Quantity(Sales, Much)*.

23.6 DISCOURSE UNDERSTANDING

DISCOURSE TEXT

In the technical sense, a **discourse or text** is any string of language, usually one that is more than one sentence long. Novels, weather reports, textbooks, conversations, and almost all nontrivial uses of language are discourses. So far we have largely ignored the problems of discourse, preferring to dissect language into individual sentences that can be studied *in vitro*. In this section, we study sentences in their native habitat.

To produce discourse, a speaker must go through the standard steps of intention, generation, and synthesis. Discourse understanding includes perception, analysis (and thus syntactic, semantic, and pragmatic interpretation), disambiguation, and incorporation. The hearer's state of knowledge plays a crucial part in arriving at an understanding—two agents with different knowledge may well understand the same text differently. Discourse understanding can be modelled crudely by the following equation:

$$KB' = \text{DISCOURSE-UNDERSTANDING}(text, KB)$$

where KB is the hearer's knowledge base, and KB' is the hearer's knowledge base after incorporating the text. The difference between KB and KB' is the hearer's **understanding** of the text. At least six types of knowledge come into play in arriving at an understanding:

1. General knowledge about the world.
2. General knowledge about the structure of coherent discourse.
3. General knowledge about syntax and semantics.
4. Specific knowledge about the situation being discussed.
5. Specific knowledge about the beliefs of the characters.
6. Specific knowledge about the beliefs of the speaker.

Let us look at an example discourse:

John went to a fancy restaurant.
He was pleased and gave the waiter a big tip.
He spent \$50.

A proper understanding of this discourse would include the fact that John ate a fancy meal at the restaurant, that the waiter was employed by the restaurant, and that John paid some of the \$50 to the waiter and most of it to the restaurant. We'll call this understanding (a). All the inferences seem obvious, but to get them right we need to know a lot and apply the knowledge in just the right way. To understand why this is hard, we give an alternative understanding (b):

John ducked into a fancy restaurant to ask directions. He was pleased that they were able to help him. Back on the street, John bumped into a man that he had met at a party the other night. All John could remember was that the man was a waiter at another restaurant and that he was interested in getting a new radio. John gave the man a tip that there was a great sale going on at the stereo store down the block. The man spent \$50 on a radio.

This is a situation that could conceivably be described by our three-sentence discourse. It is far-fetched for two reasons: First, the situation in (a) has a higher *a priori* probability—people go to restaurants to eat more often than to ask for directions. Second, the situation in (a) is more probable given the text. A rational speaker would not expect a hearer to extract understanding (b) from the discourse. To see why (a) is better, let us look at it piece by piece:

- John ate a fancy meal at the restaurant.

To get this requires going beyond disambiguation and into incorporation. There is certainly no part of the discourse that mentions the eating explicitly, but it still should be part of the updated knowledge base that the hearer comes away with. To get it, the hearer has to know that restaurants serve meals, and thus that a reason for going to a restaurant is to eat. The hearer also knows that fancy restaurants serve fancy meals, and that \$50 is a typical price for such a meal, and that paying and leaving a tip are typically done after eating a restaurant meal. Besides this general knowledge about the world, it also helps if the hearer knows that discourses are commonly structured so that they describe some steps of a plan for a character, but leave out steps that can be easily inferred from the other steps. From this, the hearer can infer from the first sentence that John has adopted the eat-at-restaurant plan, and that the eat-meal step probably occurred even if it was not mentioned.

- The waiter was employed by the restaurant.

Again, this is a mixture of general world knowledge—that restaurants employ waiters—and knowledge about discourse conventions—that the definite article "the" is used for objects that have been mentioned before, or at least have been implicitly alluded to; in this case, by the eat-at-restaurant plan.

- John paid some of the \$50 to the waiter and most of it to the restaurant.

This is another example of identifying a step in a currently active plan that matches a sentence in the text, and unifying them to arrive at the interpretation that "He" is John, and that the recipients of the deal are the restaurant and waiter.

The structure of coherent discourse

In logic, conjunction is commutative, so there is no difference between $P \wedge Q \wedge R$ and $R \wedge P \wedge Q$. But this is certainly not true of natural languages. Consider the following two discourses, which contain the same sentences in different order:

Discourse (A)

I visited Paris.
I bought you some expensive cologne.
Then I flew home.
I went to Kmart.
I bought some underwear.

Discourse (B)

I visited Paris.
Then I flew home.
I went to Kmart.
I bought you some expensive cologne.
I bought some underwear.

In (A), the preferred interpretation is that the cologne comes from Paris, whereas in (B) it comes from Kmart. Both discourses have a sequential structure in which the sentences that come later in the discourse also occur later in time. But there is more to the understanding of these discourses than just temporal ordering. In (A), we understand that a (or the) reason for going to Kmart was to buy the underwear. Similarly, we understand that visiting Paris enabled the speaker to buy cologne there, but that was not necessarily the purpose of the trip.

We need a theory of how discourses are put together. We will say that discourses are composed of **segments**, where a segment can be a clause, a complete sentence, or a group of several consecutive sentences. There are several theories built on the idea that each segment of the discourse is related to the previous one by a **coherence relation** that determines the role that each segment plays in the unfolding discourse. For example, the coherence relation between the first two sentences of (A) is one of **enablement**. *Coherence relations serve to bind the discourse together.* A speaker knows that she can extend a discourse by adding a sentence that stands in a coherence relation to the existing text, and the hearer knows that in addition to interpreting and disambiguating each sentence, he is supposed to recover the coherence relations that bind the segments together. This means that the hearer of a discourse is doing more work than the hearer in Chapter 22, who only had to worry about interpreting and disambiguating a single sentence. But it also means that the hearer of a discourse has some help, because the coherence relations constrain the possible meanings of each sentence. So even though the individual sentences may have many possible meanings, the discourse as a whole will have few meanings (preferably one).

SEGMENTS

COHERENCE
RELATION



We present the theory by Hobbs (1990), which starts with the observation that the speaker does four things in putting together a discourse:

- The speaker wants to convey a message.
- The speaker has a motivation or goal in doing this.
- The speaker wants to make it easy for the hearer to understand.
- The speaker must link the new information to what the hearer already knows.

A sentence is a coherent extension of a discourse if it can be interpreted by the hearer as being in service of one of these four points. Let us look at some examples of each of the four as they appear in the following discourse:

- (1) A funny thing happened yesterday.
- (2) John went to a fancy restaurant.
- (3) He ordered the duck.
- (4) The bill came to \$50.
- (5) John got a shock when the waiter came to collect the bill.
- (6) He realized he didn't have a cent on him.
- (7) He had left his wallet at home.
- (8) The waiter said it was all right to pay later.
- (9) He was very embarrassed by his forgetfulness.

Sentence (1) describes what the speaker wants to talk about—a funny thing. It is a meta-comment, an evaluation by the speaker of what the coming message holds. Once the speaker has made it clear that she has the goal of describing this, it is coherent to add (2), which starts to describe the funny thing. The coherence relation between (1) and (2) is that uttering (2) is part of the speaker's plan that was implicitly alluded to by (1). More formally, we can say that two adjacent discourse segments S_i and S_j stand in the **evaluation** coherence relation, if from S_i we can infer that S_j is a step in the speaker's plan for achieving some discourse goal.

Sentences (3) and (4) are coherent because they can be interpreted as steps in John's plan of eating a meal. Sentences (2) and (3) stand in the enablement coherence relation, and (3) and (4) are in the **causal** relation. Both relations arise from the speaker's goal of conveying a message about the world. Thus, we see that understanding discourse involves two levels of **plan recognition**—recognizing the speaker's plans and the characters' plans.

Sentences (5) and (6) overlap in their content: the shock is the realization, but it is described in a different way. We say that (5) and (6) stand in the **elaboration** relation. This is an example of the speaker making it easier for the hearer by lessening the amount of inference the hearer has to make. Sentence (6) could well have been left out of the discourse, but then the jump from (5) to (7) would have been a little harder to make. Note that once we recognize that (6) is an elaboration of (5), we can infer that "he" in (6) refers to John. Without the coherence relation, we might be tempted to assume that "he" refers to the waiter, who was mentioned more recently.

The relation between (6) and (7) is one of **explanation**: the reason John didn't have a cent on him is because he left his wallet at home. This is one case where a sentence that appears later in the discourse actually occurs earlier in the real world. This is an example of the speaker linking a new explanation to the hearer's existing knowledge.

Sentence (9) stands in a causal relation with the discourse segment comprised of (5) and (6). Recognizing this allows us to interpret "he" in (9) as John, not the waiter.

There have been several catalogs of coherence relations. Mann and Thompson (1983) give a more elaborate one that includes solutionhood, evidence, justification, motivation, reason, sequence, enablement, elaboration, restatement, condition, circumstance, cause, concession, background, and thesis-antithesis.

ATTENTION

The theory of Grosz and Sidner (1986) also accounts for where the speaker and hearer's **attention** is focused during the discourse. Their theory includes a pushdown stack of focus spaces. Certain utterances cause the focus to shift by pushing or popping elements off the stack. For example, in Discourse (A), the sentence "I visited Paris" causes a new focus to be pushed on the stack. Within that focus, the speaker can use definite descriptions to refer to, say, "the museums" or "the cafes." The sentence "Then I flew home" would cause the focus on Paris to be popped from the stack; from that point on, the speaker would need to use an indefinite description such as "the cafe I went to in Paris" rather than a simple definite description. There is also a more specific notion of local focus that affects the interpretation of pronouns.

23.7 SUMMARY

In this chapter, we have seen what can be done with state-of-the-art natural language processing, and what is involved in getting there. The main points were as follows:

- Natural language processing techniques make it practical to develop programs that make queries to a database, extract information from texts, retrieve relevant documents from a collection, translate from one language to another, or recognize spoken words. In all these areas, there exist programs that are useful, but there are no programs that do a thorough job in an open-ended domain.
- It is possible to parse sentences efficiently using an algorithm that is careful to save its work, avoid unnecessary work, and pack the results into a forest rather than an exhaustive list of trees.
- In recent years, there has been a shift of emphasis from the grammar to the lexicon. Building a lexicon is a difficult task.
- Natural languages have a huge variety of syntactic forms. Nobody has yet captured them all, but we can extend the simple grammar of the last chapter to give more complete coverage of English sentences.
- Choosing the right interpretation of a sentence in the situation in which it was uttered requires evidence from many sources, including syntactic, lexical, and semantic.
- Most of the interesting language comes in connected discourse rather than in isolated sentences. Coherence relations constrain how a discourse is put together, and thus constrain the possible interpretations it has.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

Locke and Booth (1955) present the very early protohistory of machine translation, and capture the enthusiasm of the early days. Later disillusionment with machine translation is described by Bar-Hillel (1960); Lindsay (1963) also points out some of the obstacles to machine translation having to do with the interaction between syntax and semantics, and the need for world knowledge. Quinlan and O'Brien (1992) describe TAUM-METEO. Voorhees (1993) reports some recent applications based on Wordnet.

One problem with the prototype natural language interfaces we describe—LUNAR (Woods, 1972), CHAT (Pereira, 1983), and TEAM (Grosz *et al.*, 1987), is that they are necessarily incomplete. In the mid-1980s, several companies invested the effort to build serious natural language database access programs. Artificial Intelligence Corporation's INTELLECT (Harris, 1984) is one of the best known. With a decade of tuning it performs better than a prototype, but still has the difficult problem of revealing to the user its range of competence. The NL-MENU system (Tennant *et al.*, 1983) mitigates this problem by using a mode of interaction in which the user builds a query by selecting words and phrases from a series of on-screen menus instead of typing. In other words, the system reveals what it is capable of accepting rather than attempting to handle anything the user throws at it.

The fundamental advances in efficient, general parsing methods were made in the late 1960s, with a few twists since then (Kasami, 1965; Younger, 1967; Earley, 1970; Graham *et al.*, 1980). Maxwell and Kaplan (1993) show how chart parsing with non-context-free grammars can be made efficient in the average case. Church and Patil (1982) introduce some refinements in the resolution of syntactic ambiguity. A number of researchers have attempted to use quantitative measures of the goodness of syntactic and semantic fit in parsing and semantic interpretation. This can either be done within a basically deductive framework (Hobbs *et al.*, 1993) or within the framework of Bayesian belief network reasoning (Charniak and Goldman, 1992; Wu, 1993).

Nunberg (1979) gives an outline of a computational model of metonymy. Lakoff and Johnson (1980) give an engaging analysis and catalog of common metaphors in English. Ortony (1979) presents a collection of articles on metaphor; Helman (1988) focuses on analogical reasoning, but also contains a number of articles about the phenomenon of metaphor. Martin (1990) presents a computational approach to metaphor interpretation.

Kukich (1992) surveys the literature on spelling correction.

Hobbs (1990) outlines the theory of discourse coherence on which this chapter's exposition is based. Mann and Thompson (1983) give a similar catalog of coherence relations. Grimes (1975) lays the groundwork for much of this work. Grosz and Sidner (1986) present a theory of discourse coherence based on shifting focus of attention. Joshi, Webber, and Sag (1981) collect important early work in the field. Webber presents a model of the interacting constraints of syntax and discourse on what can be said at any point in the discourse (1983) and of the way verb tense interacts with discourse (1988). The idea of tracking the characters' goals and plans as a means of understanding stories was first studied by Wilensky (Wilensky, 1983). A plan-based model of speech acts was suggested first by Cohen and Perrault (1979). Cohen, Morgan, and Pollack (1990) collect more recent work in this area.

EXERCISES

23.1 Read the following text once for understanding and remember as much of it as you can. There will be a test later.

The procedure is actually quite simple. First you arrange things into different groups. Of course, one pile may be sufficient depending on how much there is to do. If you have to go somewhere else due to lack of facilities that is the next step, otherwise you are pretty well set. It is important not to overdo things. That is, it is better to do too few things at once than too many. In the short run this may not seem important but complications can easily arise. A mistake is expensive as well. At first the whole procedure will seem complicated. Soon, however, it will become just another facet of life. It is difficult to foresee any end to the necessity for this task in the immediate future, but then one can never tell. After the procedure is completed one arranges the material into different groups again. Then they can be put into their appropriate places. Eventually they will be used once more and the whole cycle will have to be repeated. However, this is part of life.

23.2 Describe how a simple pseudo-natural-language (template-based) explanation facility can be built for a vanilla, backward-chaining, logical reasoning system. The explanation facility should be written as a program WHY that generates an explanation after ASK has answered a question from the user. The explanation should consist of a description of the premises and inference method used to reach the conclusion; the user should be able to further query the premises to see how they were derived.

23.3 Without looking back at Exercise 23.1, answer the following questions:

- What are the four steps that are mentioned?
- What step is left out?
- What is "the material" that is mentioned in the text?
- What kind of mistake would be expensive?
- Is it better to do too few or too many?
- Why?

23.4 Open any book or magazine to a random page and write down the first 20 nominal compounds you find. Characterize the semantic relations (e.g., made-of, used-for, etc.).

23.5 Open any book or magazine to a random page and copy down the first 10 sentences. How many of them are in £3? Show the parses of the sentences that are, and explain what went wrong for the sentences that are not.

23.6 Work out the grammar rules for possessive noun phrases. There are three parts to this. (a) Write a rule of the form $Det \rightarrow NP(Case(Possessive))$. (b) Make sure this rule combines with the $NP \rightarrow Det\ Noun$ rule to produce the right semantics. (c) Write a rule that builds a possessive

NP using the 's ending. You will first have to decide if 's attaches to a noun or *NP*. To help you do this, consider at least the following examples:

- the Queen of England's speech
- the attorney general's decision
- the man I saw yesterday's name
- the man and woman's house
- Russell and Norvig's text

23.7 Collect some examples of time expressions, such as "two o'clock," "midnight," and "12:46." Also think up some examples that are ungrammatical, such as "thirteen o'clock" or "half past two fifteen." Write a grammar for the time language.

23.8 In this exercise, you will write rules for noun phrases consisting of a noun with no determiner. Consider these examples:

- a. Dogs like bones.
- b. I ate rice.
- c. Gold is valuable.
- d. I saw some gold in 2,2.
- e. I saw gold in 2,2.

Write down the semantics for each of these sentences. Then use that to write down lexical entries for "gold" and "rice," and to write a rule (or rules) of the form *NP* — *Noun*.

23.9 We said that $\exists e, x \ e \in Sleep(John, Past)$ A $Name(x) = John$ was a plausible interpretation for "John slept." But it is not quite right, because it blurs the distinction between "John slept" and "Some person named John slept." The point is that the former sentence presupposes that speaker and hearer agree on which John is being talked about. Write grammar and lexical rules that will distinguish the two examples.

23.10 Write grammar rules for the category *AdjP*, or adjective phrase, using reified categories. Show how to derive $\exists g \ (g \models Fake(Gun))$ as the semantics of "a fake gun." An adjective phrase can be either a lone adjective (*big*), a conjunction (*big and dumb*), or an adjective phrase modified by an adjective or adverb (*light green* or *very dumb*).

23.11 One way to define the task of spelling correction is this: given a misspelled word and a dictionary of correctly spelled words, find the word(s) in the dictionary that can be transformed into the misspelled word in one insertion, deletion, substitution, or transposition. Given a dictionary of w words and a misspelled word that is k letters long, give the average case time complexity of spelling correction for a dictionary implemented as (a) a hash table, (b) a b-tree, and (c) a trie.

23.12 We forgot to mention that the title of the text in Exercise 23.1 is "Washing Clothes." Go back and reread the text, and answer the questions in Exercise 23.3. Did you do better this time? Bransford and Johnson (1973) used this text in a better-controlled experiment and found that the title helped significantly. What does this tell you about discourse comprehension?

23.13 Implement a version of the chart-parsing algorithm that returns a packed tree of all edges that span the entire input.

23.14 Implement a version of the chart-parsing algorithm that returns a packed tree for the longest leftmost edge, and then if that edge does not span the whole input, continues the parse from the end of that edge. Show why you will need to call PREDICT before continuing. The final result is a list of packed trees such that the list as a whole spans the input.

24 PERCEPTION

In which we connect the computer to the raw, unwashed world.

24.1 INTRODUCTION

SENSORS

Perception provides agents with information about the world they inhabit. Perception is initiated by **sensors**. A sensor is anything that can change the computational state of the agent in response to a change in the state of the world. It could be as simple as a one-bit sensor that detects whether a switch is on or off, or as complex as the retina of the human eye, which contains more than a hundred million photosensitive elements.

There are a variety of sensory modalities that are available to artificial agents. Those they share with humans include vision, hearing, and touch. In this chapter, our focus will be on vision, because this is by far the most useful sense for dealing with the physical world. Hearing in the context of speech recognition is also covered briefly in Section 24.7. Touch, or **tactile sensing**, is discussed in Chapter 25, where we examine its use in dexterous manipulation by robots.

We will not have all that much to say about the design of sensors themselves. The main focus will be on the processing of the raw information that they provide. The basic approach taken is to first understand how sensory stimuli are created by the world, and then to ask the following question: *if sensory stimuli are produced in such and such a way by the world, then what must the world have been like to produce this particular stimulus?* We can write a crude mathematical analogue of this question. Let the sensory stimulus be S , and let W be the world (where W will include the agent itself). If the function f describes the way in which the world generates sensory stimuli, then we have

$$S = f(W)$$

Now, our question is: given f and S , what can be said about W ? A straightforward approach would try to calculate what the world is like by inverting the equation for generating the stimulus:

$$W = f^{-1}(S)$$



Unfortunately, f does not have a proper inverse. For one thing, we cannot see around corners, so we cannot recover all aspects of the current world state from the stimulus. Moreover, even the part we *can* see is enormously ambiguous. A key aspect of the study of perception is to understand what additional information can be brought to bear to resolve ambiguity.

A second, and perhaps more important, drawback of the straightforward approach is that it is trying to solve too difficult a problem. In many cases, the agent does not *need* to know everything about the world. Sometimes, just one or two predicates are needed—such as "Is there any obstacle in front of me?" or "Is that an electrical outlet over there?"

In order to understand what sorts of processing we will need to do, let us look at some of the possible uses for vision:

MANIPULATION

◊ **Manipulation:** grasping, insertion, and so on, need local shape information and feedback ("getting closer, too far to the right, . . .) for motor control.

NAVIGATION

◊ **Navigation:** finding clear paths, avoiding obstacles, and calculating one's current velocity and orientation.

OBJECT
RECOGNITION

◊ **Object recognition:** a useful skill for distinguishing between tasty mice and dangerous carnivores; edible and inedible objects; close relations and strangers; ordinary vehicles, Volvos, and police cars.

None of these applications requires the extraction of complete descriptions of the environment.

This chapter is organized as follows. In Section 24.2, we study the process of image formation. We cover both the geometry of the process, which dictates where a given point will be imaged, and the photometry of the process, which dictates how bright the point will appear. Section 24.3 treats the basic image-processing operations commonly used in early vision. They set the stage for later analysis that extracts the information needed for tasks such as manipulation, navigation, and recognition. In Section 24.4, we study various cues in the image that can be harnessed to this end, including motion, stereopsis, texture, shading, and contour. Section 24.5 discusses the information needed for visually guided manipulation and navigation, and Section 24.6 covers various approaches to object recognition. Finally, Section 24.7 addresses the problem of perception in the context of speech recognition, thereby helping to pinpoint the issues that arise in perception in general.

24.2 IMAGE FORMATION

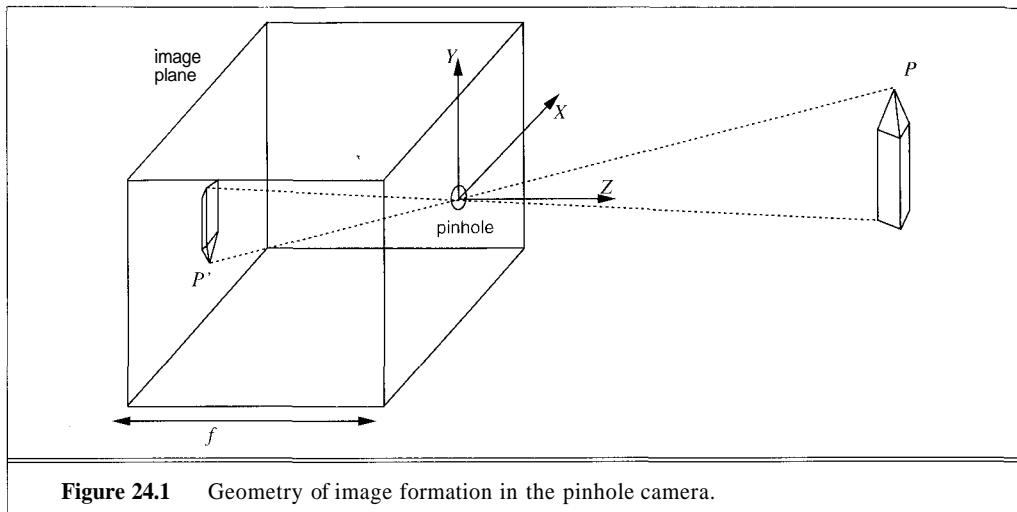
SCENE
IMAGE

Vision works by gathering light scattered from objects in the **scene** and creating a 2-D **image**. In order to use this image to obtain information about the scene, we have to understand the geometry of the process.

Pinhole camera

PINHOLE CAMERA

The simplest way to form an image is to use a **pinhole camera** (Figure 24.1). Let P be a point in the scene, with coordinates (X, Y, Z) , and P' be its image on the image plane, with coordinates



(x, y, z) . If f is the distance from the pinhole O to the image plane, then by similar triangles, we can derive the following equations:

$$\frac{-x}{f} = \frac{X}{Z}, \quad \frac{-y}{f} = \frac{Y}{Z} \quad \rightarrow \quad x = -\frac{fX}{Z}, \quad y = -\frac{fY}{Z}$$

Note that the image is *inverted*, both left-right and up-down, compared to the scene as indicated in the equations by the negative signs. These equations define an image formation process known as **perspective projection**.

Equivalently, we can model the perspective projection process with the projection plane being at a distance f in front of the pinhole. This device of imagining a projection surface in front was first recommended to painters in the Italian Renaissance by Alberti in 1435 as a technique for constructing geometrically accurate depictions of a three-dimensional scene. For our purposes, the main advantage of this model is that it avoids lateral inversion and thereby eliminates the negative signs in the perspective projection equations.

Under perspective projection, parallel lines appear to converge to a point on the horizon — think of railway tracks. Let us see why this must be so. We know from vector calculus that an arbitrary point P_λ on the line passing through (X_0, Y_0, Z_0) in the direction (U, V, W) is given by $(X_0 + \lambda U, Y_0 + \lambda V, Z_0 + \lambda W)$, with λ varying between $+\infty$ and $-\infty$. The projection of P_λ on the image plane is given by

$$\left(f \frac{X_0 + \lambda U}{Z_0 + \lambda W}, f \frac{Y_0 + \lambda V}{Z_0 + \lambda W} \right)$$

As $\lambda = \infty$ or $\lambda = -\infty$, this becomes $p_\infty = (fU/W, fV/W)$ if $W \neq 0$. We call p_∞ the **vanishing point** associated with the family of straight lines with direction (U, V, W) . It does *not* depend on the point (X_0, Y_0, Z_0) through which the straight line in the scene passes, only on the direction.

If the object is relatively shallow compared to its distance from the camera, we can approximate perspective projection by **scaled orthographic projection**. The idea is the following. If the depth Z of points on the object varies in some range $Z_0 \pm AZ$, with $AZ \ll Z_0$, then the

perspective scaling factor f/Z can be approximated by a constant $s = f/Z_0$. The equations for projection from the scene coordinates (X, Y, Z) to the image plane become $x = sX$ and $y = sY$. Note that scaled orthographic projection is an approximation valid only for parts of the scene with not much internal depth variation. It should not be used to study properties "in the large." An example to convince you of the need for caution: under orthographic projection, parallel lines stay parallel instead of converging to a vanishing point!

Lens systems

LENS Vertebrate eyes and real cameras use a **lens**. A lens is much wider than a pinhole, enabling it to let in more light. This is paid for by the fact that not all the scene can be in sharp focus at the same time. The image of an object at distance Z in the scene is produced at a fixed distance from the lens Z' , where the relation between Z and Z' is given by the lens equation

$$\frac{1}{Z} + \frac{1}{Z'} = \frac{1}{f}$$

DEPTH OF FIELD

where f is the focal length of the lens. Given a certain choice of image distance Z'_0 between the nodal point of the lens and the image plane, scene points with depths in a range around Z_0 , where Z_0 is the corresponding object distance, will be imaged in reasonably sharp focus. This range of depths in the scene is referred to as the **depth of field**.

Note that because the object distance Z is typically much greater than the image distance Z' or f , we often make the following approximation:

$$\frac{1}{Z} + \frac{1}{Z'} \approx \frac{1}{Z'} \Rightarrow \frac{1}{Z'} \approx \frac{1}{f}$$

Thus, the image distance $Z' \approx f$. We can therefore continue to use the pinhole camera perspective projection equations to describe the geometry of image formation in a lens system.

In order to focus objects at different distances Z , the lens in the eye (see Figure 24.2) changes shape, whereas the lens in a camera moves in the Z -direction.

The image plane is coated with photosensitive material:

- Silver halides on photographic film.
- Rhodopsin and variants in the retina.
- Silicon circuits in the CCD (charge-coupled device) camera. Each site in a CCD integrates the electrons released by photon absorption for a fixed time period.

PIXELS

In the eye and CCD camera, the image plane is subdivided into **pixels**: typically 512×512 (0.25×10^6) in the CCD camera, arranged in a rectangular grid; 120×10^6 rods and 6×10^6 cones in the eye, arranged in a hexagonal mosaic.

In both cases, we can model the signal detected in the image plane by the variation in image brightness over time: $I(x, y, t)$. Figure 24.3 shows a digitized image of a stapler on a desk, and Figure 24.4 shows an array of image brightness values associated with a 12×12 block of pixels extracted from the stapler image. A computer program trying to interpret the image would have to start from such a representation.

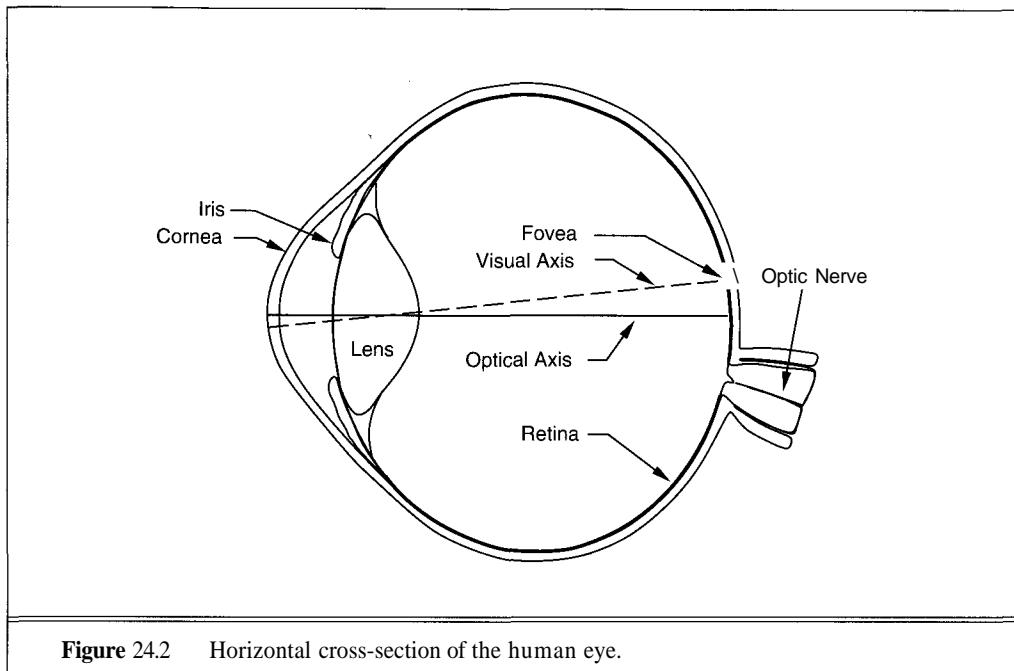


Figure 24.2 Horizontal cross-section of the human eye.

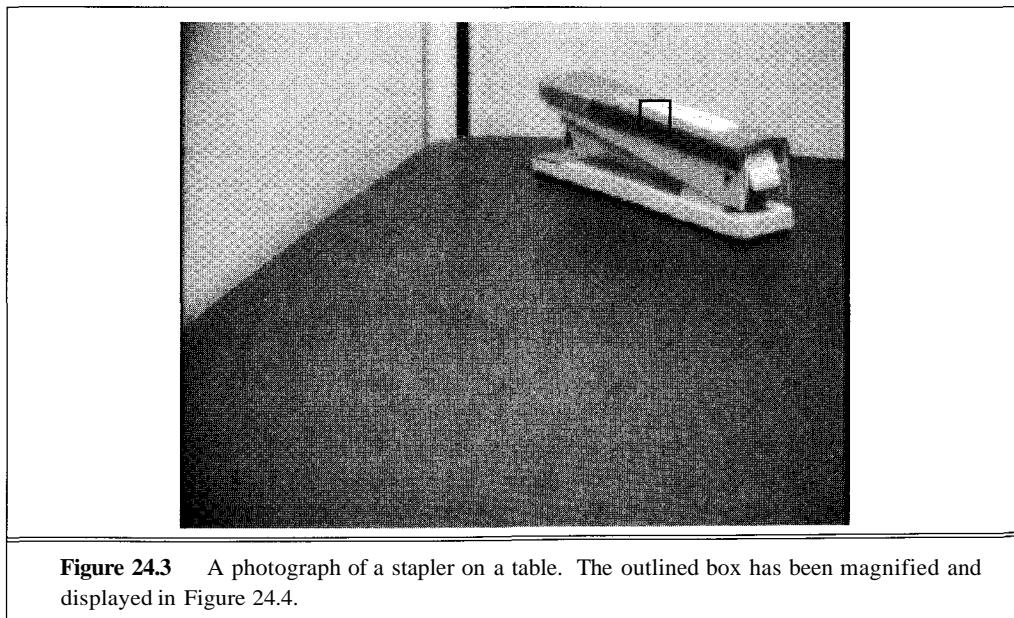


Figure 24.3 A photograph of a stapler on a table. The outlined box has been magnified and displayed in Figure 24.4.

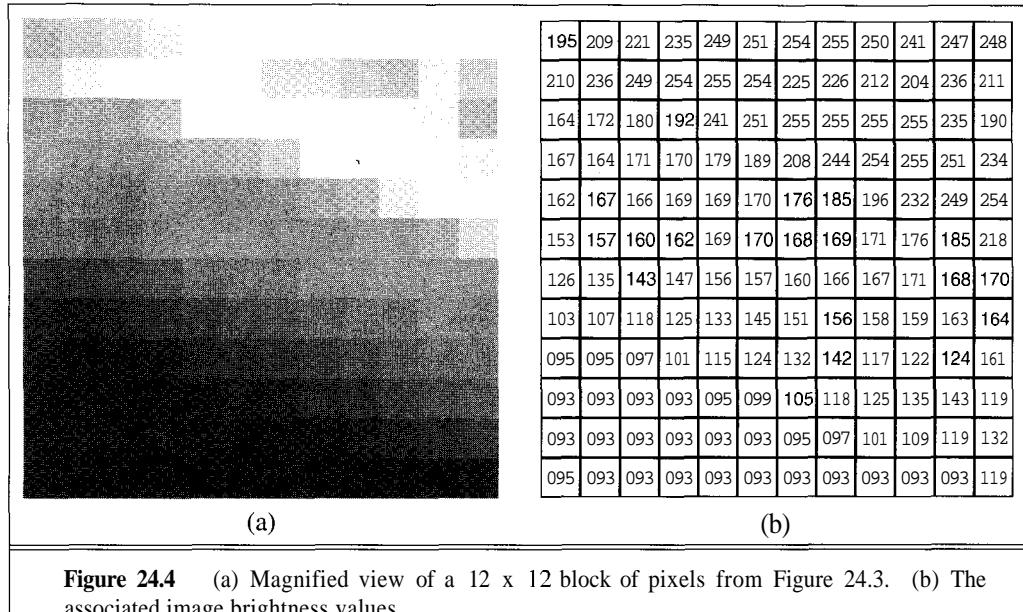


Figure 24.4 (a) Magnified view of a 12 x 12 block of pixels from Figure 24.3. (b) The associated image brightness values.

Photometry of image formation

The brightness of a pixel p in the image is proportional to the amount of light directed toward the camera by the surface patch S_p that projects to pixel p . This in turn depends on the reflectance properties of S_p , the position and distribution of the light sources. There is also a dependence on the reflectance properties of the rest of the scene because other scene surfaces can serve as indirect light sources by reflecting light received by them onto S_p .

The light reflected from an object is characterized as being either diffusely or specularly reflected. Diffusely reflected light is light that has penetrated below the surface of an object, been absorbed, and then re-emitted. The surface appears equally bright to an observer in any direction. Lambert's cosine law is used to describe the reflection of light from a perfectly diffusing or **Lambertian** surface. The intensity E of light reflected from a perfect diffuser is given by

$$E = \rho E_0 \cos \theta$$

where E_0 is the intensity of the light source; ρ is the albedo, which varies from 0 (for perfectly black surfaces) to 1 (for pure white surfaces); and θ is the angle between the light direction and the surface normal.

Specularly reflected light is reflected from the outer surface of the object. Here the energy of reflected light is concentrated primarily in a particular direction—the one where the reflected ray is in the same plane as the incident ray and satisfies the condition that the angle of reflection is equal to the angle of incidence. This is the behavior of a perfect mirror.

In real life, surfaces exhibit a combination of diffuse and specular properties. Modelling this on the computer is the bread and butter of computer graphics. Rendering realistic images is usually done by ray tracing, which aims to simulate the physical process of light originating

from light sources and being reflected and rereflected multiple times. The shape-from-shading problem in computer vision is aimed at inverting the process, that is, starting from the "rendered" image and figuring out the layout of the three-dimensional scene that gave rise to it. We will talk about this in more depth in Section 24.4.

Spectrophotometry of image formation

We have been talking of image intensity $I(x,y, t)$, merrily ignoring the fact that visible light comes in a range of wavelengths—ranging from 400 nm on the violet end of the spectrum to 700 nm on the red end. Given that there is a continuum of wavelengths, what does it mean that we have three primary colors? The explanation is that color is quite literally in the eye of the beholder. There are three different cone types in the eye with three different spectral sensitivity curves $R_k(\lambda)$. The output of the k th cone at location (x,y) at time t then is $I_k(x,y, t) = \int I(x,y, t, \lambda)R_k(\lambda)d\lambda$. The infinite dimensional wavelength space has been projected to a three-dimensional color space. This means that we ought to think of I as a three-dimensional vector at (x,y, t) . Because the eye maps many different frequency spectra into the same color sensation, we should expect that there exist **metamers**—different light spectra that appear the same to a human.

METAMERS

24.3 IMAGE-PROCESSING OPERATIONS FOR EARLY VISION

EDGES

Figure 24.5 shows an image of a scene containing a stapler resting on a table as well as the **edges** detected on this image. Edges are curves in the image plane across which there is a "significant" change in image brightness. The ultimate goal of edge detection is the construction of an idealized line drawing such as Figure 24.6. The motivation is that edge contours in the image correspond to important scene contours. In the example, we have depth discontinuities, labelled 1; surface-orientation discontinuities, labelled 2; a reflectance discontinuity, labelled 3; and an illumination discontinuity (shadow), labelled 4.

As you can see, there is a big difference between the output of an edge detector as shown in Figure 24.5(b) and an ideal line drawing. Typically, there are missing contours (such as the top edge of the stapler), as well as noise contours that do not correspond to anything of significance in the scene. Later stages of processing based on edges have to take these errors into account.

How do we detect edges in an image? Consider the profile of image brightness along a 1-D cross-section perpendicular to an edge, for example, the one between the left edge of the table and the wall. It looks something like what is shown in Figure 24.7(a). The location of the edge corresponds to $x = 50$.

Because edges correspond to locations in images where the brightness undergoes a sharp change, a naive idea would be to differentiate the image and look for places where the magnitude of the derivative $I'(x)$ is large. Well, that almost works. In Figure 24.7(b) we see that although there is a peak at $x = 50$, there are also subsidiary peaks at other locations (e.g., $x = 75$) that could potentially be mistaken as true edges. These arise because of the presence of noise in the image. We get much better results by combining the differentiation operation with **smoothing**.

SMOOTHING

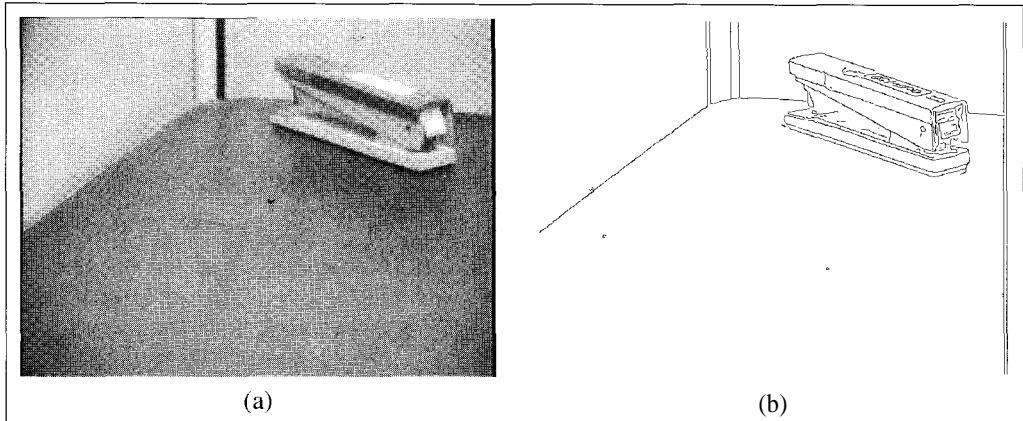


Figure 24.5 (a) Photograph of a stapler. (b) Edges computed from (a).

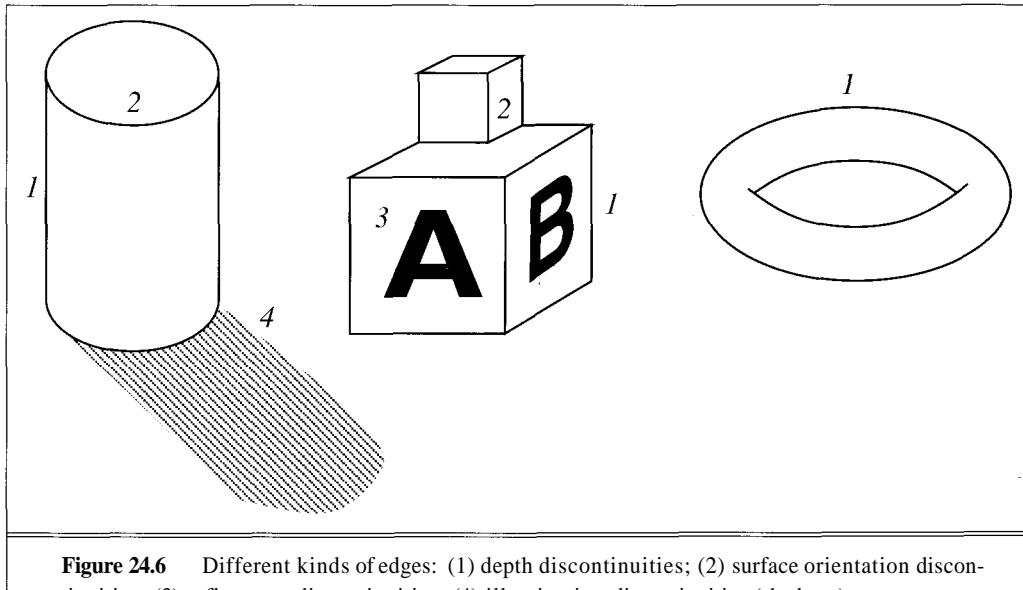
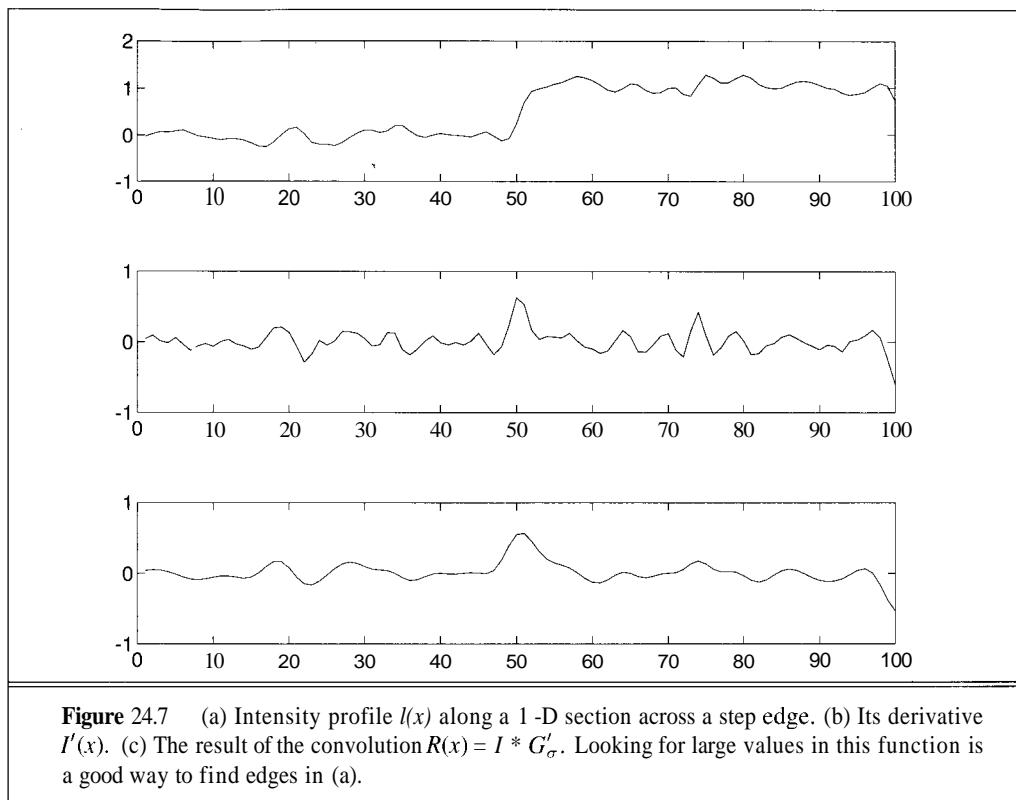


Figure 24.6 Different kinds of edges: (1) depth discontinuities; (2) surface orientation discontinuities; (3) reflectance discontinuities; (4) illumination discontinuities (shadows).

The result is Figure 24.7(c), in which the central peak at $x = 50$ remains and the subsidiary peaks are much diminished. This allows one to find the edges without getting confused by the noise.

To understand these ideas better, we need the mathematical concept of **convolution**. Many useful image-processing operations such as smoothing and differentiation can be performed by convolving the image with suitable functions.



Convolution with linear filters

The result of convolving two functions f and g is the new function h , denoted as $h = f^* g$, which is defined by

$$h(x) = \int_{-\infty}^{+\infty} f(u) g(x-u) du \quad \text{and} \quad h(x) = \sum_{u=-\infty}^{+\infty} f(u) g(x-u)$$

for continuous and discrete domains respectively. Typically, the functions f and g that we work with take nonzero values only in some finite interval, so the summation can be easily performed on a computer.

The generalization to functions defined on two dimensions (such as images) is straightforward. We replace the 1-D integrals (or sums) by 2-D integrals (or sums). The result of convolving two functions f and g is the new function h , denoted as $h = f^* g$, which is defined by

$$h(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(u, v) g(x-u, y-v) du dv$$

or if we take the domains of the two functions to be discrete

$$h(x, y) = \sum_{-\infty}^{+\infty} \sum_{-\infty}^{+\infty} f(u, v) g(x-u, y-v)$$

Edge detection

Let us go back to our 1-Dexample in Figure 24.7. We want to make the notion of image smoothing more precise. One standard form of smoothing is to convolve the image with a Gaussian function

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2}$$

Now it can be shown that for any functions f and g , $f * g' = (f * g)'$, so smoothing the image by convolving with a Gaussian G_σ and then differentiating is equivalent to convolving the image with $G'_\sigma(x)$, the first derivative of a Gaussian:

$$G'_\sigma(x) = \frac{-x}{\sqrt{2\pi}\sigma^3} e^{-x^2/2\sigma^2}$$

So, we have a simple algorithm for 1 -D edge detection:

1. Convolve the image / with G'_σ to obtain R .
2. Find the absolute value of R .
3. Mark those peaks in $\|R\|$ that are above some prespecified threshold T_n . The threshold is chosen to eliminate spurious peaks due to noise.

In two dimensions, we need to cope with the fact that the edge may be at any angle θ . To detect vertical edges, we have an obvious strategy: convolve with $G'_\sigma(x)G_\sigma(y)$. In the y -direction, the effect is just to smooth (because of the Gaussian convolution), and in the x -direction, the effect is that of differentiation accompanied with smoothing. The algorithm for detecting vertical edges then is as follows:

1. Convolve the image $I(x, y)$ with $f_V(x, y) = G'_\sigma(x)G_\sigma(y)$ to obtain $R_V(x, y)$.
2. Find the absolute value of $R_V(x, y)$.
3. Mark those peaks in $\|R_V\|(x, y)$ that are above some prespecified threshold T_n .

In order to detect an edge at an arbitrary orientation, we need to convolve the image with two filters $f_V = G'_\sigma(x)G_\sigma(y)$ and $f_H = G'_\sigma(y)G_\sigma(x)$, which is just f_V rotated by 90° . The algorithm for detecting edges at arbitrary orientations is

1. Convolve the image $I(x, y)$ with $f_V(x, y)$ and $f_H(x, y)$ to obtain $R_V(x, y)$ and $R_H(x, y)$, respectively. Define $R(x, y) = R_V^2(x, y) + R_H^2(x, y)$
2. Find the absolute value of $R(x, y)$.
3. Mark those peaks in $\|R\|(x, y)$ that are above some prespecified threshold T_n .

Once we have marked edge pixels by this algorithm, the next stage is to link those pixels that belong to the same edge curves. This can be done by assuming that any two neighboring pixels that are both edge pixels with consistent orientations must belong to the same edge curve.

We have just outlined the basic procedure used in the Canny edge detector (Canny, 1986), which is a standard algorithm widely used for detecting edges in images.

24.4 EXTRACTING 3-D INFORMATION USING VISION

We need to extract 3-D information for performing certain tasks such as manipulation, navigation, and recognition. There are three aspects of this:

1. Segmentation of the scene into distinct objects.
2. Determining the position and orientation of each object relative to the observer.
3. Determining the shape of each object.

SEGMENTATION

Segmentation of the image is a key step towards organizing the array of image pixels into regions that would correspond to semantically meaningful entities in the scene. For recognition, we would like to know what features belong together so that one could compare them with stored models; to grasp an object, one needs to know what belongs together as an object.

Edge detection, as discussed in the last section, is a useful first step toward image and scene segmentation, but not adequate by itself. This is because of two reasons: (a) some fraction of the edge curves that correspond to surface boundaries may be low contrast and not get detected; (b) many of the edge curves that are detected may correspond to noise, surface markings, or shadows. Segmentation is best viewed as part of extraction of 3-D information about the scene.

POSE

Determining the position and orientation of an object relative to the observer (the so-called **pose** of the object) is most important for manipulation and navigation tasks. To move around in a grocery store, one needs to know the locations of the obstacles, so that one can plan a path avoiding them. If one wants to pick up and grasp an object, one needs to know its position relative to the hand so that an appropriate trajectory of moves could be generated. Manipulation and navigation actions typically are done in a control loop setting—the sensory information provides feedback to modify the motion of the robot, or the robot arm. In fact, often we may be interested more in relative change of position.

Let us specify position and orientation in mathematical terms. The position of a point P in the scene is characterized by three numbers, the (X, Y, Z) coordinates of P in a coordinate frame with its origin at the pinhole and the Z -axis along the optical axis (Figure 24.1). What we have available is the perspective projection of the point in the image (x, y) . This specifies the ray from the pinhole along which P lies; what we do not know is the distance. The term "orientation" could be used in two senses:

1. The orientation of the object as a whole. This can be specified in terms of a three-dimensional rotation relating its coordinate frame to that of the camera.
2. The orientation of the surface of the object at P . This can be specified by a vector n that specifies the direction of the unit surface normal vector, which is locally perpendicular to the surface. Often we express the surface orientation using the variables **slant** and **tilt**. Slant is the angle between the Z -axis and n . Tilt is the angle between the X -axis and the projection of n on the image plane.

SLANT TILT

SHAPE

When the camera moves relative to an object, both the object's distance and its orientation change. What is preserved is the **shape** of the object. If the object is a cube, that fact is not changed when the object moves. Geometers have been attempting to formalize shape for centuries—the basic concept being that shape is what remains unchanged under some group of transformations, for

example, combinations of rotations and translations. The difficulty lies in finding a representation of global shape that is general enough to deal with the wide variety of objects in the real world—not just simple forms like cylinders, cones, and spheres—and yet can be recovered easily from the visual input. The problem of characterizing the *local* shape of a surface is much better understood. Essentially, this can be done in terms of curvature—how does the surface normal change as one moves in different directions on the surface. For a plane, there is no change at all. For a cylinder, if one moves parallel to the axis there is no change, but in the perpendicular direction, the surface normal rotates at a rate inversely proportional to the radius of the cylinder. And so on. All this is studied in the subject of differential geometry.

The shape of an object is relevant for some manipulation tasks, for example, deciding where to grasp an object. But its most significant role is in object recognition, where geometric shape along with color and texture provide the most significant cues to enable us to identify objects, classify what is in the image as an example of some class one has seen before, and so on.

The fundamental question is the following: Given the fact that during perspective projection, all points in the 3-D world along a ray from the pinhole have been projected to the same point in the image, how do we recover 3-D information? There are a number of cues available in the visual stimulus for this, including **motion**, **binocular stereopsis**, **texture**, **shading**, and **contour**. Each of these cues relies on background assumptions about physical scenes in order to provide unambiguous interpretations. We discuss each of these cues in the following subsections.

Motion

OPTICAL FLOW

If the camera moves relative to the three-dimensional scene, the resulting apparent motion in the image is called **optical flow**. This describes the direction and speed of motion of features *in the image* as a result of relative motion between the viewer and the scene. In Figure 24.8, we show two frames from a video of a rotating Rubik's cube. Figure 24.9 shows the optical flow vectors computed from these images. The optical flow encodes useful information about scene structure. For example, when viewed from a moving car, distant objects have much slower apparent motion than close objects; thus, the rate of apparent motion can tell us something about distance.

SUM OF SQUARED DIFFERENCES

The optical flow vector field can be represented by its components $v_x(x, y)$ in the x -direction and $v_y(x, y)$ in the y -direction. To measure optical flow, we need to find corresponding points between one time frame and the next. We exploit the fact that image patches around corresponding points have similar intensity patterns. Consider a block of pixels centered at pixel p , (x_0, y_0) at time t_0 . This block of pixels is to be compared with pixel blocks centered at various candidate pixels q_i at $(x_0 + D_x, y_0 + A)$ at time $t_0 + D_t$. One possible measure of similarity is the **sum of squared differences (SSD)**:

$$SSD(D_x, D_y) = \sum_{(x,y)} (I(x, y, 0) - I(x + D_x, y + D_y, t + D_t))^2$$

CROSS-CORRELATION

Here (x, y) range over pixels in the block centered at (x_0, y_0) . We find the (A, D_t) that minimizes the SSD. The optical flow at (x_0, y_0) is then $(v_x, v_y) = (A/D_t, D_y/D_t)$. Alternatively, one can maximize the **cross-correlation**:

$$Correlation(D_x, A) = \sum_{(x,y)} I(x, y, t)I(x + D_x, y + D_y, t + A)$$

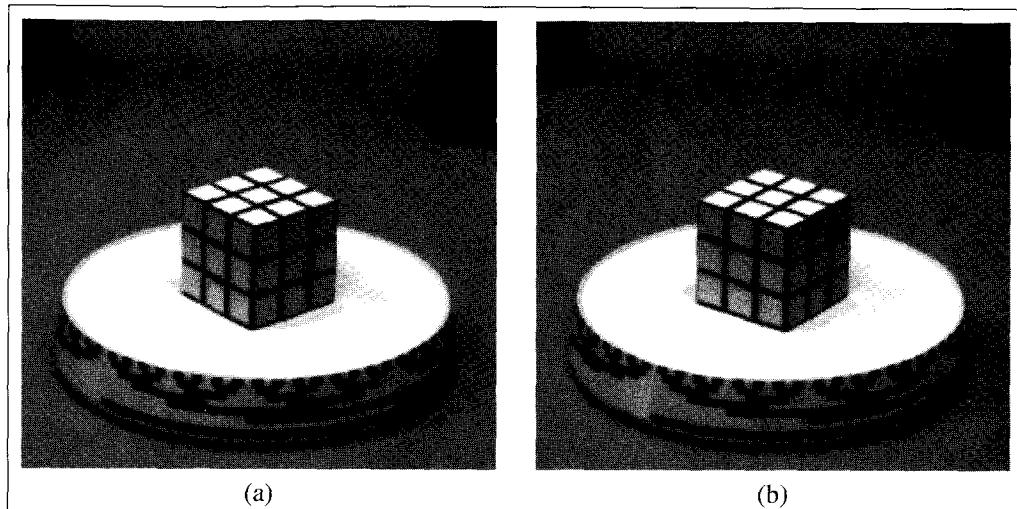


Figure 24.8 (a) A Rubik's cube on a rotating turntable. (b) The same cube, shown 19/30 seconds later. (Image courtesy of Richard Szeliski.)

Cross-correlation works best when there is texture in the scene, resulting in windows containing significant brightness variation among the pixels. If one is looking at a uniform white wall, then the cross-correlation is going to be nearly the same for the different candidate matches q , and the algorithm is reduced to making a blind guess.

Suppose that the viewer has translational velocity T and angular velocity ω (which thus describe the **egomotion**). One can derive an equation relating the viewer velocities, the optical flow, and the positions of objects in the scene. In fact, assuming $f = 1$,

$$\begin{aligned} v_x(x, y) &= \left[-\frac{T_x}{Z(x, y)} - \omega_y + \omega_z y \right] - x \left[-\frac{T_z}{Z(x, y)} - \omega_x y + \omega_y x \right] \\ v_y(x, y) &= \left[-\frac{T_y}{Z(x, y)} - \omega_z x + \omega_x \right] - y \left[-\frac{T_z}{Z(x, y)} - \omega_x y + \omega_y x \right] \end{aligned}$$

where $Z(x, y)$ gives the z-coordinate of the scene point corresponding to the image point at (x, y) .

One can get a good intuition by considering the case of pure translation. In that case, the flow field becomes

$$v_x(x, y) = \frac{-T_x + xT_z}{Z(x, y)}, \quad v_y(x, y) = \frac{-T_y + yT_z}{Z(x, y)}$$

One can observe some interesting properties. Both components of the optical flow, $v_x(x, y)$ and $v_y(x, y)$, are zero at the point $x = T_x/T_z, y = T_y/T_z$. This point is called the **focus of expansion** of the flow field. Suppose we change the origin in the x - y plane to lie at the focus of expansion; then the expressions for optical flow take on a particularly simple form. Let (x', y') be the new coordinates defined by $x' = x - T_x/T_z, y' = y - T_y/T_z$. Then

$$v_x(x', y') = \frac{x' T_z}{Z(x', y')} \quad \dots \quad v_y(x', y') = \frac{y' T_z}{Z(x', y')}$$

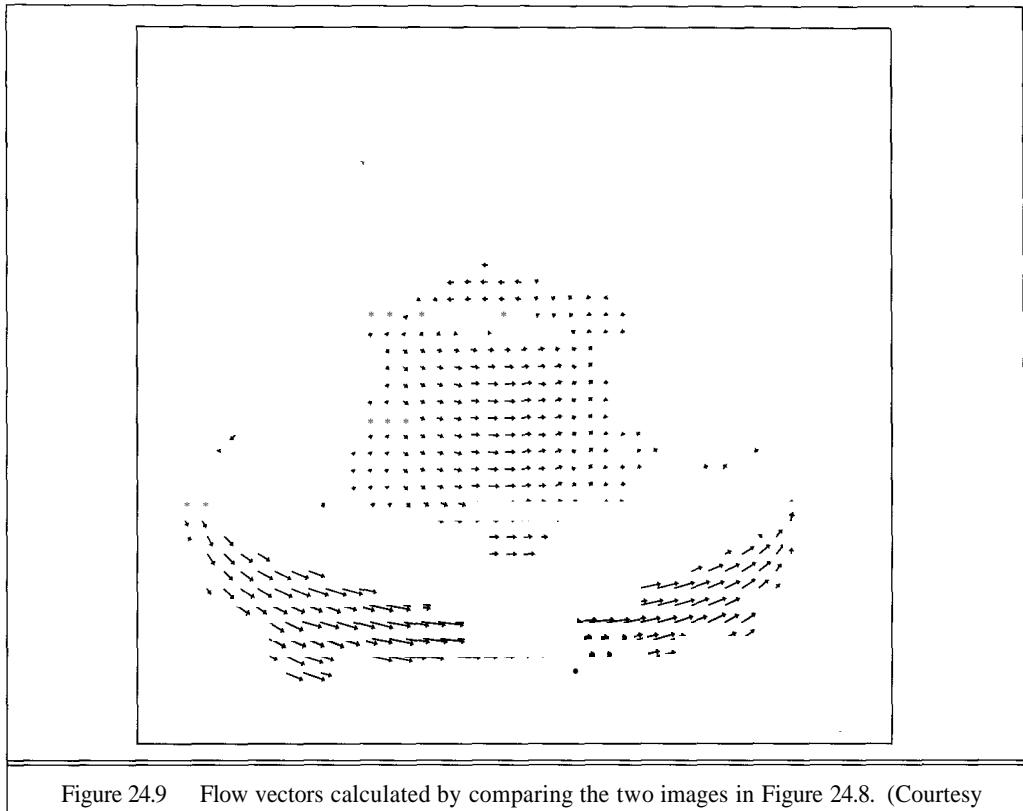


Figure 24.9 Flow vectors calculated by comparing the two images in Figure 24.8. (Courtesy of Joe Weber and Jitendra Malik.)

This equation has some interesting applications. Suppose you are a fly trying to land on wall, and you want to know the time to contact at the current velocity. This time is given by Z/T_z . Note that although the instantaneous optical flow field cannot provide either the distance Z or the velocity component T_z , it can provide the ratio of the two, and can therefore be used to control the landing approach. Experiments with real flies show that this is exactly what they use.

To recover depth, one should make use of multiple frames. If the camera is looking at a rigid *body*, the shape does not change from frame to frame and thus we are able to better deal with the inherently noisy optical flow measurements. Results from one such approach due to Tomasi and Kanade (1992) are shown in Figures 24.10 and 24.11.

Binocular stereopsis

The idea here is rather similar to motion parallax, except that instead of using images over time, we use two (or more) images separated in space, such as are provided by the forward-facing eyes of humans. Because a given feature in the scene will be in a different place relative to the z-axis of each image plane, if we superpose the two images, there will be a **disparity** in the location of

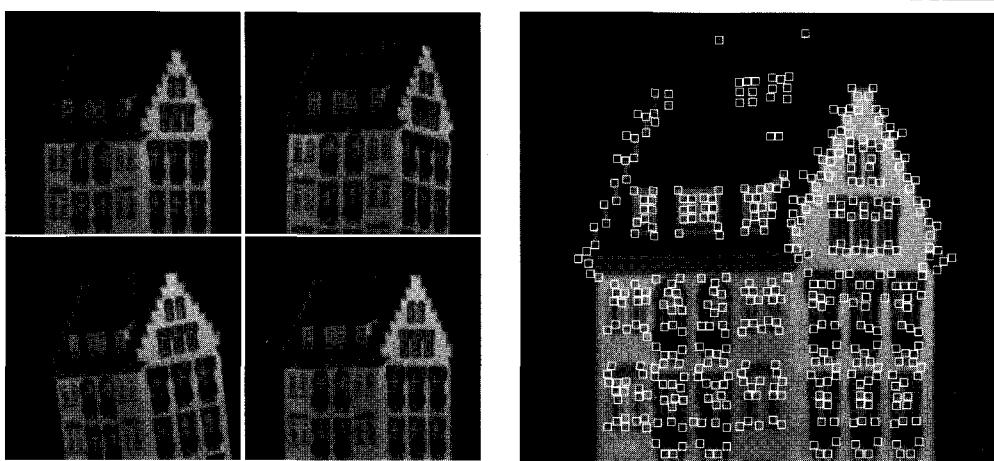


Figure 24.10 (a) Four frames from a video sequence in which the camera is moved and rotated relative to the object. (b) The first frame of the sequence, annotated with small boxes highlighting the features found by the feature detector. (Courtesy of Carlo Tomasi.)

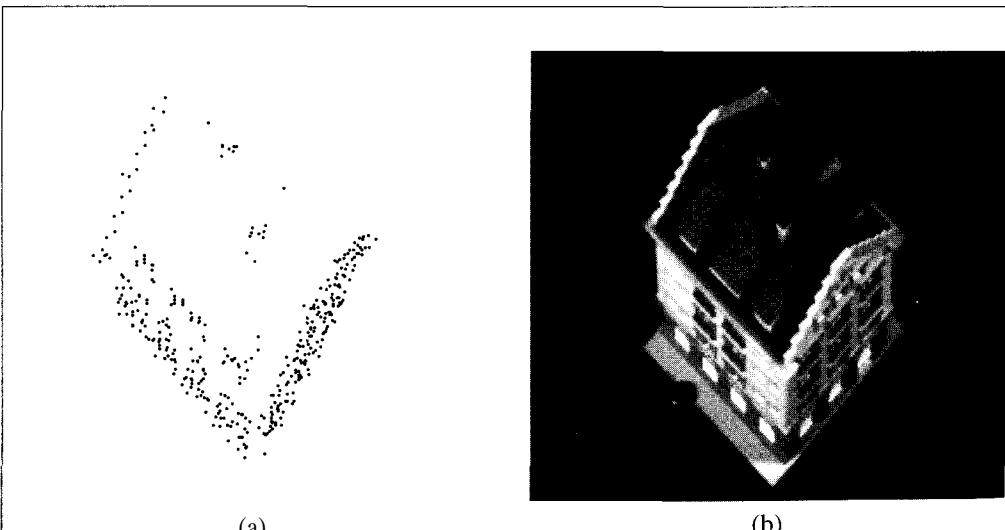


Figure 24.11 (a) 3-D reconstruction of the locations of the image features in Figure 24.10, shown from above. (b) The real house, taken from the same position.

the image feature in the two images. You can see this clearly in Figure 24.12, where the nearest point of the pyramid is shifted to the left in the right image and to the right in the left image.

Let us work out the geometrical relationship between disparity and depth. First, we will consider the case when both the eyes (or cameras) are looking forward with their optical axes

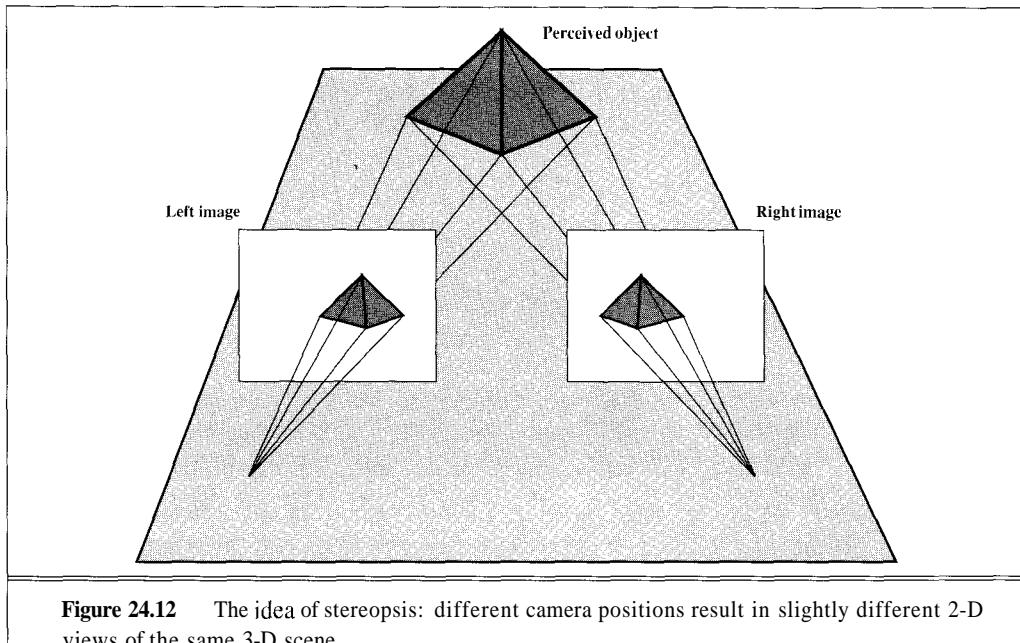


Figure 24.12 The idea of stereopsis: different camera positions result in slightly different 2-D views of the same 3-D scene.

parallel. The relationship of the right camera to the left camera is then just translation along the x -axis by an amount b , the baseline. We can use the optical flow equations from the previous section to compute the horizontal and vertical disparity as $H = v_x At$. $V = v_y At$, given that $T_x = b/At$ and $T_y = T_z = 0$. The rotational parameters ω_x , ω_y , and ω_z are zero. One obtains $H = b/Z$, $V = 0$. In words, the horizontal disparity is equal to the ratio of the baseline to the depth, and the vertical disparity is zero.

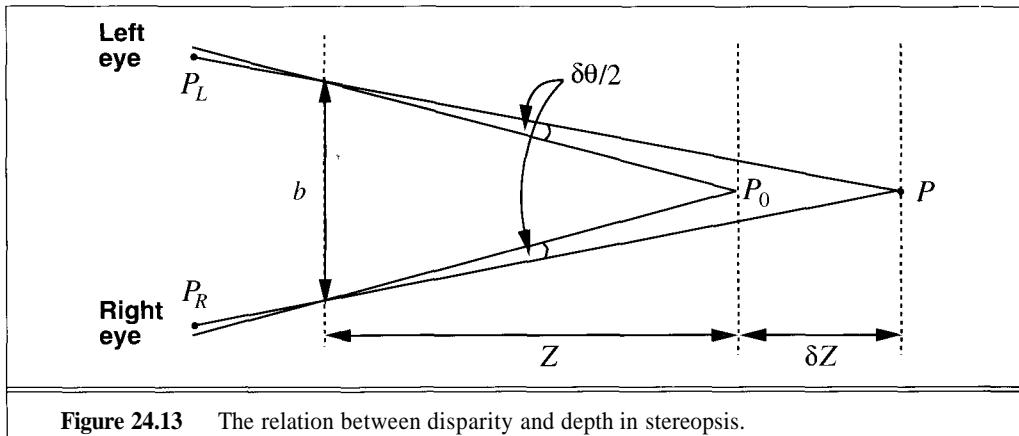
This is, of course, the simplest viewing geometry (relationship between the two cameras) that we could consider. Under normal viewing conditions, humans **fixate**; that is, there is some point in the scene at which the optical axes of the two eyes intersect. Figure 24.13 shows two eyes fixated at a point PQ , which is at a distance Z from the midpoint of the eyes. For convenience, we will compute the *angular* disparity, measured in radians. The disparity at the point of fixation PQ is zero. For some other point P in the scene that is $6Z$ further away, we can compute the angular displacements of the left and right images of P , which we will call PL and PR . If each of these is displaced by an angle $60/2$ relative to PQ , then the displacement between PL and PR , which is the disparity of P , is just 60 . From simple geometry, we have

$$\frac{-b}{\delta Z} = \frac{-b}{Z^2}$$

FIXATE

BASELINE

In humans, b (the **baseline**) is about 6 cm. Suppose that Z is about 100 cm. The smallest detectable 69 (corresponding to the pixel size) is about 5 seconds of arc, or 2.42×10^{-5} radians, giving a $6Z$ of about 0.4 mm. For $Z = 30$ cm (1 ft), we get the impressively small value $\delta Z = 0.036$ mm. Stating this in words, at a distance of 30 cm, humans can discriminate depths that differ by as little as 0.036 mm, enabling us to thread needles and the like.



Note that unlike the case of motion, we have assumed that we know the viewing geometry, or the relative orientation between the eyes. This is often a reasonable assumption. In the case of the eyes, the brain has commanded a particular state of the ocular muscles to move the eyes, and hence the positions of the eyes relative to the head are known. Similarly, in a binocular camera system, one knows the relative configuration.

EPIPOLAR LINES

Knowledge of the viewing geometry is very useful in trying to measure disparity. As in the case of optical flow, we can try to find corresponding points between the left and right images by maximizing some measure of similarity. However, one does not have to search in a two-dimensional region. Corresponding points must always lie along **epipolar lines** in the images (see Figure 24.14). These lines correspond to the intersections of an epipolar plane (the plane through a point in the scene and the nodal points of the two eyes) with the left and right image planes. Exploiting this epipolar constraint reduces an initially two-dimensional search to a one-dimensional one. Obviously determination of the epipolar lines requires a knowledge of the viewing geometry.

A simple-minded approach for finding disparity would be to search along epipolar lines looking to maximize the cross-correlation, just as in the case of optical flow. Given a point p_i in the left view, its corresponding point q_i in the right view is obtained by searching along the associated epipolar line in the other view. For each of the possible matches q , the cross-correlation between windows centered at p_i and q is computed. The corresponding point is declared to be the pixel q_i for which the cross-correlation is maximized.

One can do better by exploiting some additional constraints:

1. Uniqueness: a point in one image can correspond to at most one point in the other image. We say at most one, because it is possible that the point may be occluded in the other view.
2. Piecewise continuity of surfaces in the scene: the fact the world is usually piecewise continuous means that nearby points in the scene have nearby values of depth, and consequently of disparity, except across object boundaries and occlusions.

An example of a system that exploits these multiple constraints is the work of Belhumeur (1993). Belhumeur's results for an oblique view of a box (Figure 24.15) are shown in Figure 24.16. His

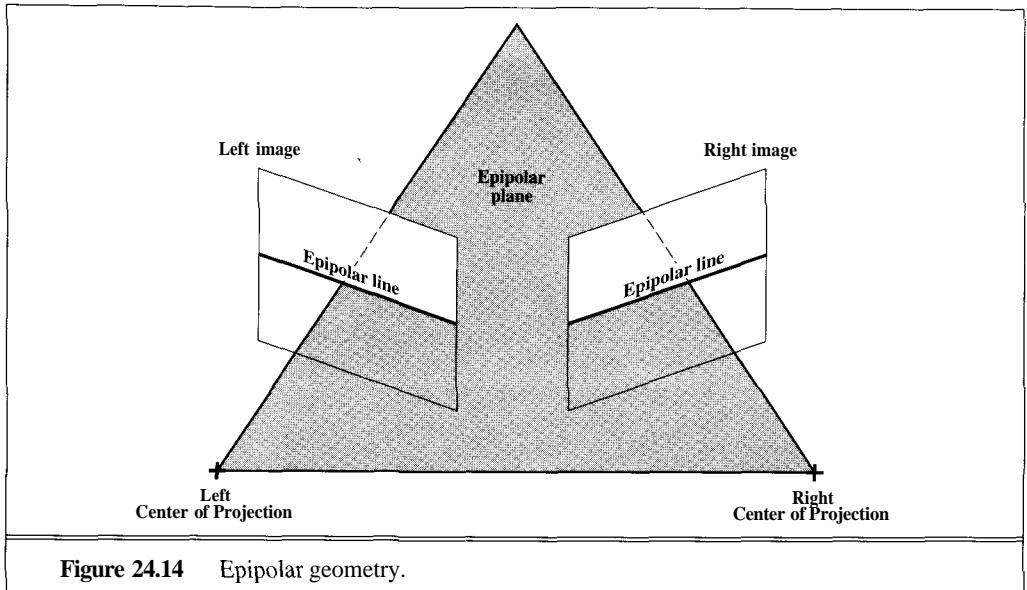


Figure 24.14 Epipolar geometry.

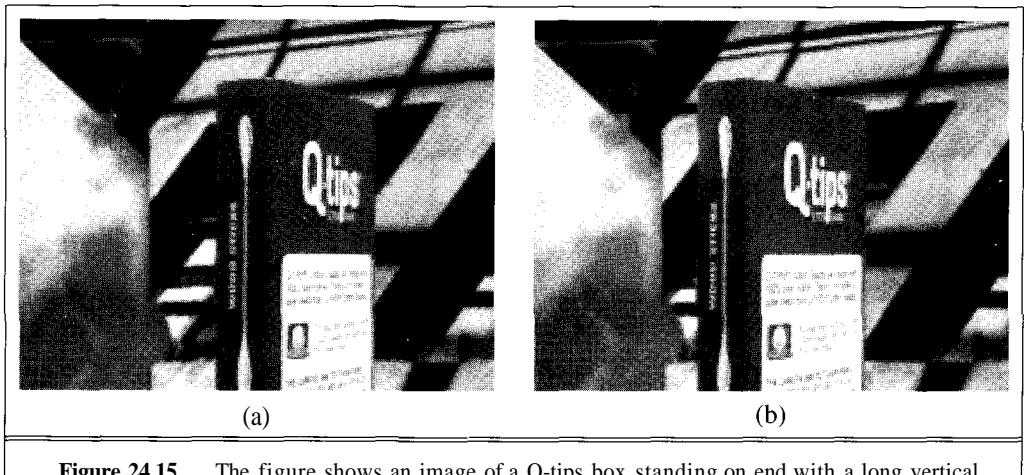


Figure 24.15 The figure shows an image of a Q-tips box standing on end with a long vertical crease protruding toward the camera. Behind the box is a flat surface.

algorithm uses the ordering constraint and deals with depth discontinuities where the windows at corresponding points in the image are not sampling corresponding patches in the scene. Because of occlusion in one of the views, there is a strip seen only in one eye. Note also the use of stereopsis for scene segmentation as demonstrated in Figure 24.16(c).

A standard criticism of area-based matching approaches is that they are susceptible to errors when corresponding patches in the two images do not look similar. This can happen for a variety of reasons: (1) the surface reflectance function has an appreciable specular component,

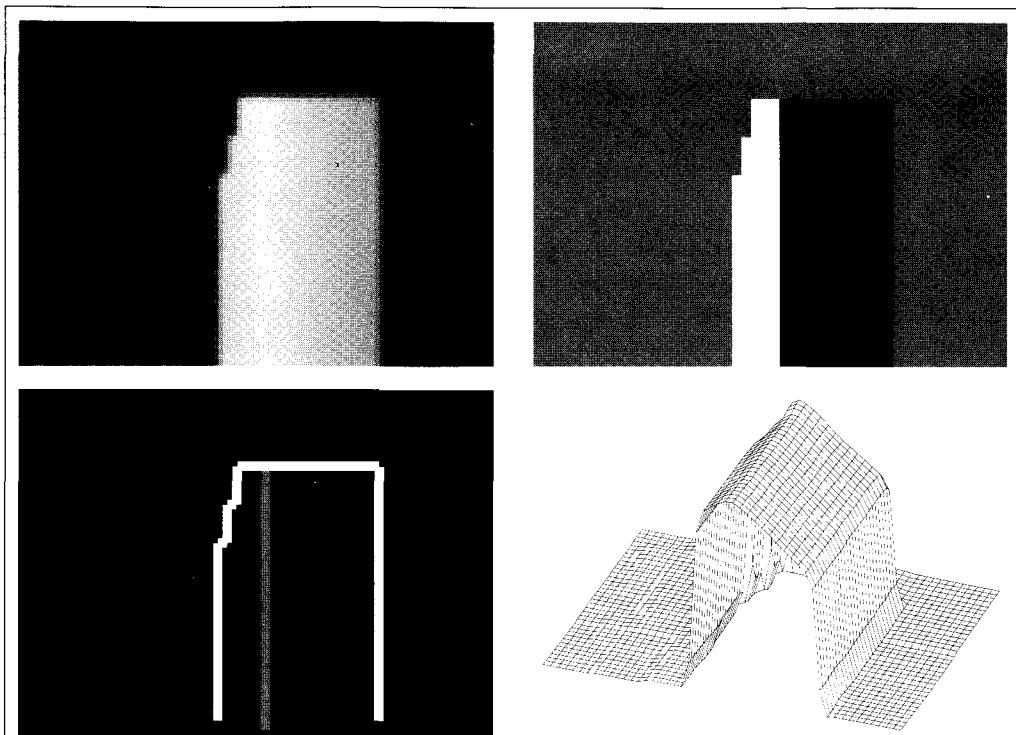


Figure 24.16 Results from processing the Q-tips stereo pair. (a) Image of depth. (b) Image of smoothed slope. (c) Object boundaries (white) and surface creases (gray). (d) Wire frame of depth. (Images courtesy of Peter Belhumeur.)

so the brightness of a point in the scene is a function of viewpoint; (2) there is a differing amount of foreshortening in the two views, because the patch makes different angles to the optical axes in the two views.

Another family of approaches is based on first finding edges and then looking for matches. Edges are deemed compatible if they are near enough in orientation and have the same sign of contrast across the edge. Corresponding edges are usually assumed to obey the same left-to-right ordering in each image, which allows one to restrict the number of possible matches and lends itself to efficient algorithms based on **dynamic programming**. With any edge-based approach, however, the resulting depth information is sparse, because it is available only at edge locations. Thus, a further step is needed to interpolate depth across surfaces in the scene.

Texture gradients

TEXTURE

Texture in everyday language is a property of surfaces associated with the tactile quality they suggest (texture has the same root as textile). In computational vision, it refers to a closely related concept, that of a spatially repeating pattern on a surface that can be sensed visually. Examples

include the pattern of windows on a building, the stitches on a sweater, the spots on a leopard's skin, blades of grass on a lawn, pebbles on a beach or a crowd of people in a stadium. Sometimes the arrangement is quite periodic, as in the stitches on a sweater; in other instances, as in pebbles on a beach the regularity is only in a statistical sense—the density of pebbles is roughly the same on different parts of the beach.

TEXELS

What we just said is true *in the scene*. In the image, the apparent size, shape, spacing, and so on, of the texture elements (the **texels**) do indeed vary, as illustrated in Figure 24.17. The tiles are identical in the scene. There are two main causes for this variation in the projected size and shape of the tiles:

1. Varying distances of the different texels from the camera. Recall that under perspective projection, distant objects appear smaller. The scaling factor is $1/Z$.
2. Varying foreshortening of the different texels. This is related to the orientation of the texel relative to the line of sight from the camera. If the texel is perpendicular to the line of sight, there is no foreshortening. The magnitude of the foreshortening effect is proportional to $\cos(\sigma)$, where σ is the slant of the plane of the texel.

GRADIENTS

After some mathematical analysis (Garding, 1992), one can compute expressions for the rate of change of various image texel features, such as area, foreshortening, and density. These **texture gradients** are functions of the surface shape as well as its slant and tilt with respect to the viewer.

To recover shape from texture, one can use a two-step process: (a) measure the texture gradients; (b) estimate the surface shape, slant, and tilt that would give rise to the measured texture gradients. We show the results of an algorithm developed by Malik and Rosenholtz(1994) in Figures 24.17 and 24.18.

Shading

Shading—variation in the intensity of light received from different portions of a surface in the scene—is determined by scene geometry and reflectance properties of the surfaces. In computer graphics, the objective is to determine the image brightness $I(x, y)$ given the scene geometry and reflectance properties. In computer vision, our hope might be to invert the process, that is, recover the scene geometry and reflectance properties given the image brightness $I(x, y)$. This has proved difficult to do in anything but the simplest cases.

Let us start with an example of a situation where we can in fact solve for shape from shading. Consider a Lambertian surface illuminated by a distant point light source. We will assume that the surface is distant enough from the camera so that we could use orthographic projection as an approximation to perspective projection. The image brightness is

$$I(x, y) = k\mathbf{n}(x, y) \cdot \mathbf{s}$$

where k is a scaling constant, \mathbf{n} is the unit surface normal vector, and \mathbf{s} is the unit vector in the direction of the light source. Because \mathbf{n} and \mathbf{s} are unit vectors, their dot product is just the cosine of the angle between them. Surface shape is captured in the variation of the surface normal \mathbf{n} along the surface. Let us assume that k and \mathbf{s} are known. Our problem then is to recover the surface normal $\mathbf{n}(x, y)$ given the image intensity $I(x, y)$.



Figure 24.17 A scene illustrating texture gradient. Assuming that the real texture is uniform allows recovery of the surface orientation. The computed surface orientation is indicated by overlaying a white circle and pointer, transformed as if the circle were painted on the surface at that point. (Image courtesy of Jitendra Malik and Ruth Rosenholtz.)

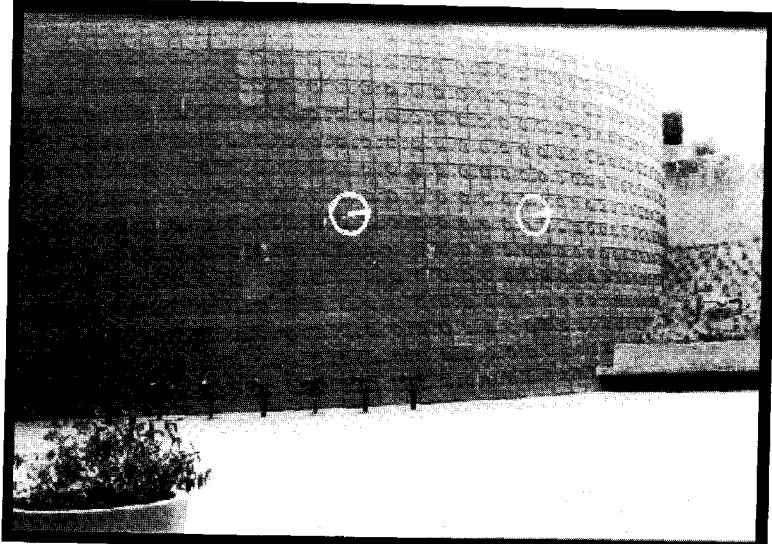


Figure 24.18 Recovery of shape from texture for a curved surface.

INTEGRABILITY

The first observation to make is that the problem of determining n , given the brightness I at a given pixel (x,y) , is underdetermined locally. We can compute the angle that n makes with the light source vector, but that only constrains it to lie on a certain cone of directions with axis s and apex angle $\theta = \cos^{-1}(I/k)$. To proceed further, note that n cannot vary arbitrarily from pixel to pixel. It corresponds to the normal vector of a smooth surface patch and consequently must also vary in a smooth fashion—the technical term for the constraint is **integrability**. Several different techniques have been developed to exploit this insight. One technique is simply to rewrite n in terms of the partial derivatives Z_x and Z_y of the depth $Z(x, y)$. This results in a partial differential equation for Z that can be solved to yield the depth $Z(x, y)$, given appropriate boundary conditions.

REFLECTANCE MAP

One can generalize the approach somewhat. It is not necessary for the surface to be Lambertian nor for the light source to be a point source. As long as one is able to determine the **reflectance map** $R(n)$, which specifies the brightness of a surface patch as a function of its surface normal n , essentially the same kind of techniques can be used.

The real difficulty comes in dealing with interreflections. If we consider a typical indoor scene, such as the objects inside an office, surfaces are illuminated not only by the light sources, but also by the light reflected from other surfaces in the scene that effectively serve as secondary light sources. These mutual illumination effects are quite significant. The reflectance map formalism completely fails in this situation—image brightness depends not just on the surface normal, but also on the complex spatial relationships among the different surfaces in the scene.

Humans clearly do get some perception of shape from shading, so this remains an interesting problem in spite of all these difficulties.

Contour

When we look at a line drawing, such as Figure 24.19, we get a vivid perception of 3-D shape and layout. How? After all, we saw earlier that there is an infinity of scene configurations that can give rise to the same line drawing. Note that we get even a perception of surface slant and tilt. It could be due to a combination of high-level knowledge about typical shapes as well as some low-level constraints.

LINE LABELLING

We will consider the qualitative knowledge available from a line drawing. As discussed earlier, lines in a drawing can have multiple significances (see Figure 24.6 and the accompanying text). The task of determining the actual significance of each line in an image is called **line labelling**, and was one of the first tasks studied in computer vision. For now, let us deal with a simplified model of the world where the objects have no surface marks and where the lines due to illumination discontinuities like shadow edges and specularities have been removed in some preprocessing step, enabling us to limit our attention to line drawings where each line corresponds either to a depth or orientation discontinuity.

LIMB

Each line then can be classified as either the projection of a **limb** (the locus of points on the surface where the line of sight is tangent to the surface) or as an **edge** (a surface normal discontinuity). Additionally, each edge can be classified as a convex, concave, or occluding edge. For occluding edges and limbs, we would like to determine which of the two surfaces bordering the curve in the line drawing is nearer in the scene. These inferences can be represented by giving each line one of 6 possible **line labels** as illustrated in Figure 24.20.

LINE LABELS

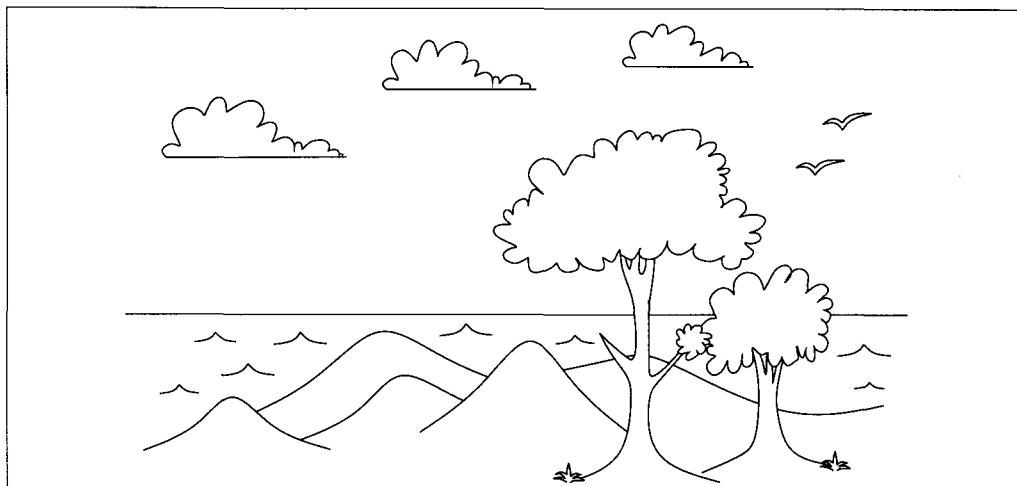


Figure 24.19 An evocative line drawing. (Courtesy of Isha Malik.)

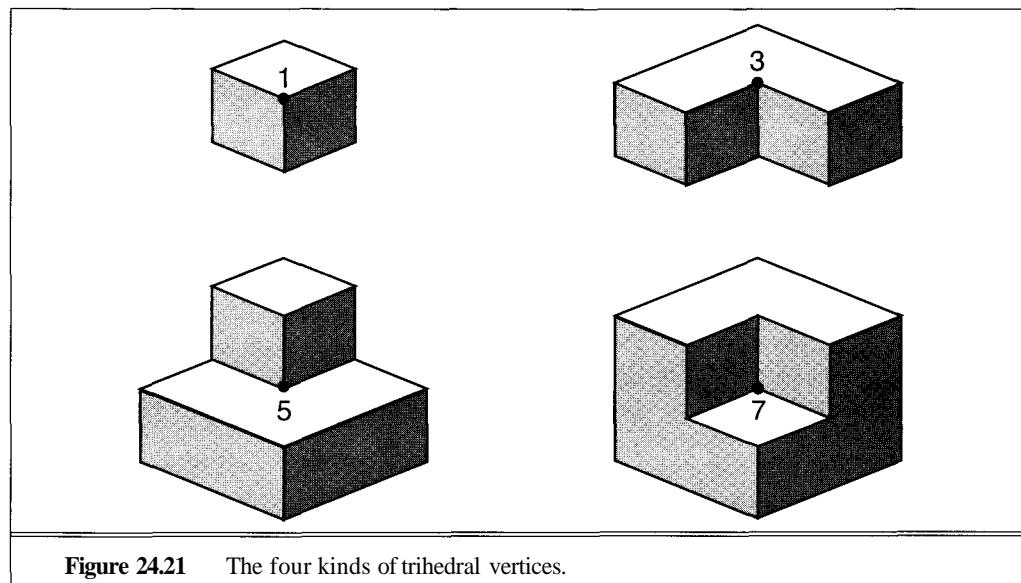
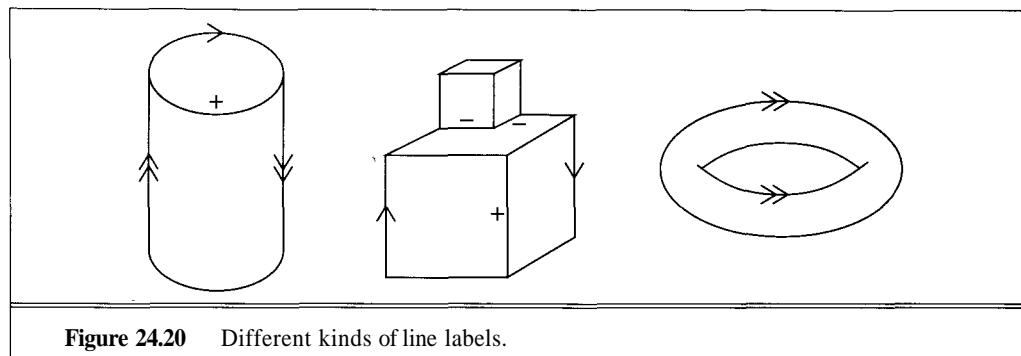
1. “+” and “−” labels represent convex and concave edges, respectively. These are associated with surface normal discontinuities where both surfaces that meet along the edge are visible.
2. A “←” or a “→” represents an occluding convex edge. When viewed from the camera, both surface patches that meet along the edge lie on the same side, one occluding the other. As one moves in the direction of the arrow, these surfaces are to the right.
3. A “↔” or a “→→” represents a limb. Here the surface curves smoothly around to occlude itself. As one moves in the direction of the twin arrows, the surface lies to the right. The line of sight is tangential to the surface for all points on the limb. Limbs move on the surface of the object as the viewpoint changes.

Of the 6^n combinatorially possible label assignments to the n lines in a drawing, only a small number are physically possible. The determination of these label assignments is the line labelling problem. Note that the problem only makes sense if the label is the same all the way along a line. This is not always true, because the label can change along a line for images of curved objects. In this section, we will deal only with polyhedral objects, so this is not a concern.

Huffman (1971) and Clowes (1971) independently attempted the first systematic approach to polyhedral scene analysis. Huffman and Clowes limited their analysis to scenes with opaque **triangular** solids—objects in which exactly three plane surfaces come together at each vertex. For scenes with multiple objects, they also ruled out object alignments that would result in a violation of the trihedral assumption, such as two cubes sharing a common edge. **Cracks**, that is, “edges” across which the tangent planes are continuous, were also not permitted. For the trihedral world, Huffman and Clowes made an exhaustive listing of all the different vertex types and the different ways in which they could be viewed under general viewpoint. The general viewpoint condition essentially ensures that if there is a small movement of the eye, none of the junctions changes character. For example, this condition implies that if three lines intersect in the image, the corresponding edges in the scene must also intersect.

TRIHDERAL

CRACKS



OCTANTS

The four ways in which three plane surfaces can come together at a vertex are shown in Figure 24.21. These cases have been constructed by taking a cube and dividing it into eight **octants**. We want to generate the different possible trihedral vertices at the center of the cube by filling in various octants. The vertex labeled 1 corresponds to 1 filled octant, 3 to 3 filled octants, and so on. Readers should convince themselves that these are indeed *all* the possibilities. For example, if one fills two octants in a cube, one cannot construct a valid trihedral vertex at the center. Note also that these four cases correspond to different combinations of convex and concave edges that meet at the vertex.

The three edges meeting at the vertex partition the surrounding space into eight octants. A vertex can be viewed from any of the octants not occupied by solid material. Moving the viewpoint within a single octant does not result in a picture with different junction types. The vertex labeled 1 in Figure 24.21 can be viewed from any of the remaining seven octants to give the junction labels in Figure 24.22.

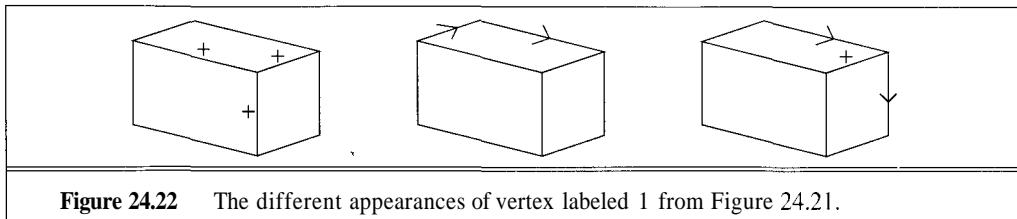


Figure 24.22 The different appearances of vertex labeled 1 from Figure 24.21.

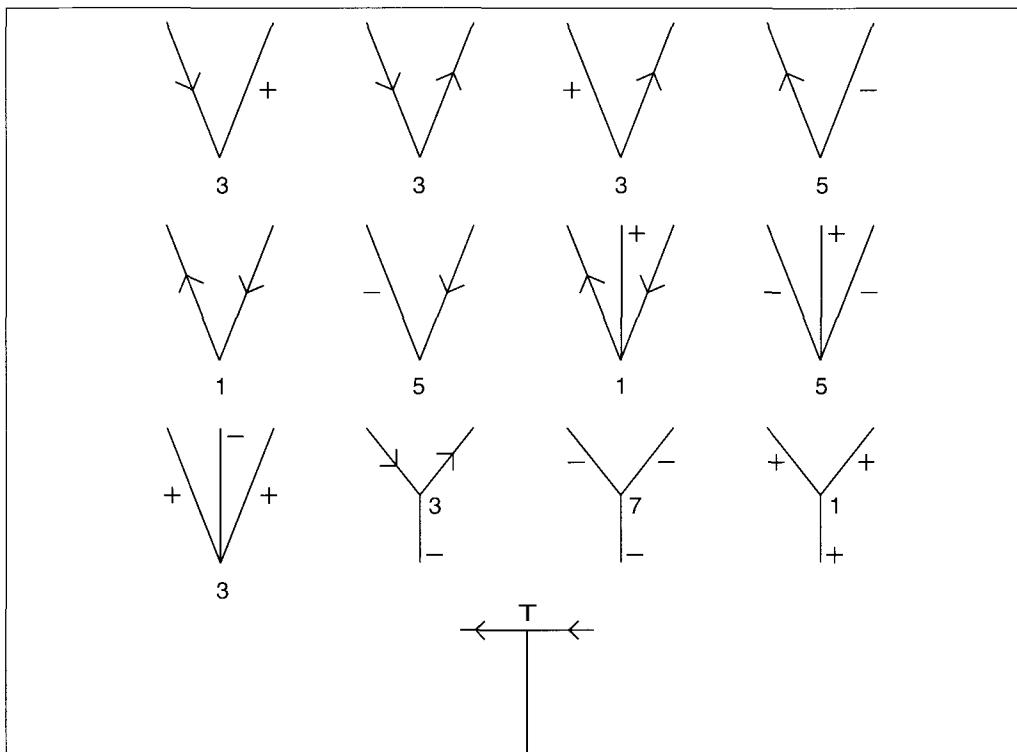


Figure 24.23 The Huffman–Clowes label set.

An exhaustive listing of the different ways each vertex can be viewed results in the possibilities shown in Figure 24.23. We get four different junction types which can be distinguished in the image: L-, Y-, arrow, and T-junctions. L-junctions correspond to two visible edges. Y- and arrow junctions correspond to a triple of edges—in a Y-junction none of the three angles is greater than π . T-junctions are associated with occlusion. When a nearer, opaque surface blocks the view of a more distant edge, one obtains a continuous edge meeting a half edge. The four T-junction labels correspond to the occlusion of four different types of edges.

When using this junction dictionary to find a labeling for the line drawing, the problem is to determine which junction interpretations are globally consistent. Consistency is forced by the rule that each line in the picture must be assigned one and only one label along its entire length.

Waltz (1975) proposed an algorithm for this problem (actually for an augmented version with shadows, cracks, and separably concave edges) that was one of the first applications of constraint satisfaction in AI (see Chapter 3). In the terminology of CSPs, the variables are the junctions, the values are labellings for the junctions, and the constraints are that each line has a single label. Although Kirousis and Papadimitriou (1988) have shown that the line-labelling problem for trihedral scenes is NP-complete, standard CSP algorithms perform well in practice.

Although the Huffman–Clowes–Waltz labeling scheme was limited to trihedral objects, subsequent work by Mackworth(1973) and Sugihara (1984) resulted in a generalization for arbitrary polyhedra and work by Malik (1987) for piecewise smooth curved objects.

24.5 USING VISION FOR MANIPULATION AND NAVIGATION

One of the principal uses of vision is to provide information for manipulating objects—picking them up, grasping, twirling, and so on—as well as navigating in a scene while avoiding obstacles.

The capability to use vision for these purposes is present in the most primitive of animal visual systems. Perhaps the evolutionary origin of the vision sense can be traced back to the presence of a photosensitive spot on one end of an organism that enabled it to orient itself toward (or away from) the light. Flies use vision based on optic flow to control their landing responses.

Mobile robots moving around in an environment need to know where the obstacles are, where free space corridors are available, and so on. More on this in Chapter 25.

Let us study the use of vision in driving in detail. Consider the tasks for the driver of the car in the bottom left corner of Figure 24.24.

1. Keep moving at a reasonable speed v_0 .
2. Lateral control—ensure that the vehicle remains securely within its lane, that is, keep $d_l = d_r$.
3. Longitudinal control—ensure that there is a safe distance d_2 between it and the vehicle in front of it.
4. Monitor vehicles in neighboring lanes (at distances d_1 and d_3) and be prepared for appropriate maneuvers if one of them decides to change lanes.

The problem for the driver is to generate appropriate steering, actuation or braking actions to best accomplish these tasks. Focusing specifically on lateral and longitudinal control, what information is needed for these tasks?

For lateral control, one needs to maintain a representation of the position and orientation of the car relative to the lane. The view of the road from a camera mounted on a car is shown in Figure 24.25. We detect edges corresponding to the lane marker segments and then fit smooth curves to these. The parameters of these curves carry information about the lateral position of the car, the direction it is pointing relative to the lane, and the curvature of the lane. This information, along with the dynamics of the car, is all that is needed by the steering control system. Note also that because from every frame to the next frame there is only a small change in the position

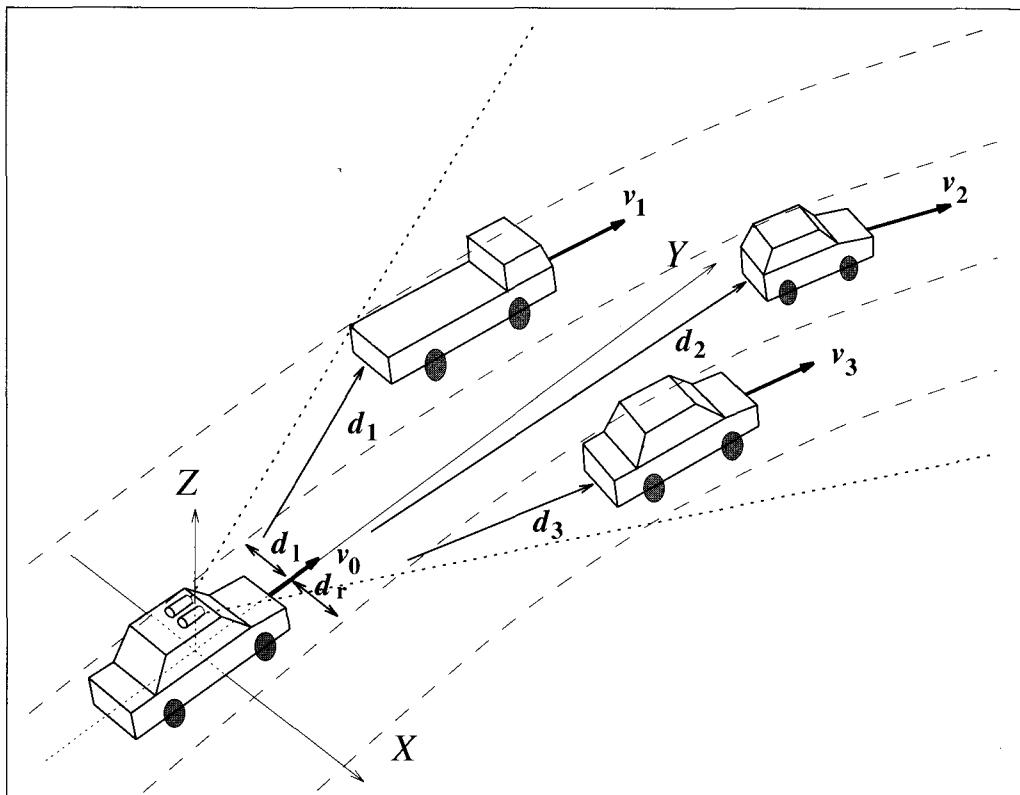


Figure 24.24 The information needed for visual control of a vehicle on a freeway.

of the projection of the lane in the image, one knows *where* to look for the lane markers in the image—in the figure, this is indicated by showing the search windows.

For longitudinal control, one needs to know distances to the vehicles in front. This can be accomplished using binocular stereopsis or optical flow. Both these approaches can be simplified by exploiting the domain constraints derived from the fact that one is driving on a planar surface. Using these techniques, Dickmanns and Zapp (1987) have demonstrated visually controlled car driving on freeways at high speeds. Pomerleau (1993) achieved similar performance using a neural network approach (see discussion on page 586).

The driving example makes one point very clear: *for a specific task, one does not need to recover all the information that in principle can be recovered from an image.* One does not need to recover the exact shape of every vehicle, solve for shape-from-texture on the grass surface adjacent to the freeway, and so on. The needs of the task require only certain kinds of information and one can gain considerable computational speed and robustness by recovering only that information and fully exploiting the domain constraints. Our purpose in discussing the general approaches in the previous section was that they form the basic theory, which one can specialize for the needs of particular tasks.



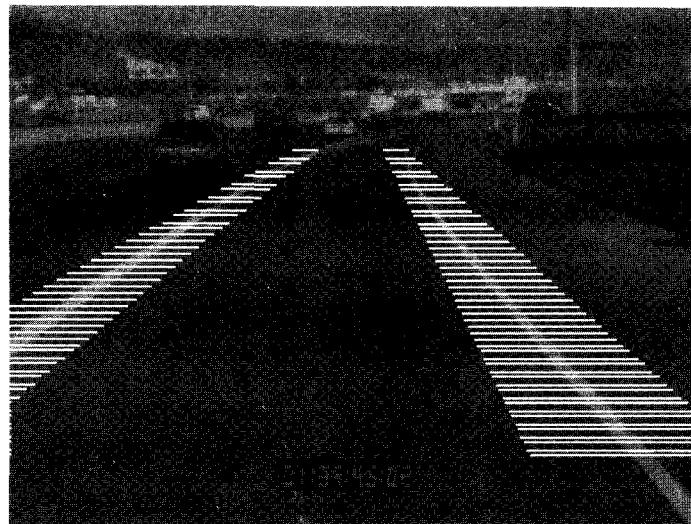


Figure 24.25 Image of a road taken from a camera inside the car.

24.6 OBJECT REPRESENTATION AND RECOGNITION

The object recognition problem can be defined as follows. Given

1. a scene consisting of one or more objects chosen from a collection of objects O_1, O_2, \dots, O_n known a priori, and
2. an image of the scene taken from an unknown viewer position and orientation,

determine the following:

1. Which of the objects O_1, O_2, \dots, O_n are present in the scene.
2. For each such object, determine its position and orientation relative to the viewer.

Obviously, this determination requires prior storage of suitable descriptions of the objects.

This problem is clearly of considerable industrial importance. For example, in an assembly process, the robot has to identify the different components and compute their positions and orientations in order to generate a grasping or pick-and-place strategy. In addition to the engineering standpoint, the problem is of great scientific interest. Humans have the impressive ability to recognize thousands of objects almost instantaneously even when the objects are partially occluded or presented in highly simplified line drawings.

The two fundamental issues that any object recognition scheme must address are the representation of the models and the matching of models to images.

GENERALIZED CYLINDERS

First, let us consider the representation problem. The two most popular representations of 3-D objects in computer vision have been polyhedral approximations and generalized cylinders. Polyhedral descriptions are general, but they get painfully long if a high accuracy is desired in modeling curved objects. They are also very cumbersome for users to input. **Generalized cylinders** (Figure 24.26) provide compact descriptions for a wide class of objects and have been used in a number of object recognition systems.

A generalized cylinder is defined by a planar cross section, an axis (which may be curved), and a sweeping rule which describes how the cross section changes along the axis. Many shapes can be built up using generalized cylinders as parts. Generalized cylinders are not always ideal for representing arbitrary objects; the object may have to be decomposed into many parts, each of which is a generalized cylinder. In such situations, there can be many alternative decompositions into parts with each part describable as a generalized cylinder in several ways. This leads to difficulties at matching time. In general, the problem of effective shape representation for curved objects is largely unsolved.

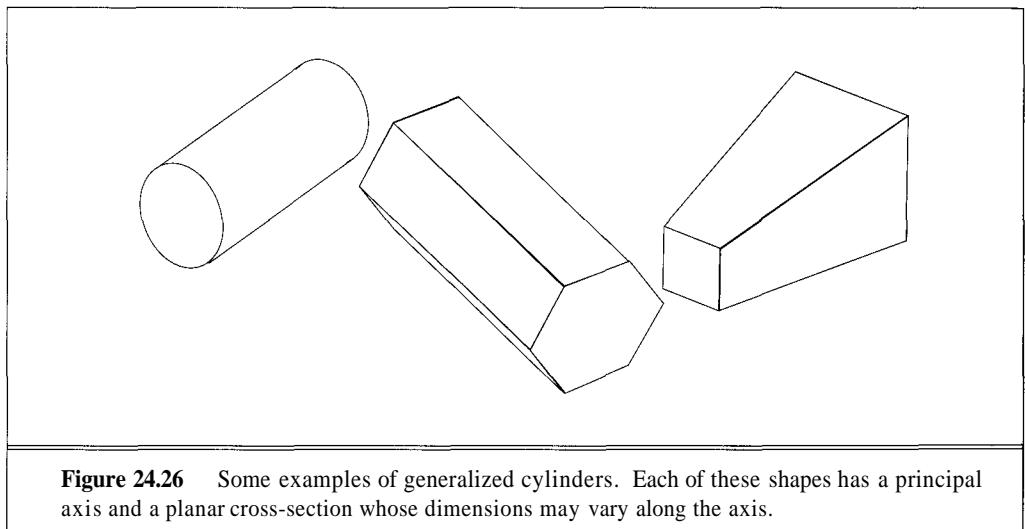


Figure 24.26 Some examples of generalized cylinders. Each of these shapes has a principal axis and a planar cross-section whose dimensions may vary along the axis.

The alignment method

We will consider one particular version of the problem in greater detail—we are asked to identify a three-dimensional object from its projection on the image plane. For convenience, the projection process is modelled as a scaled orthographic projection. We do not know the pose of the object—its position and orientation with respect to the camera.

The object is represented by a set of m features or distinguished points $\mu_1, \mu_2, \dots, \mu_m$ in three-dimensional space—perhaps vertices for a polyhedral object. These are measured in some coordinate system natural for the object. The points are then subjected to an unknown 3-D rotation R , followed by translation by unknown amount t and projection to give rise to image

feature points p_1, p_2, \dots, p_n on the image plane. In general, $n \neq m$ because some model points may be occluded. The feature detector in the image also may miss true features and mark false ones due to noise. We can express this as

$$p_i = \Pi(\mathbf{R}\mu_i + \mathbf{t})$$

for 3-D model point μ_i and corresponding image point p_i . Here Π denotes perspective projection or one of its approximations such as scaled orthographic projection. We can summarize this by the equation $p_i = Q\mu_i$ where Q is the (unknown) transformation that brings the model points in alignment with the image. Assuming the object is rigid, the transformation Q is the *same* for all the model points.

One can solve for Q given the 3-D coordinates of three model points and their 2-D projections. The intuition is as follows: one can write down equations relating the coordinates of P_i to those of μ_i . In these equations, the unknown quantities correspond to the parameters of the rotation matrix R and the translation vector t . If we have sufficiently many equations, we ought to be able to solve for Q . We will not give any proof here, but merely state the following result (Huttenlocher and Ullman, 1990):

Given three noncollinear points μ_1, μ_2 , and μ_3 in the model, and their projections on the image plane, p_1, p_2 , and p_3 under scaled orthographic projection, there exist exactly two transformations from the three-dimensional model coordinate frame to a two-dimensional image coordinate frame.

These transformations are related by a reflection around the image plane and can be computed by a simple closed-form solution. We will just assume that there exists a function FIND-TRANSFORM, as shown in Figure 24.27.

If we could identify the corresponding model features for three features in the image, we could compute Q , the pose of the object. The problem is that we do not know these correspondences. The solution is to operate in a generate-and-test paradigm. We have to guess an initial correspondence of an image triplet with a model triplet and use the function FIND-TRANSFORM to hypothesize Q . If the guessed correspondence was correct, then Q will be correct, and when applied to the remaining model points will result in prediction of the image points. If the guessed correspondence was incorrect, then Q will be incorrect, and when applied to the remaining model points would not predict the image points.

function FIND-TRANSFORM($p_1, p_2, p_3, \mu_1, \mu_2, \mu_3$) **returns** a transform Q such that

$$Q(\mu_1) = p_1$$

$$Q(\mu_2) = p_2$$

$$Q(\mu_3) = p_3$$

inputs: p_1, p_2, p_3 , image feature points
 μ_1, μ_2, μ_3 , model feature points

Figure 24.27 The definition of the transformation-finding process. We omit the algorithm (Huttenlocher and Ullman, 1990).

```

function ALIGN(Imagefeature points, Modelfeature points) returns a solution or failure
  loop do
    choose an untried triplet  $p_{i_1}, p_{i_2}, p_{i_3}$  from image
    if no untried triplets left then return failure
    while there are still untried model triplets left do
      choose an untried triplet  $\mu_{j_1}, \mu_{j_2}, \mu_{j_3}$  from model
       $Q \leftarrow \text{FIND-TRANSFORM}(p_{i_1}, p_{i_2}, p_{i_3}, \mu_{j_1}, \mu_{j_2}, \mu_{j_3})$ 
      if projection according to  $Q$  explains image then
        return (success,  $Q$ )
    end
  end

```

Figure 24.28 An informal description of the alignment algorithm.

This is the basis of the algorithm ALIGN, which seeks to find the pose for a given model and return failure otherwise (see Figure 24.28). The worst-case time complexity of the algorithm is proportional to the number of combinations of model triplets and image triplets—this gives the number of times Q has to be computed—times the cost of verification. This gives $\binom{n}{3} \binom{m}{3}$ times the cost of verification. The cost of verification is $m \log n$, as we must predict the image position of each of m model points, and find the distance to the nearest image point, a $\log n$ operation if the image points are arranged in an appropriate data structure. Thus, the worst-case complexity of the alignment algorithm is $O(m^4 n^3 \log n)$, where m and n are the number of model and image points, respectively.

One can lower the time complexity by a number of ideas. One simple technique is to hypothesize matches only between pairs of image and model points. Given two image points and the edges at these points, a third virtual point can be constructed by extending the edges and finding the intersection. This lowers the complexity to $O(m^3 n^2 \log n)$. Techniques based on using pose clustering in combination with randomization (Olson, 1994) can be used to bring the complexity down to $O(mn^3)$. Results from the application of this algorithm to the stapler image are shown in Figure 24.29.

Using projective invariants

Alignment using outline geometry and recognition is considered successful if outline geometry in an image can be explained as a perspective projection of the geometric model of the object. A disadvantage is that this involves trying each model in the model library, resulting in a recognition complexity proportional to the number of models in the library.

A solution is provided by using **geometric invariants** as the shape representation. These shape descriptors are *viewpoint invariant*, that is, they have the same value measured on the object or measured from a perspective image of the object, and are unaffected by object pose. The simplest example of a projective invariant is the "cross-ratio" of four points on a line, illustrated

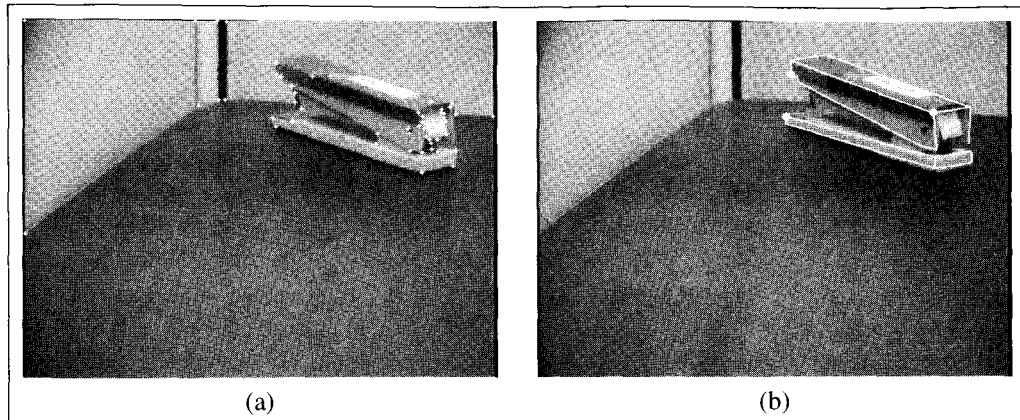


Figure 24.29 (a) Corners found in the stapler image. (b) Hypothesized reconstruction overlaid on the original image. (Courtesy of Clark Olson.)

in Figure 24.30. Under perspective projection, the ratios of distances are not preserved—think of the spacing of sleepers on an image of a receding railway track. The spacing is constant in the world, but decreases with distance from the camera in an image. However, the ratio of ratio of distances on a line is preserved, that is, it is the same measured on the object or in the image.

INDEX FUNCTIONS

Invariants are significant in vision because they can be used as **index functions**, so that a value measured in an image directly indexes a model in the library. To take a simple example, suppose there are three models $\{A, B, C\}$ in the library, each with a corresponding and distinct invariant value $\{I(A), I(B), I(C)\}$. Recognition proceeds as follows: After edge detection and grouping, invariants are measured from image curves. If a value $/ = I(B)$ is measured, then there is evidence that object B is present. It is not necessary to consider objects A and C any further. It may be that for a large model base, all invariants are not distinct (i.e., several models may share invariant values). Consequently, when an invariant measured in the image corresponds to a value in the library, a recognition hypothesis is generated. Recognition hypotheses corresponding to the same object are merged if compatible. The hypotheses are *verified* by back projecting the outline as in the alignment method. An example of object recognition using invariants is given in Figure 24.31.

Another advantage of invariant shape representation is that models can be acquired directly from images. It is not necessary to make measurements on the actual object, because the shape descriptors have the same value when measured in any image. This simplifies and facilitates automation of model acquisition. It is particularly useful in applications such as recognition from satellite images.

Although the two approaches to object recognition that we have described are useful in practice, it should be noted that we are far away from human competence. The generation of sufficiently rich and descriptive representations from images, segmentation and grouping to identify those features that belong together, and the matching of these to object models are difficult research problems under active investigation.

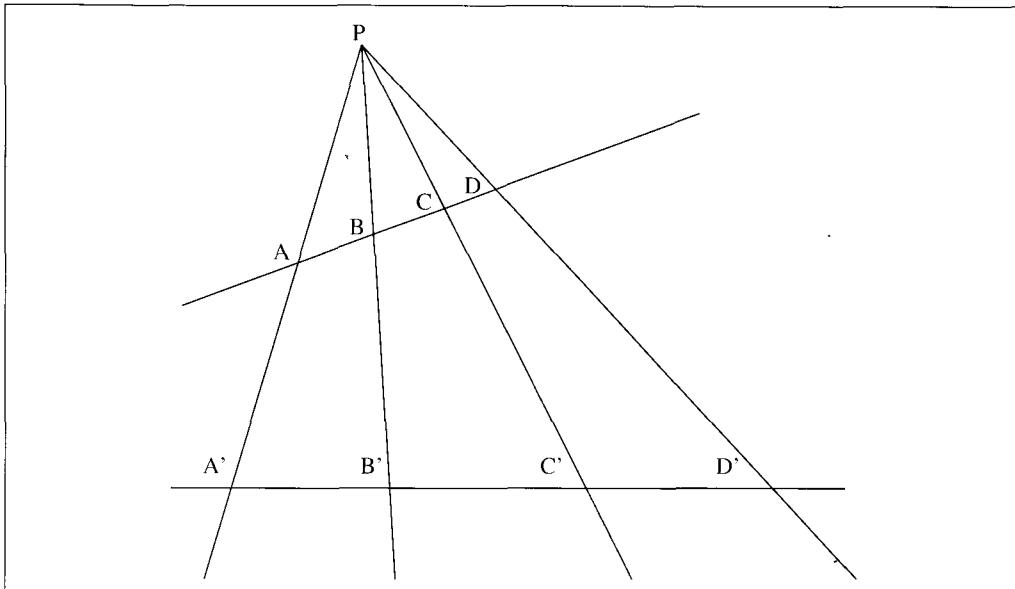


Figure 24.30 Invariance of the cross-ratio: $AD \cdot BC / AB \cdot CD = A'D' \cdot B'C' / A'B' \cdot C'D'$. Exercise 24.7 asks you to verify this fact.

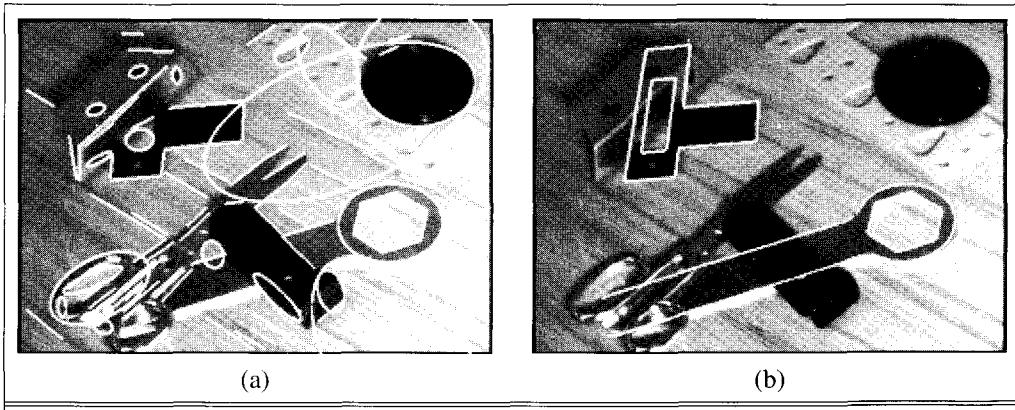


Figure 24.31 (a) A scene containing a number of objects, two of which also appear in the model library. These are recognized using invariants based on lines and conics. The image shows 100 fitted lines and 27 fitted conics superimposed in white. Invariants are formed from combinations of lines and conics, and the values index into a model library. In this case, there are 35 models in the library. Note that many lines are caused by texture, and that some of the conics correspond to edge data over only a small section. (b) The two objects from the library are recognized correctly. The lock striker plate is matched with a single invariant and 50.9% edge match, and the spanner with three invariants and 70.7% edge match. Courtesy of Andrew Zisserman.

24.7 SPEECH RECOGNITION

SPEECH
RECOGNITION

In this section, we turn from vision to another type of percept—speech. Speech is the dominant modality for communication between humans, and promises to be important for communication between humans and machines, if it can just be made a little more reliable. **Speech recognition** is the task of mapping from a digitally encoded acoustic signal to a string of words. **Speech understanding** is the task of mapping from the acoustic signal all the way to an interpretation of the meaning of the utterance. A speech understanding system must answer three questions:

1. What speech sounds did the speaker utter?
2. What words did the speaker intend to express with those speech sounds?
3. What meaning did the speaker intend to express with those words?

PHONES

To answer question 1, we have to first decide what a speech sound is. It turns out that all human languages use a limited repertoire of about 40 or 50 sounds, called **phones**. Roughly speaking, a phone is the sound that corresponds to a single vowel or consonant, but there are some complications: combinations of letters such as “th” and “ng” produce single phones, and some letters produce different phones in different contexts (for example, the “a” in *rat* and *rate*). Figure 24.32 lists all the phones in English with an example of each. Once we know what the possible sounds are, we need to characterize them in terms of features that we can pick out of the acoustic signal, such as the frequency or amplitude of the sound waves.

HOMOPHONES
SEGMENTATION

Question 2 is conceptually much simpler. You can think of it as looking up words in a dictionary that is arranged by pronunciation. We get a sequence of three phones, *[k]*, *[æ]*, and *[tʃ]*, and find in the dictionary that this is the pronunciation for the word “cat.” Two things make this difficult. The first is the existence of **homophones**, different words that sound the same, like “two” and “too.”¹ The second is **segmentation**, the problem of deciding where one word ends and the next begins. Anyone who has tried to learn a foreign language will appreciate this problem; at first all the words seem to run together. Gradually, one learns to pick out words from the jumble of sounds. In this case, first impressions are correct; a spectrographic analysis shows that in fluent speech, the words really *do* run together with no silence between them. We learn to identify word boundaries despite the lack of silence.

Question 3 we already know how to answer—use the parsing and analysis algorithms described in Chapter 22. Some speech understanding systems extract the most likely string of words and pass them directly to an analyzer. Other systems have a more complex control structure that considers multiple possible word interpretations so that understanding can be achieved even if some individual words are not recognized correctly.

We will shortly define a model that answers questions 1 and 2, but first we will explain a little about how the speech signal is represented.

¹ It is also true that one word can be pronounced several ways—you say tow-may-tow and I say tow-mah-tow. This makes it more tedious to construct the pronunciation dictionary, but it does not make it any harder to look up a word.

Vowels		Consonants B-N		Consonants P-Z	
Phone	Example	Phone	Example	Phone	Example
[iy]	beat	[b]	bet	[p]	pet
[ih]	bit	[ch]	Chet	[r]	rat
[ey]	bet	[d]	debt	[s]	set
[æ]	bat	[f]	fat	[sh]	shoe
[ah]	but	[g]	get	[t]	ten
[ao]	<u>bought</u>	[hh]	hat	[th]	thick
[ow]	boat	[hv]	high	[dh]	that
[uh]	book	[Uh]	jet	[dx]	butter
[ux]	beauty	[k]	kick	[v]	vet
[er]	Bert	[l]	let	[w]	wet
[ay]	<u>buy</u>	[el]	bottle	[wh]	which
[oy]	<u>boy</u>	[m]	met	[y]	<u>yet</u>
[axr]	diner	[em]	bottom	[z]	zoo
[aw]	down	[n]	net	[zh]	measure
[ax]	about	[en]	button		
[ix]	roses	[ng]	<u>sing</u>		
[aa]	cot	[eng]	<u>Washington</u>	[‐]	(silence)

Figure 24.32 The DARPA phonetic alphabet, listing all the phones used in English. There are several alternative notations, including an International Phonetic Alphabet (IPA), which contains the phones in all known languages.

Signal processing

Sound is an analog energy source. When a sound wave strikes a microphone, it is converted to an electrical current, which can be passed to an analog-to-digital converter to yield a stream of bits representing the sound. We have two choices in deciding how many bits to keep. First, the **sampling rate** is the frequency with which we look at the signal. For speech, a sampling rate between 8 and 16 KHz (i.e., 8 to 16,000 times per second) is typical. Telephones deliver only about 3 KHz. Second, the **quantization factor** determines the precision to which the energy at each sampling point is recorded. Speech recognizers typically keep 8 to 12 bits. That means that a low-end system, sampling at 8 KHz with 8-bit quantization, would require nearly half a megabyte per minute of speech. This is a lot of information to manipulate, and worse, it leaves us very far from our goal of discovering the phones that make up the signal.

The first step in coming up with a better representation for the signal is to group the samples together into larger blocks called **frames**. This makes it possible to analyze the whole frame for the appearance of speech phenomena such as a rise or drop in frequency, or a sudden onset or cessation of energy. A frame length of about 10 msec (i.e., 80 samples at 8 KHz) seems to be long enough so that most such phenomena can be detected and that few short-duration phenomena will be missed. Within each frame, we represent what is happening with a vector of **features**. For example, we might want to characterize the amount of energy at each of several frequency ranges.

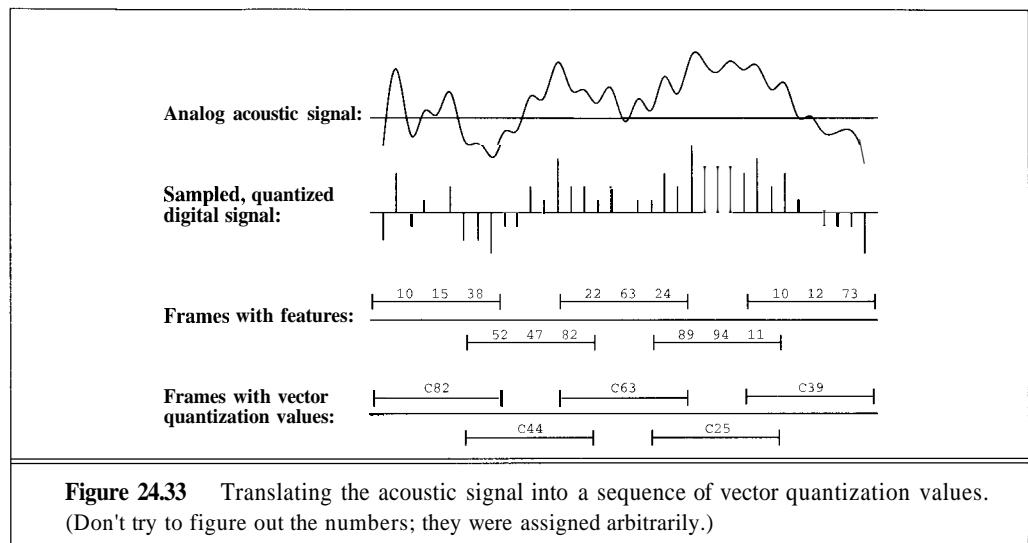
SAMPLING RATE

QUANTIZATION FACTOR

FRAMES

FEATURES

Other important features include overall energy in a frame, and the difference from the previous frame. Picking out features from a speech signal is like listening to an orchestra and saying "here the French horns are playing loud and the violins are playing softly." Breaking the sound down into components like this is much more useful than leaving it as a single undifferentiated sound source. Figure 24.33 shows frames with a vector of three features. Note that the frames overlap; this prevents us from losing information if an important acoustic event just happens to fall on a frame boundary.



VECTOR QUANTIZATION

The final step in many speech signal processing systems is **vector quantization**. If there are n features in a frame, we can think of this as an n -dimensional space containing many points. Vector quantization divides this n -dimensional space into, say, 256 regions labelled C1 through C256. Each frame can then be represented with a single label rather than a vector of n numbers. So we end up with just one byte per frame, which is about a 100-fold improvement over the original half megabyte per minute. Of course, some information is lost in going from a feature vector to a label that summarizes a whole neighborhood around the vector, but there are automated methods for choosing an optimal quantization of the feature vector space so that little or no inaccuracy is introduced (Jelinek, 1990).

There are two points to this whole exercise. First, we end up with a representation of the speech signal that is compact. But more importantly, we have a representation that is likely to encode features of the signal that will be useful for word recognition. A given speech sound can be pronounced so many ways: loud or soft, fast or slow, high-pitched or low, against a background of silence or noise, and by any of millions of different speakers each with different accents and vocal tracts. Signal processing hopes to capture enough of the important features so that the commonalities that define the sound can be picked out from this backdrop of variation. (The dual problem, **speaker identification**, requires one to focus on the variation instead of the commonalities in order to decide who is speaking.)

Defining the overall speech recognition model

Speech recognition is the diagnostic task of recovering the words that produce a given acoustic signal. It is a classic example of reasoning with uncertainty. We are uncertain about how well the microphones (and digitization hardware) have captured the actual sounds, we are uncertain about which phones would give rise to the signal, and we are uncertain about which words would give rise to the phones. As is often the case, the diagnostic task can best be approached with a causal model—the words cause the signal. We can break this into components with Bayes' rule:

$$P(\text{words}|\text{signal}) = \frac{P(\text{words})P(\text{signal}|\text{words})}{P(\text{signal})}$$

LANGUAGE MODEL
ACOUSTIC MODEL

Given a *signal*, our task is to find the sequence of *words* that maximizes $P(\text{words}|\text{signal})$. Of the three components on the right-hand side, $P(\text{signal})$ is a normalizing constant that we can ignore. $P(\text{words})$ is known as the **language model**. It is what tells us, when we are not sure if we heard "bad boy" or "pad boy" that the former is more likely. Finally, $P(\text{signal}|\text{words})$ is the **acoustic model**. It is what tells us that "cat" is very likely to be pronounced [kæt].

The language model: $P(\text{words})$

In *Take the Money and Run*, a bank teller interprets Woody Alien's sloppily written hold-up note as saying "I have a gub." A better language model would have enabled the teller to determine that the string "I have a gun" has a much higher prior probability of being on a hold-up note. That makes "gun" a better interpretation even if $P(\text{signal}|gub)$ is a little higher than $P(\text{signal}|gun)$. The language model should also tell us that "I have a gun" is a much more probable utterance than "gun a have I."

At first glance, the language model task seems daunting. We have to assign a probability to each of the (possibly infinite) number of strings. Context-free grammars are no help for this task, but probabilistic context-free grammars (PCFGs) are promising. Unfortunately, as we saw in Chapter 22, PCFGs aren't very good at representing contextual effects. In this section, we approach the problem using the standard strategy of defining the probability of a complex event as a product of probabilities of simpler events. Using the notation $w_1 \cdots w_n$ to denote a string of n words and w_i to denote the i th word of the string, we can write an expression for the probability of a string as follows:²

$$\begin{aligned} P(w_1 \cdots w_n) &= P(w_1) P(w_2|w_1) P(w_3|w_1w_2) \cdots P(w_n|w_1 \cdots w_{n-1}) \\ &= \prod_{i=1}^n P(w_i|w_1 \cdots w_{i-1}) \end{aligned}$$

BIGRAM

Most of these terms are quite complex and difficult to estimate or compute, and they have no obvious relation to CFGs or PCFGs. Fortunately, we can approximate this formula with something simpler and still capture a large part of the language model. One simple, popular, and effective approach is the **bigram** model. This model approximates $P(w_i|w_1 \cdots w_{i-1})$ with $P(w_i|w_{i-1})$. In other words, it says that the probability of any given word is determined solely by the previous word in the string. The probability of a complete string is given by

$$P(w_1 \cdots w_n) = P(w_1) P(w_2|w_1) P(w_3|w_2) \cdots P(w_n|w_{n-1}) = \prod_{i=1}^n P(w_i|w_{i-1})$$

² Actually, it would be better if all the probabilities were conditioned on the situation. Few speech recognizers do this, however, because it is difficult to formalize what counts as a situation.

Word	Unigram count	Previous words									
		OF	IN	IS	ON	TO	FROM	THAT	WITH	LINE	VISION
THE	367	179	143	44	44	65	35	30	17	0	0
ON	69	0	0	1	0	0	0	0	0	0	0
OF	281	0	0	2	0	1	0	3	0	4	0
TO	212	0	0	19	0	0	0	0	0	0	1
IS	175	0	0	0	0	0	0	13	0	1	3
A	153	36	36	33	23	21	14	3	15	0	0
THAT	124	0	3	18	0	1	0	0	0	0	0
WE	105	0	0	0	1	0	0	12	0	0	0
LINE	17	1	0	0	0	1	0	0	0	0	0
VISION	13	3	0	0	1	0	1	0	0	0	0

Figure 24.34 A partial table of unigram and bigram counts for the words in this chapter. The word "the" appears 367 times in all (out of 17613 total words), the bigram "of the" appeared 179 times (or about 1%), and the bigram "in the" appeared 143 times. It turns out these are the only two bigrams that occur more than 100 times.

A big advantage of the bigram model is that it is easy to train the model by counting the number of times each word pair occurs in a representative corpus of strings and using the counts to estimate the probabilities. For example, if "a" appears 10,000 times in the training corpus and it is followed by "gun" 37 times, then $\hat{P}(\text{gun}_i | a_{i-1}) = 37/10,000$, where by \hat{P} we mean the estimated probability. After such training one would expect "I have" and "a gun" to have relatively high estimated probabilities, while "I has" and "an gun" would have low probabilities. One problem is that the training corpus would probably not contain "gub" at all and more importantly, it would be missing many valid English words as well, so these words would be assigned an estimated probability of zero. Therefore, it is customary to set aside a small portion of the probability distribution for words that do not appear in the training corpus. Figure 24.34 shows some bigram counts derived from the words in this chapter.

TRIGRAM

It is possible to go to a **trigram** model that provides values for $P(w_i | w_{i-1} w_{i-2})$. This is a more powerful language model, capable of determining that "ate a banana" is more likely than "ate a bandana." The problem is that there are so many more parameters in trigram models that it is hard to get enough training data to come up with accurate probability estimates. A good compromise is to use a model that consists of a weighted sum of the trigram, bigram, and unigram (i.e., word frequency) models. The model is defined by the following formula (with $c_1 + c_2 + c_3 = 1$):

$$P(w_1 \dots w_n) = c_1 P(w_i) + c_2 P(w_i | w_{i-1}) + c_3 P(w_i | w_{i-1} w_{i-2})$$

Bigram or trigram models are not as sophisticated as PCFGs, but they account for local context-sensitive effects better, and manage to capture some local syntax. For example, the fact that the word pairs "I has" and "man have" get low scores is reflective of subject-verb agreement. The problem is that these relationships can only be detected locally: "the man have" gets a low score, but "the man over there have" is not penalized.

The acoustic model: $P(\text{signal}|\text{words})$

The acoustic model is responsible for saying what sounds will be produced when a given string of words is uttered. We divide the model into two parts. First, we show how each word is described as a sequence of phones, and then we show how each phone relates to the vector quantization values extracted from the acoustic signal.

Some words have very simple pronunciation models. The word "cat," for example, is always pronounced with the three phones fk a? t]. There are, however, two sources of phonetic variation. First, different dialects have different pronunciations. The top of Figure 24.35 gives an example of this: for "tomato," you say [tow mey tow] and I say [tow maa tow]. The alternative pronunciations are specified as a **Markov model**. In general, a Markov model is a way of describing a process that goes through a series of states. The model describes all the possible paths through the state space and assigns a probability to each one. The probability of transitioning from the current state to another one depends only on the current state, not on any prior part of the path. (This is the Markov property mentioned in Chapter 17.)

The top of Figure 24.35 is a Markov model with seven states (circles), each corresponding to the production of a phone. The arrows denote allowable transitions between states, and each transition has a probability associated with it.³ There are only two possible paths through the model, one corresponding to the phone sequence [t ow m ey t ow] and the other to [t ow m aa t ow]. The probability of a path is the product of the probabilities on the arcs that make up the path. In this case, most of the arc probabilities are 1 and we have

$$P([\text{towmeytow}] | \text{"tomato"}) = P([\text{towmaatow}] | \text{"tomato"}) = 0.5$$

COARTICULATION

The second source of phonetic variation is **coarticulation**. Remember that speech sounds are produced by moving the tongue and jaw and forcing air through the vocal tract. When the speaker is talking slowly and deliberately, there is time to place the tongue in just the right spot before producing a phone. But when the speaker is talking quickly (or sometimes even at a normal pace), the movements slur together. For example, the [t] phone is produced with the tongue at the top of the mouth, whereas the [ow] has the tongue near the bottom. When spoken quickly, the tongue often goes to an intermediate position, and we get [tah] rather than [tow]. The bottom half of Figure 24.35 gives a more complicated pronunciation model for "tomato" that takes this coarticulation effect into account. In this model there are four distinct paths and we have

$$P([\text{towmeytow}] | \text{"tomato"}) = P([\text{towmaatow}] | \text{"tomato"}) = 0.1$$

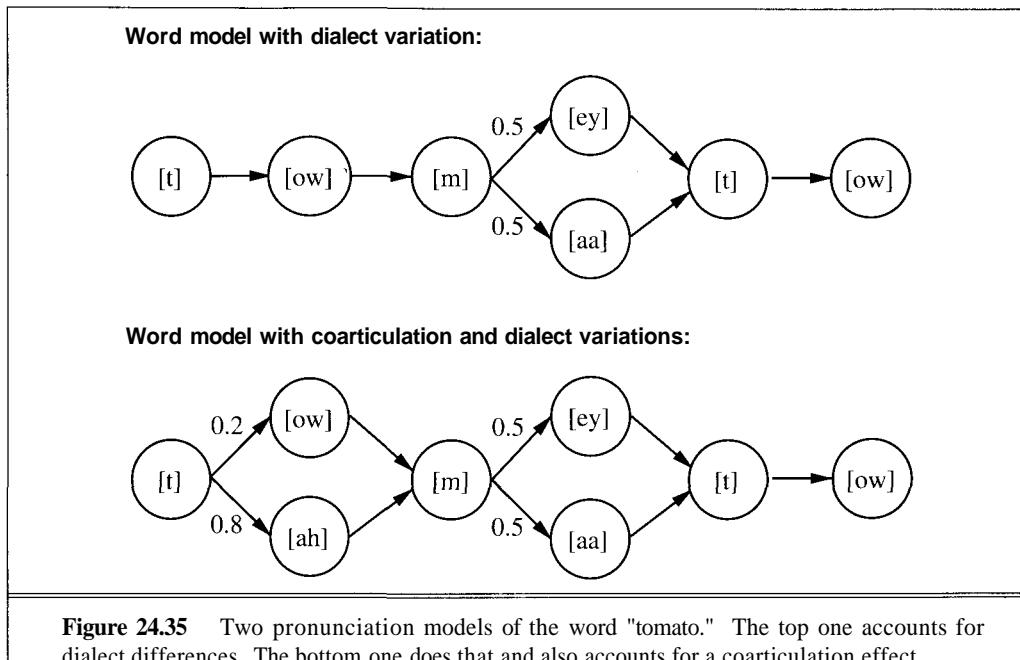
$$P([\text{tahmeytow}] | \text{"tomato"}) = P([\text{tahmaatow}] | \text{"tomato"}) = 0.4$$

HIDDEN MARKOV MODEL

Similar models would be constructed for every word we want to be able to recognize. Now if the speech signal were a list of phones, then we would be done with the acoustic model. We could take a given input signal (e.g., [towmeytow]) and compute $P(\text{signal}|\text{words})$ for various word strings (e.g., "tomato," "toe may tow," and so on). We could then combine these with $P(\text{words})$ values taken from the language model to arrive at the *words* that maximize $P(\text{words}|\text{signal})$.

Unfortunately, signal processing does not give us a string of phones. So all we can do so far is maximize $P(\text{words}|\text{phones})$. Figure 24.36 shows how we can compute $P(\text{signal}|\text{phone})$ using a model called a **hidden Markov model** or **HMM**. The model is for a particular phone,

³ Arcs with probability 1 are unlabelled in Figure 24.35. The 0.5 numbers are estimates based on the two authors' preferred pronunciations.



[m], but all phones will have models with similar topology. A hidden Markov model is just like a regular Markov model in that it describes a process that goes through a sequence of states. The difference is that in a regular Markov model, the output is a sequence of state names, and because each state has a unique name, the output uniquely determines the path through the model. In a hidden Markov model, each state has a probability distribution of possible outputs, and the same output can appear in more than one state.⁴ HMMs are called hidden models because the true state of the model is hidden from the observer. In general, when you see that an HMM outputs some symbol, you can't be sure what state the symbol came from.

Suppose our speech signal is processed to yield the sequence of vector quantization values [C1,C4,C6]. From the HMM in Figure 24.36, we can compute the probability that this sequence was generated by the phone [m] as follows. First, we note that there is only one path through the model that could possibly generate this sequence: the path from Onset to Mid to End, where the output labels from the three states are C1, C4, and C6, respectively. By looking at the probabilities on the transition arcs, we see that the probability of this path is $0.7 \times 0.1 \times 0.6$ (these are the values on the three horizontal arrows in the middle of the Figure 24.36). Next, we look at the output probabilities for these states to see that the probability of [C1,C4,C6] given this path is $0.5 \times 0.7 \times 0.5$ (these are the values for $P(C1|Onset)$, $P(C4|Mid)$ and $P(C6|End)$, respectively). So the probability of [C1,C4,C6] given the [m] model is

$$P([C1, C4, C6] | [m]) = (0.7 \times 0.1 \times 0.6) \times (0.5 \times 0.7 \times 0.5) = 0.00735$$

⁴ Note that this means that the "tomato" models in Figure 24.35 are actually hidden Markov models, because the same output (e.g., [t]) appears on more than one state.

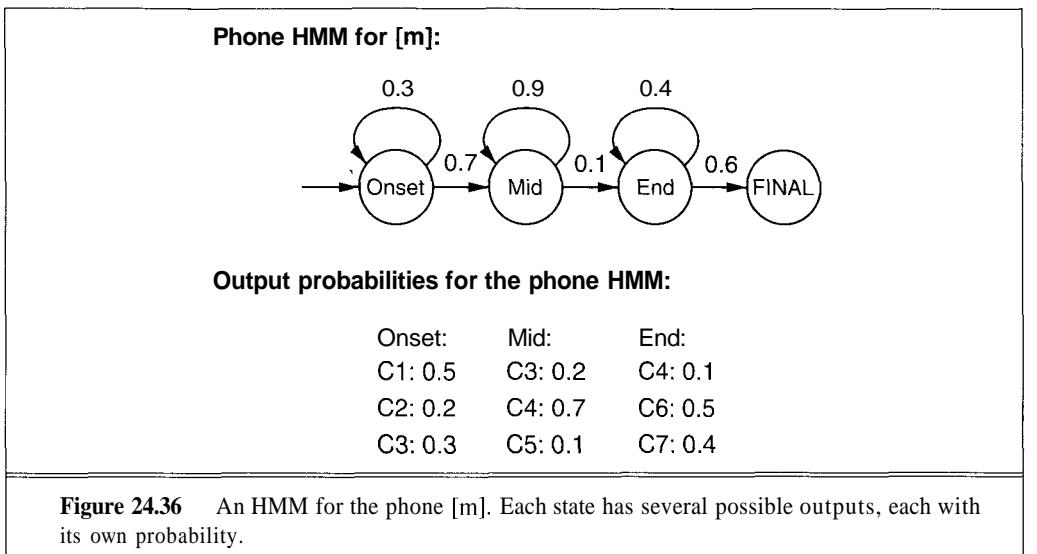


Figure 24.36 An HMM for the phone [m]. Each state has several possible outputs, each with its own probability.

We could repeat the calculation for all the other phone models to see which one is the most probable source of the speech signal.

Actually, most phones have a duration of 50-100 milliseconds, or 5-10 frames at 10 msec/frame. So the [C1,C4,C6] sequence is unusually quick. Suppose we have a more typical speaker who generates the sequence [C1,C1,C4,C4,C6,C6] while producing the phone. It turns out there are two paths through the model that generate this sequence. In one of them both C4s come from the Mid state (note the arcs that loop back), and in the other the second C4 comes from the End state. We calculate the probability that this sequence came from the [m] model in the same way: take the sum over all possible paths of the probability of the path times the probability that the path generates the sequence.

$$\begin{aligned}
 P([C1, C1, C4, C4, C6, C6] | [m]) = & \\
 (0.3 \times 0.7 \times 0.9 \times 0.1 \times 0.4 \times 0.6) \times (0.5 \times 0.5 \times 0.7 \times 0.7 \times 0.5 \times 0.5) + & \\
 (0.3 \times 0.7 \times 0.1 \times 0.4 \times 0.4 \times 0.6) \times (0.5 \times 0.5 \times 0.7 \times 0.1 \times 0.5 \times 0.5) & \\
 = 0.0001477
 \end{aligned}$$

We see that the loops in the phone model allow the model to represent both fast and slow speech, a very important source of variation. The multiple vector quantization values on each state represent other sources of variation. Altogether, this makes for a fairly powerful model. The hard part is getting good probability values for all the parameters. Fortunately, there are ways of acquiring these numbers from data, as we shall see.

Putting the models together

We have described three models. The language bigram model gives us $P(\text{word}_i | \text{word}_{i-1})$. The word pronunciation HMM gives us $P(\text{phones} | \text{word})$. The phone HMM gives us $P(\text{signal} | \text{phone})$. If we want to compute $P(\text{words} | \text{signal})$, we will need to combine these models in some way. One

approach is to combine them all into one big HMM. The bigram model can be thought of as an HMM in which every state corresponds to a word and every word has a transition arc to every other word. Now replace each word-state with the appropriate word model, yielding a bigger model in which each state corresponds to a phone. Finally, replace each phone-state with the appropriate phone model, yielding an even bigger model in which each state corresponds to a distribution of vector quantization values.

Some speech recognition systems complicate the picture by dealing with coarticulation effects at either the word/word or phone/phone level. For example, we could use one phone model for [ow] when it follows a [t] and a different model for [ow] when it follows a [g]. There are many trade-offs to be made—a more complex model can handle subtle effects, but it will be harder to train. Regardless of the details, we end up with one big HMM that can be used to compute $P(\text{words}|\text{signal})$.

The search algorithm

From a theoretical point of view, we have just what we asked for: a model that computes $P(\text{words}|\text{signal})$. All we have to do is enumerate all the possible word strings, and we can assign a probability to each one. Practically, of course, this is infeasible, because there are too many candidate word strings. Fortunately, there is a better way.

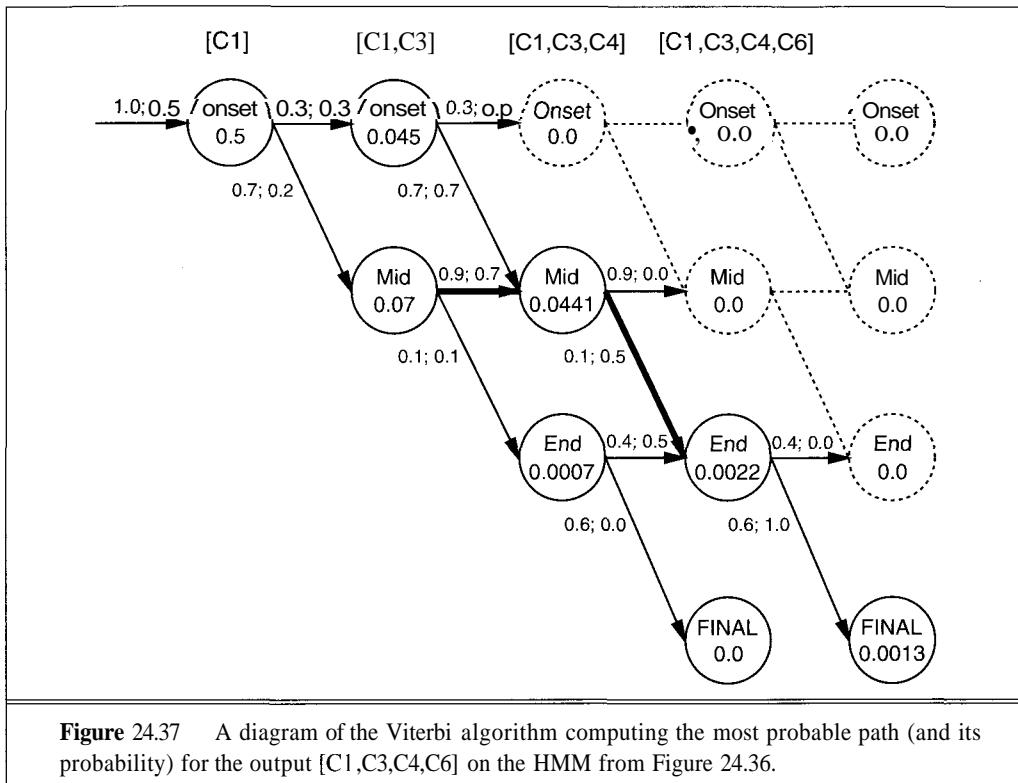
VITERBI ALGORITHM

The **Viterbi algorithm** takes an HMM model and an output sequence, $[C_1, C_2, \dots, C_n]$, and returns the most probable path through the HMM that outputs the sequence. It also returns the probability for the path. Think of it as an iterative algorithm that first finds *all* the paths that output the first symbol, C_1 . Then, for each of those paths it finds the most probable path that outputs the rest of the sequence, given that we have chosen a particular path for C_1 . So far this doesn't sound very promising. If the length of the sequence is n and there are M different states in the model, then this algorithm would seem to be at least $O(M^n)$.

The key point of the Viterbi algorithm is to use the Markov property to make it more efficient. The Markov property says that the most probable path for the rest of any sequence can depend only on the state in which it starts, not on anything else about the path that got there. That means we need not look at all possible paths that lead to a certain state; for each state, we only need to keep track of the *most probable* path that ends in that state. Thus, the Viterbi algorithm is an instance of **dynamic programming**.

Figure 24.37 shows the algorithm working on the HMM from Figure 24.36 and the output sequence $[C_1, C_3, C_4, C_6]$. Each column represents one iteration of the algorithm. In the leftmost column, we see that there is only one way to generate the sequence $[C_1]$, with the path [Onset]. The oval labelled "Onset 0.5" means that the path ends in the Onset state and has probability 0.5. The arc leading into the oval has the label "1.0; 0.5," which means that the probability of making this transition is 1.0, and the probability of outputting a C_1 , given that the transition is made, is 0.5. In the second column, we consider all the possible continuations of the paths in the first column that could lead to the output $[C_1, C_3]$. There are two such paths, one ending in the Onset state and one in the Mid state. In the third column, it gets more interesting. There are two paths that lead to the Mid state, one from Onset and the other from Mid. The bold arrow indicates that the path from Mid is more probable (it has probability 0.0441), so that is the only

one we have to remember. The path [Onset,Onset,Mid] has a lower probability, 0.022, so it is discarded. We continue in this fashion until we reach the FINAL state, with probability 0.0013. By tracing backwards and following the bold arrows whenever there is a choice, we see that the most probable path is [Onset,Onset,Mid,End,FINAL]. The Viterbi algorithm is $O(bMn)$, where b is the branching factor (the number of arcs out of any state). If the model is fully connected, then $b = M$, and the algorithm is $O(M^2n)$, which is still quite an improvement over $O(M^n)$.



Training the model

The HMM approach is used in speech recognition for two reasons. First, it is a reasonably good performance element—we saw that the Viterbi algorithm is linear in the length of the input. More importantly, HMMs can be learned directly from a training set of [signal,words] pairs. This is important because it is far too difficult to determine all the parameters by hand. There are other approaches that make better performance elements than HMMs, but they require the training data to be labelled on a phone-by-phone basis rather than a sentence-by-sentence basis, and that too is a difficult task. The standard algorithm for training an HMM is called the Baum–Welch or forward-backward algorithm. Rabiner (1990) gives a tutorial on this and other HMM algorithms.

The best current speech recognition systems recognize from about 80% to 98% of the words correctly, depending on the quality of the signal, the allowable language, the length of each input, and the variation in speakers. Speech recognition is easy when there is a good microphone, a small vocabulary, a strong language model that predicts what words can come next, a limit of one-word utterances (or a requirement for pauses between words), and when the system can be trained specifically for a single speaker. The systems are not as accurate over phone lines, when there is noise in the room, when there is a large vocabulary with no restrictions, when the words of an utterance run together, and when the speaker is new to the system.

24.8 SUMMARY

Although perception appears to be an effortless activity for humans, it requires a significant amount of sophisticated computation. This chapter studies vision as the prime example of perceptual information processing. The goal of vision is to extract information needed for tasks such as manipulation, navigation, and object recognition. We also looked at speech recognition.

- The process of **image formation** is well-understood in its geometric and physical aspects. Given a description of a 3-D scene, we can easily produce a picture of it from some arbitrary camera position (the graphics problem). Inverting the process by going from an image to a description of the scene is difficult.
- To extract the visual information necessary for the tasks of manipulation, navigation, and recognition, intermediate representations have to be constructed. **Image-processing** algorithms extract primitive elements from the image, such as edges and regions.
- In the image, there exist multiple cues that enable one to obtain 3-D information about the scene. These include motion, stereopsis, texture, shading, and contour analysis. Each of these cues relies on background assumptions about physical scenes in order to provide unambiguous interpretations.
- Object recognition in its full generality is a very hard problem. We discussed two relatively simple techniques—alignment and indexing using geometric invariants—that provide robust recognition in restricted contexts.
- Speech recognition is a problem in diagnosis. It can be solved with a language model and an acoustic model. Current emphasis is on systems that do well both as a performance element and a learning element.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

Systematic attempts to understand human vision can be traced back to ancient times. Euclid (ca. 300 B.C.) wrote about natural perspective, the mapping that associates with each point P in the three-dimensional world the direction of the ray OP joining the center of projection O to the

point P . He was well aware of the notion of motion parallax. The mathematical understanding of perspective projection, this time in the context of projection onto planar surfaces, had its next significant advance in the fifteenth century in Renaissance Italy. Brunelleschi (1413) is usually credited with creating the first paintings based on geometrically correct projection of the three-dimensional scene. In 1435, Alberti codified the rules and inspired generations of artists whose artistic achievements amaze us to this day (Kemp, 1990). Particularly notable in their development of the science of perspective, as it was called in those days, were Leonardo Da Vinci and Albrecht Dürer. Leonardo's late fifteenth century descriptions of the interplay of light and shade (chiaroscuro), umbra and penumbra regions of shadows, and aerial perspective are still worth reading in translation (Kemp, 1989).

Although perspective was known to the Greeks, they were curiously confused by the role of the eyes in vision. Aristotle thought of the eyes as devices emitting rays, rather in the manner of modern laser range finders. This mistaken view was laid to rest by the work of Arab scientists, such as Alhazen in the tenth century. The development of various kinds of cameras followed. These consisted of rooms (*camera* is Latin for chamber) where light would be let in through a small hole in one wall to cast an image of the scene outside on the opposite wall. Of course, in all these cameras, the image was inverted, which caused no end of confusion. If the eye was to be thought of as such an imaging device, how do we see right side up? This exercised the greatest brains of the era (including Leonardo). It took the work of Kepler and Descartes to settle the question. Descartes placed an eye from which the opaque cuticle had been removed in a hole in a window shutter. This resulted in an inverted image being formed on a piece of paper laid out on the retina. While the retinal image is indeed inverted, this need not cause a problem if the brain interprets the image the right way. In modern jargon, one just has to access the data structure appropriately.

The next major advances in the understanding of vision took place in the nineteenth century. The work of Helmholtz and Wundt, described in Chapter 1, established psychophysical experimentation as a rigorous scientific discipline. Through the work of Young, Maxwell, and Helmholtz, a trichromatic theory of color vision was established. That humans can see depth if the images presented to the left and right eyes are slightly different was demonstrated by Wheatstone's (1838) invention of the stereoscope. The device immediately became very popular in parlors and salons throughout Europe. The essential concept of binocular stereopsis, that two images of a scene taken from slightly different viewpoints carry information sufficient to obtain a 3-D reconstruction of the scene, was exploited in the field of photogrammetry. Key mathematical results were obtained—Kruppa (1913) proved that given two views of five distinct points, one could reconstruct the rotation and translation between the two camera positions as well as the depth of the scene (up to a scale factor). Although the geometry of stereopsis had been understood for a long time, the correspondence problem in photogrammetry used to be solved by humans trying to match up corresponding points. The amazing ability of humans in solving the correspondence problem was illustrated by Julesz's invention of the random dot stereogram (Julesz, 1971). Both in computer vision and in photogrammetry, much effort was devoted to solving this problem in the 1970s and 1980s.

The second half of the nineteenth century was a major foundational period for the psychophysical study of human vision. In the first half of the twentieth century the most significant research results in vision were obtained by the Gestalt school of psychology led by Max

Wertheimer. With the slogan "The whole is greater than the sum of the parts," they laid primary emphasis on grouping processes, both of contours and regions. Constructing computational models of these processes remains a difficult problem to this day.

The period after World War 2 was marked by renewed activity. Most significant was the work of J. J. Gibson (1950; 1979), who pointed out the importance of optical flow as well as texture gradients in the estimation of environmental variables such as surface slant and tilt. He reemphasized the importance of the stimulus and how rich it was. Gibson, Olum, and Rosenblatt (1955) pointed out that the optical flow field contained enough information to determine the egomotion of the observer relative to the environment. In the computational vision community, work in this area and the (mathematically equivalent) area of structure from motion developed mainly in the 1980s, following the seminal works of Koenderink and van Doorn (1975), Ullman (1979), and Longuet-Higgins (1981). Faugeras (1993) presents a comprehensive account of our understanding in this area. In the 1990s, with the increase in computer speed and storage, the importance of motion sequence analysis from digital video is growing rapidly.

In computational vision, major early works in shape from texture are due to Bajcsy and Liebermann (1976) and Stevens (1981). Whereas this work was for planar surfaces, a comprehensive analysis for curved surfaces is due to Gardsing (1992) and Malik and Rosenholtz (1994).

In the computational vision community, shape from shading was first studied by Berthold Horn (1970). Horn and Brooks (1989) present an extensive survey of the main papers in the area. This framework made a number of simplifying assumptions, the most critical of which was ignoring the effect of mutual illumination. The importance of mutual illumination has been well-appreciated in the computer graphics community, where ray tracing and radiosity have been developed precisely to take mutual illumination into account. A theoretical and empirical critique may be found in Forsyth and Zisserman (1991).

In the area of shape from contour, after the key initial contributions of Huffman (1971) and Clowes (1971), Mackworth (1973) and Sugihara (1984) completed the analysis for polyhedral objects. Malik (1987) developed a labeling scheme for piecewise smooth curved objects. Understanding the visual events in the projection of smooth curved objects requires an interplay of differential geometry and singularity theory. The best study is Koenderink's (1990) *Solid Shape*.

In the area of three-dimensional object recognition, the seminal work was Roberts's (1963) thesis at MIT. It is often considered to be the first PhD thesis in computer vision and introduced several key ideas including edge detection and model-based matching. The idea of alignment, first introduced by Roberts, resurfaced in the 1980s in the work of Lowe (1987) and Huttenlocher and Ullman (1990). Generalized cylinders were introduced by Binford in 1971, and were used extensively by Brooks in the ACRONYM system (Brooks, 1981). Geometrical invariants were studied extensively in the late nineteenth century by English and German mathematicians. Their use in object recognition is surveyed by Mundy and Zisserman (1992). Excellent results have been achieved even in cluttered scenes (Rothwell *et al.*, 1993).

A word about the research methodology used in computer vision. The early development of the subject, like that of rest of AI, was mostly through Ph.D. theses that consisted largely of descriptions of implemented systems. The work lacked significant contact with the literature on human vision and photogrammetry, in which many of the same problems had been studied. David Marr played a major role in connecting computer vision to the traditional areas of biological vision—psychophysics and neurobiology. His main work, *Vision* (Marr, 1982), was published

posthumously. That book conveys the excitement of working in vision better than any written since, despite the fact that many of the specific hypotheses and models proposed by Marr have not stood the test of time.

Under Marr's influence, reconstruction of the three-dimensional scene from various cues became the dominant ideology of the day. This of course proved to be a difficult problem, and it was inevitable that people would question whether it was really necessary. The old ideas of Gibson—active vision and affordances—came back. The most solid proof that reconstruction was not necessary for many (or most) tasks came from the work of Dickmanns in Germany, who demonstrated robust driving using a control system perspective (Dickmanns and Zapp, 1987). As a general philosophy, active vision was advocated by Ruzena Bajcsy (1988) and John Aloimonos (1988). A number of papers are collected in a special issue of *CVGIP* (Aloimonos, 1992). In the 1990s, the dominant perspective is that of vision as a set of processes aimed at extracting information for manipulation, navigation, and recognition.

Eye, Brain and Vision by David Hubel (1988) and *Perception* by Irvin Rock (1984) provide excellent introductions to the field of biological vision. A *Guided Tour of Computer Vision* (Nalwa, 1993) is a good general introduction to computer vision; *Robot Vision* (Horn, 1986) and *Three-Dimensional Computer Vision* (Faugeras, 1993) cover more advanced topics. Two of the main journals for computer vision are the *IEEE Transactions on Pattern Analysis and Machine Intelligence* and the *International Journal of Computer Vision*. Computer vision conferences include ICCV (International Conference on Computer Vision), CVPR (Computer Vision and Pattern Recognition), and ECCV (European Conference on Computer Vision).

The hidden Markov model was first used to model language by Markov himself in a letter-sequence analysis of the text of *Eugene Onegin* (Markov, 1913). Early development of algorithms for inferring Markov models from data was carried by Baum and Petrie (1966). These were applied to speech by Baker (1975) and Jelinek (1976). In 1971, the Defense Advanced Research Projects Agency (DARPA) of the United States Department of Defense, in cooperation with a number of research centers, set out a five-year plan for speech recognition research. The two most significant systems to emerge from this massive effort were HEARSAY-II (Erman *et al.*, 1980) and HARPY (Lowerre and Reddy, 1980). HARPY was the only system that clearly met the rigorous specifications of the five-year plan. It used a highly compiled network representation for all possible meaningful sequences of speech elements. HEARSAY-II, however, has had a greater influence on other research because of its use of the **blackboard architecture**. HEARSAY-II was designed as an expert system with a number of more or less independent, modular **knowledge sources** which communicated via a common **blackboard** from which they could write and read. Because this representation was less compiled and more modular than HARPY'S, HEARSAY-II was much easier to comprehend and modify, but was not fast enough to meet the DARPA criteria.

A good introduction to speech recognition is given by Rabiner and Juang (1993). Waibel and Lee (1990) collect important papers in the area, including some tutorial ones. Lee (1989) describes a complete modern speech recognition system. The presentation in this chapter drew on the survey by Kay, Gawron, and Norvig (1994), and on an unpublished manuscript by Dan Jurafsky. Speech recognition research is published in *Computer Speech and Language* and the *IEEE Transactions on Acoustics, Speech, and Signal Processing*, and at the DARPA Workshops on Speech and Natural Language Processing.

EXERCISES

24.1 In the shadow of a tree with a dense, leafy canopy, one sees a number of light spots. Surprisingly, they all appear to be circular. Why? After all, the gaps between the leaves through which the sun shines through are not likely to be circular.

24.2 Label the line drawing in Figure 24.38, assuming that the outside edges have been labelled as occluding and that all vertices are trihedral. Do this by a backtracking algorithm that examines the vertices in the order A, B, C, and D, picking at each stage a choice consistent with previously labelled junctions and edges. Now try the vertices in the order B, D, A, and C.

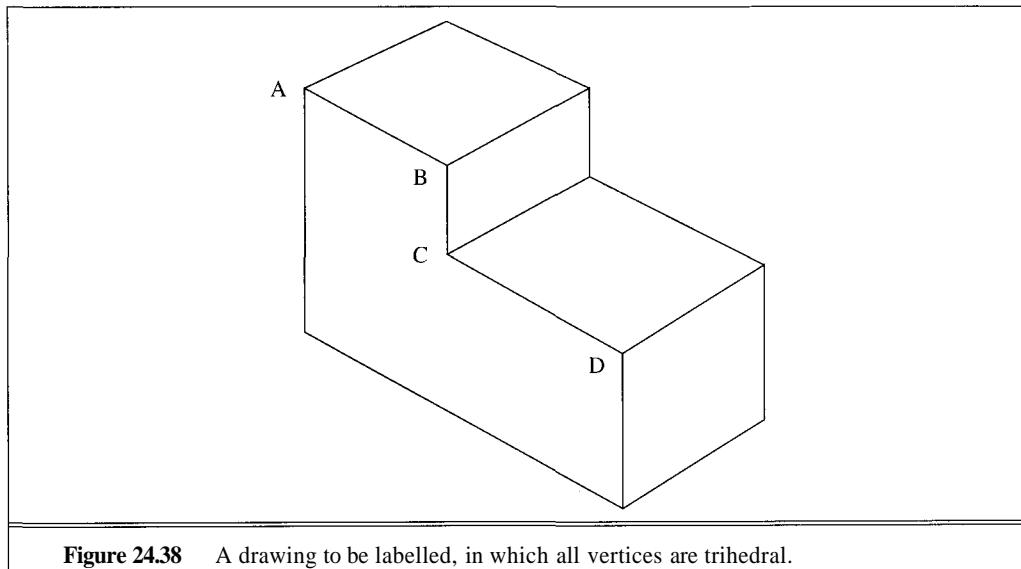


Figure 24.38 A drawing to be labelled, in which all vertices are trihedral.

24.3 Consider an infinitely long cylinder of radius r oriented with its axis along the y -axis. The cylinder has a Lambertian surface and is viewed by a camera along the positive z -axis. What will you expect to see in the image of the cylinder if the cylinder is illuminated by a point source at infinity located on the positive x -axis. Explain your answer by drawing the isobrightness contours in the projected image. Are the contours of equal brightness uniformly spaced?

24.4 Edges in an image can correspond to a variety of scene events. Consider the photograph on the cover of your book and assume that it is a picture of a real 3-D scene. Identify a set of ten different brightness edges in the image, and for each, decide whether it corresponds to a discontinuity in (a) depth, (b) surface normal, (c) reflectance, or (d) illumination.

24.5 Show that convolution with a given function f commutes with differentiation, that is,

$$(f * g)' = f * g'$$

24.6 A stereo system is being contemplated for terrain mapping. It will consist of two CCD cameras, each having 512×512 pixels on a 10cm x 10cm square sensor. The lenses to be used have focal length 16cm, and focus is fixed at infinity. For corresponding points (u_1, v_1) in the left image and (u_2, v_2) in the right image, $v_1 = V_2$ as the x-axes in the two image planes are parallel to the epipolar lines. The optical axes of the two cameras are parallel. The baseline between the cameras is 1 meter.

- a. If the nearest range to be measured is 16 meters, what is the largest disparity that will occur (in pixels)?
- b. What is the range resolution, due to the pixel spacing, at 16 meters?
- c. What range corresponds to a disparity of one pixel?

24.7 In Figure 24.30, physically measure the cross ratio of the points $ABCD$ as well as of the points $A'B'C'D'$. Are they equal?

24.8 We wish to use the alignment algorithm in an industrial situation where flat parts are moving along a conveyor belt and being photographed by a camera vertically above the conveyor belt. The pose of the part is specified by three variables, one for the rotation and two for the 2-D position. This simplifies the problem and the function FIND-TRANSFORM needs only two pairs of corresponding image and model features to determine the pose. Determine the worst-case complexity of the alignment procedure in this context.

24.9 Read this chapter from the beginning until you find ten examples of homophones. Does the status of a word as a homophone depend on the accent of the speaker?

24.10 Calculate the most probable path through the HMM in Figure 24.36 for the output sequence [C1,C2,C3,C4,C4,C6,C7]. Also give its probability.

24.11 Some sports announcers have been known to celebrate a score with the drawn out pronunciation [g ow ow ow ow ow el]. Draw a word HMM for "goal" such that the most probable path has a sequence of four [ow]s, but any number greater than 1 is possible.

24.12 The Viterbi algorithm finds the most probable sequence of phones corresponding to the speech signal. Under the assumption that some words can be pronounced with more than one sequence of phones, explain why the Viterbi algorithm only computes an approximation to $P(\text{words}|\text{signal})$.

25 ROBOTICS

In which agents are endowed with physical effectors with which to do mischief.

25.1 INTRODUCTION

ROBOT

The Robot Institute of America defines a **robot** as *a programmable, multifunction manipulator designed to move material, parts, tools, or specific devices through variable programmed motions for the performance of a variety of tasks*. This definition is not very demanding; a conveyer belt with a two-speed switch would arguably satisfy it.

AUTONOMOUS
ROBOTS

We will define **robot** simply as *an active, artificial agent whose environment is the physical world*. The *active* part rules out rocks, the *artificial* part rules out animals, and the *physical* part rules out pure software agents or **softbots**, whose environment consists of computer file systems, databases and networks. We will be concerned primarily with **autonomous robots**, those that make decisions on their own, guided by the feedback they get from their physical sensors.

Most of the design of an autonomous robot is the same as the design of any autonomous agent. To some extent, we could take a generic planning agent (Chapter 11) or decision-making agent (Chapter 16), equip it with wheels, grippers, and a camera, point it out the door, and wish it good luck. Unfortunately, unless we pointed it at an exceptionally benign environment, it would not fare very well. The real world, in general, is very demanding. We can see this by considering the five properties of environments from page 46:

- The real world is **inaccessible**. Sensors are imperfect, and in any case can only perceive stimuli that are near the agent.
- The real world is **nondeterministic**, at least from the robot's point of view. Wheels slip, batteries run down, and parts break, so you never know if an action is going to work. That means a robot needs to deal with uncertainty (Part V).
- The real world is **nonepisodic**—the effects of an action change over time. So a robot has to handle sequential decision problems (Chapter 17) and learning (Part VI).

- The real world is **dynamic**. Therefore, a robot has to know when it is worth deliberating and when it is better to act immediately.
- The real world is **continuous**, in that states and actions are drawn from a continuum of physical configurations and motions. Because this makes it impossible to enumerate the set of possible actions, many of the search and planning algorithms described in earlier chapters will need to be modified.

We could go on about the unique properties of robotics, but instead we will refer you to Exercise 25.1. It does not require any additional reading, it will give you an instant and visceral appreciation of the problems of robotics (and their possible solutions), and it can be amusing.

This chapter has three main points. First, we look at the tasks that robots perform (Section 25.2) and the special effectors and sensors they use (Section 25.3). Second, we step away from *robots per se* and look at agent architectures for inaccessible, nondeterministic, nonepisodic, dynamic domains (Section 25.4). Third, we look at the specific problem of selecting actions in a continuous state space (Sections 25.5 and 25.6). The algorithms rely on a level of computational geometry that is more suited for a specialized advanced text, so we only give qualitative descriptions of the problems and solutions.

25.2 TASKS: WHAT ARE ROBOTS GOOD FOR?

While humans do a wide variety of things using more or less the same body, robot designs vary widely depending on the task for which they are intended. In this section, we survey some of the tasks, and in the next section, we look at the available parts (effectors and sensors) that make up a robot's body.

Manufacturing and materials handling

Manufacturing is seen as the traditional domain of the robot. The repetitive tasks on a production line are natural targets for automation, and so in 1954 George Devol patented the programmable robot arm, based on the same technology—punchcards—that was used in the Jacquard loom 150 years earlier. By 1985, there were about 180,000 robots in production lines around the world, with Japan, the United States, and France accounting for 150,000 of them. The automotive and microelectronics industries are major consumers of robots. However, most robots in use today are very limited in their abilities to sense and adapt. *Autonomous* robots are still struggling for acceptance. The reasons for this are historical, cultural, and technological. Manufacturing existed and burgeoned long before robots appeared, and many tricks were developed for solving manufacturing problems without intelligence. And it is still true that simple machines are the best solution for simple tasks.

Material handling is another traditional domain for robots. Material handling is the storage, transport, and delivery of material, which can range in size from silicon chips to diesel trucks. The robots used for material handling likewise vary from table-top manipulators to gantry cranes, and

include many types of AGV (autonomous guided vehicles, typically golf-cart sized four-wheeled vehicles that are used to ferry containers around a warehouse). Handling odd-shaped parts is simplified by placing each part in a cradle or *pallet* that has a base of fixed shape. Bar codes on the pallets make it easy to identify and inventory the parts. But these techniques falter when applied to food packaging and handling, an area likely to see rapid growth in the future. The large variety of forms, weights, textures, and firmnesses in familiar foods make this task a challenge for future research.

Robots have recently been making their mark in the construction industry. Large prototype robots have been built that can move a one-ton object with an accuracy of 2.5 mm in a workspace of radius 10 m. Sheep shearing is another impressive application. Australia boasts a population of 140 million sheep, and several industrial and academic development groups have deployed automated sheep shearers. Because sheep come in different sizes, some kind of sensing (vision or mechanical probing) is needed to get an idea of the right overall shearing pattern. Tactile feedback is used to follow the contours of the animal without injuring it.

Gofer robots

MOBOTS

Mobile robots (**mobots**) are also becoming widely useful. Two primary applications are as couriers in buildings, especially hospitals, and as security guards. One company has sold over 3000 “mailmobiles.” These robots respond to requests from a computer terminal and carry documents or supplies to a destination somewhere else in the building. The robot negotiates hallways and elevators, and avoids collision with other obstacles such as humans and furniture. The robot is appropriate for this task because it has high availability (it can work 24 hours a day), high reliability so that supplies are not lost or misplaced, and its progress can be monitored to allow preparation for its arrival, or detection of failure.

We have seen (page 586) that autonomous vehicles are already cruising our highways, and AUVs (autonomous underwater vehicles) are cruising the seas. It is far less expensive to send a robot to the bottom of the ocean than a manned submarine. Furthermore, a robot can stay down for months at a time, making scientific observations, reporting on enemy submarine traffic, or fixing problems in transoceanic cables.

Hazardous environments

Mobile robots are an important technology for reducing risk to human life in hazardous environments. During the cleanup of the Chernobyl disaster, several Lunokhod lunar explorer robots were converted to remote-controlled cleaning vehicles. In Japan and France, robots are used for routine maintenance of nuclear plants. Crisis management is now being taken seriously as a technological challenge in the United States. Robots can reduce risk to humans in unstable buildings after earthquakes, near fire or toxic fumes, and in radioactive environments. They can also be used for routine tasks in dangerous situations including toxic waste cleanup, deep sea recovery, exploration, mining, and manipulation of biologically hazardous materials. A human operator is often available to guide the robot, but it also needs autonomy to recognize and respond to hazards to its own and others' well-being.

Autonomy is also essential in remote environments (such as the surface of Mars) where communication delays are too long to allow human guidance. A robot for hazardous environments should be able to negotiate very rough ground without falling or tipping over.¹ It should have sensing modalities that work with no light present. It should above all avoid doing harm to humans who may be present but unconscious or disabled in its environment. Achieving this level of autonomy along with reliability is a challenge for robotics. But the return is very great, and this area of research is bound to continue its growth.

Telepresence and virtual reality

As computers spread from scientific/business applications to a consumer technology, telepresence and virtual reality continue to captivate the public imagination. The idea of staying in one's home and being able to sense exotic environments, either real (telepresence) or imaginary (virtual reality) is indeed compelling, and is a driving force behind some major moves in the computer and entertainment industries. Among the possible environments to which one might connect, there are many important applications. Today, New York City police use teleoperated robots to answer many of the city's estimated 9000 yearly bomb scares. Architects and customers can walk through or fly over a building before it is built.² Oceanographers can collect samples with a deep-sea submersible off the Pacific coast without leaving their offices. Medical specialists can examine patients hundreds of miles away. Surgeons can practice new techniques on virtual organ models. Or they can operate with miniature implements through a catheter inserted in a tiny incision.³

Much of the research in telepresence and virtual reality fall in the robotics category. Robotics researchers have for many years been studying, designing, and improving anthropomorphic hands. They have also built exoskeletons and gloves that provide both control and sensory feedback for those hands. Tactile sensing research opens the possibility of a true remote sense of touch. Realistic virtual environments require object models with a full set of physical attributes. Simulation requires algorithms that accurately factor in those attributes, including inertia, friction, elasticity, plasticity, color, texture, and sound. There remains much work to do to fully exploit the possibilities of remote and virtual presence.

Augmentation of human abilities

A precursor to telepresence involves putting a human inside a large, strong robot. In 1969, General Electric built the Quadrupedal Walking Machine, an 11-foot, 3000-pound robot with a control harness in which a human operator was strapped. Project leader Ralph Mosher remarked that the man-machine relationship "is so close that the experienced operator begins to feel as if those mechanical legs are his own. You imagine that you are actually crawling along the ground

¹ The robot DANTE-II caused embarrassment to its sponsors when it tipped over while exploring a volcano in July, 1994.

² The cover of this book shows a tiny model of Soda Hall, the computer science building at Berkeley. The model was used for walkthroughs prior to actual construction.

³ The first medical telepresence system was deployed in 1971, under NASA sponsorship. A mobile van parked at the Papago Indian Reservation near Tucson, Arizona, allowed patients to be diagnosed remotely.

on all fours—but with incredible strength." A futuristic full-body exoskeleton of this kind was worn by Ripley (Sigourney Weaver) in the final confrontation of the movie "Aliens."

No less fascinating is the attempt to duplicate lost human effectors. When an arm or leg is amputated, the muscles in the stump still respond to signals from the brain, generating myoelectric currents. Prosthetic limbs can pick up these signals and amplify them to flex joints and move artificial fingers. Some prosthetics even have electrocutaneous feedback that gives a sense of touch. There has also been work in giving humans artificial sensors. A Japanese MITI project, for example, built a prototype robot guide dog for the blind. It uses ultrasound to make sure that its master stays in a safe area as they walk along together. Artificial retinas and cochleas, mostly based on analog VLSI, are the subject of intensive research at present.

25.3 PARTS: WHAT ARE ROBOTS MADE OF?

LINKS
JOINTS

END EFFECTORS

EFFECTOR
ACTUATOR

DEGREE OF
FREEDOM

LOCOMOTION
MANIPULATION

Robots are distinguished from each other by the effectors and sensors with which they are equipped. For example, a mobile robot requires some kind of legs or wheels, and a teleoperated robot needs a camera. We will assume that a robot has some sort of rigid body, with rigid **links** that can move about. Links meet each other at **joints**, which allow motion. For example, on a human the upper arm and forearm are links, and the shoulder and elbow are joints. The palm is a link, and fingers and thumb have three links. Wrists and finger joints are joints. Robots need not be so anthropomorphic; a wheel is a perfectly good link. Attached to the final links of the robot are **end effectors**, which the robot uses to interact with the world. End effectors may be suction cups, squeeze grippers, screwdrivers, welding guns, or paint sprayers, to name a few. Some robots have special connectors on the last link that allow them to quickly remove one end effector and attach another. The well-equipped robot also has one or more sensors, perhaps including cameras, infrared sensors, radar, sonar, and accelerometers.

Effectors: Tools for action

An **effector** is any device that affects the environment, under the control of the robot. To have an impact on the physical world, an effector must be equipped with an **actuator** that converts software commands into physical motion. The actuators themselves are typically electric motors or hydraulic or pneumatic cylinders. For simplicity, we will assume that each actuator determines a single motion or **degree of freedom**. For example, an automatic phonograph turntable has three degrees of freedom. It can spin the turntable, it can raise and lower the stylus arm, and it can move the arm laterally to the first track. A side effect of the motion of the turntable is that one or more vinyl recordings rotate as well. This motion, assuming the stylus has been lowered into a recording groove, leads to a useful and musical product.

Effectors are used in two main ways: to change the position of the robot within its environment (**locomotion**), and to move other objects in the environment (**manipulation**). A third use, to change the shape or other physical properties of objects, is more in the realm of mechanical engineering than robotics, so we will not cover it.

Locomotion

STATICALLY STABLE

DYNAMICALLY STABLE

NONHOLONOMIC

HOLONOMIC

The vast majority of land animals use legs for locomotion. Legged locomotion turns out to be very difficult for robots, and is used only in special circumstances. The most obvious application is motion in rough terrain with large obstacles. The Ambler robot (Simmons *et al.*, 1992), for example, is a six-legged robot, about 30 feet tall, capable of negotiating obstacles more than 6 feet in diameter. The Ambler, unlike most animals, is a **statically stable** walker. That is, it can pause at any stage during its gait without tumbling over. Static stability walking is very slow and energy-inefficient, and the quest for faster, more efficient legged machines has led to a series of **dynamically stable** hopping robots (Raibert, 1986), which would crash if forced to pause, but do well as long as they keep moving. These robots use rhythmic motion of four, two, or even a single leg to control the locomotion of the body in three dimensions. They do not have enough legs in contact with the ground to be stable statically, and will fall if their hopping motion stops. They are dynamically stable because corrections to leg motion keep the body upright when it is bumped or when the ground is uneven. The control of legged machines is too complex a subject to discuss here, except to remark on its difficulty and to marvel at recent successes—for example, the one-legged robot shown in Figure 25.1 can "run" in any direction, hop over small obstacles, and even somersault.

Despite the anthropomorphic attractions of legged locomotion, wheel or tread locomotion is still the most practical for most environments. Wheels and treads are simpler to build, are more efficient than legs, and provide static support. They are also easier to control, although they are not without subtle problems of their own. Consider the car-like robot of Figure 25.2. We know from experience that without obstructions, we can drive a car to any position, and leave it pointing in any direction that we choose. Thus, the car has three degrees of freedom, two for its *x-y* position, and one for its direction. But there are only two actuators, namely, driving and steering. And for small motions, the car seems to have only two degrees of freedom, because we can move it in the direction it points, or rotate it slightly, but we cannot move it sideways.

It is important here to draw the distinction between what the actuators actually do, namely, turning or steering the wheels, and what these motions do to the environment. In this case, the side effect of the wheel motion is to move the car to any point in a three-dimensional space. Because the number of *controllable* degrees of freedom is only two, which is less than the total degrees of freedom (three), this is a **nonholonomic** robot. In general, a nonholonomic robot has fewer controllable degrees of freedom than total degrees of freedom. As a rule, the larger the gap between controllable and total degrees of freedom, the harder it is to control the robot. A car with a trailer has four total degrees of freedom but only two controllable ones, and takes considerable skill to drive in reverse. If the number of total and controllable degrees of freedom of the system is the same, the robot is **holonomic**.

It is possible to build truly holonomic mobile robots, but at the cost of high mechanical complexity. It is necessary to design the wheels or treads so that motion in the driving direction is controlled, but sideways motion is free. Holonomic drives usually replace the tire or tread with a series of rollers lined up with the drive direction. Although these designs make life easier for the control architect, the common designs of nonholonomic mobile robots (Figures 25.2 and 25.3) are not that difficult to control, and their mechanical simplicity makes them the best choice in most situations. The main distinction to be made between different designs is whether the robot

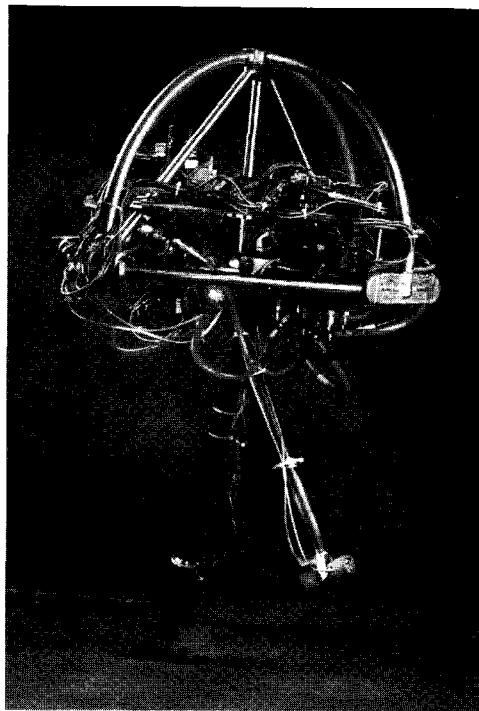


Figure 25.1 Raibert's dynamically stable hopping robot in motion. (© 1986. MIT Leg Laboratory. All rights reserved.)

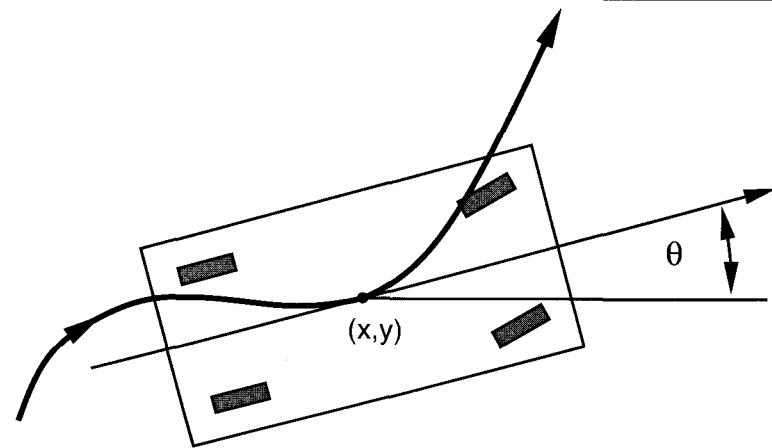


Figure 25.2 Motion of a four-wheeled vehicle with front-wheel steering.

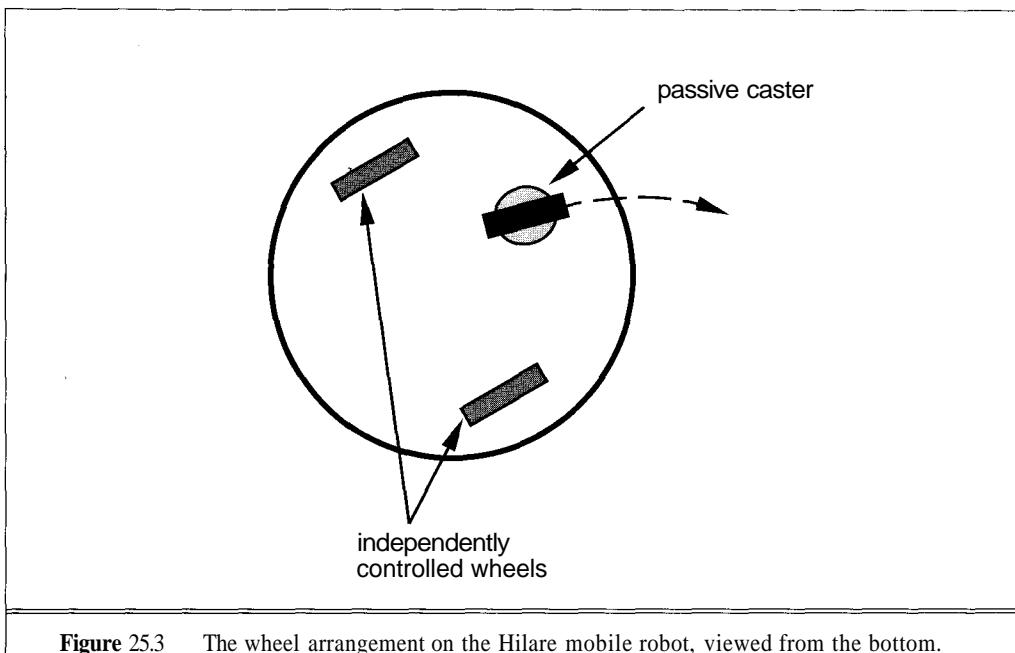


Figure 25.3 The wheel arrangement on the Hilare mobile robot, viewed from the bottom.

has a minimum turning radius or can turn on the spot. If the latter is true, then it is always possible to move from configuration *A* to *B* by turning on the spot toward *B*, moving to *B* in a straight line, and rotating on the spot to the orientation of *B*. If there is a turning-radius constraint, or if one would like to move without slowing down too much (which generates the same constraint), a special path planner is needed. It can be shown that the shortest path in such cases consists of straight-line segments joining segments of circles with the minimum radius.

Sometimes the robot's design makes the control of locomotion very difficult. Reversing a wheeled vehicle with several trailers is beyond the capability of most humans; hence some large fire engines are fitted with a second steering wheel for the back wheels to make it easier. Fire engines have only a single trailer, but two-trailer examples can be seen at any modern airport. When leaving a gate, most aircraft are driven by a nose-wheel tender. The nose-wheel tender (the car) drives a long link attached to the nose wheel (first trailer). The aircraft itself forms the second trailer. This combination must be backed out of the gate each time an aircraft departs, and the only control is the steering of the tender. Fortunately, the much greater length of the aircraft makes it insensitive to small motions of the tender and link, and the control problem is tractable for an experienced driver. Recent advances in control theory have resulted in algorithms for automatically steering vehicles with any number of trailers of any size.

Manipulation

We return now to **manipulators**, effectors that move objects in the environment. The ancestors of robot manipulators were teleoperated mechanisms that allowed humans to manipulate hazardous

KINEMATICS

ROTARY
PRISMATIC

materials, and that mimicked the geometry of a human arm. Early robots as a rule followed this precedent, and have anthropomorphic kinematics. Broadly defined, **kinematics** is the study of the correspondence between the actuator motions in a mechanism, and the resulting motion of its various parts.

Most manipulators allow for either **rotary** motion (rotation around a fixed hub) or **prismatic** motion (linear movement, as with a piston inside a cylinder). Figure 25.4 shows the Stanford Manipulator, used in several early experiments in robotics. A nearly anthropomorphic design is the Unimation PUMA shown in Figure 25.5. This design has six rotary joints arranged sequentially. The shorthand description of its kinematic configuration is “RRRRR,” (“6R” for short) listing joint types from base to tip. A free body in space has six degrees of freedom (three for x - y - z position, three for orientation), so six is the minimum number of joints a robot requires in order to be able to get the last link into an arbitrary position and orientation.

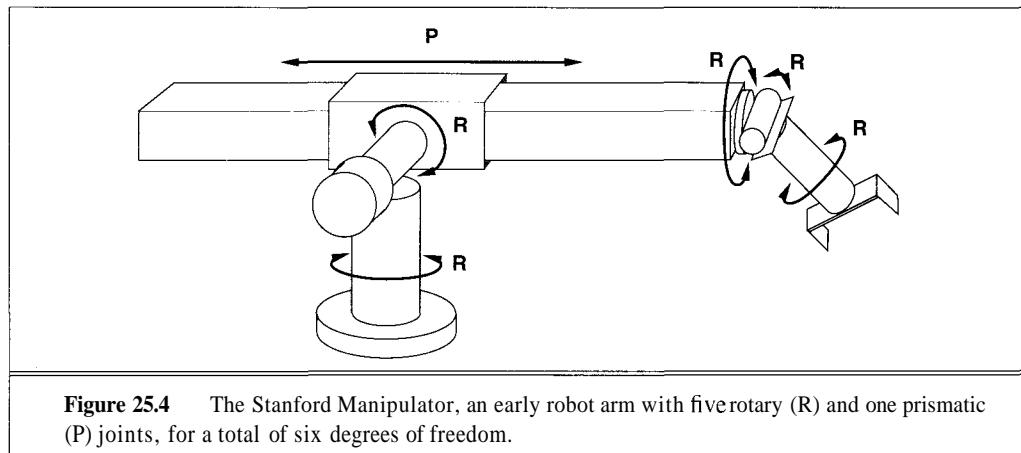


Figure 25.4 The Stanford Manipulator, an early robot arm with five rotary (R) and one prismatic (P) joints, for a total of six degrees of freedom.

In case this total of six degrees of freedom for a free body is not intuitively obvious, imagine the body is a tennis ball. The center of the ball can be described with three position coordinates. Now suppose that the ball is resting on a table with its center fixed. You can still rotate the ball without moving its center. Paint a dot anywhere on the surface of the ball, you can rotate the ball so that the dot touches the table-top. This takes two degrees of freedom, because the dot can be specified with latitude and longitude. With the center fixed and the dot touching the table-top, you can still rotate the ball about a vertical axis. This is the third and last degree of rotational freedom. (In an airplane or boat, the three types of rotation are called pitch, yaw, and roll.)

At the end of the manipulator is the robot's **end effector**, which interacts directly with objects in the world. It may be a screwdriver or other tool, a welding gun, paint sprayer, or a gripper. Grippers vary enormously in complexity. Two- and three-fingered grippers perform most tasks in manufacturing. The mechanical simplicity of these grippers makes them reliable and easy to control, both important attributes for manufacturing.

At the other end of the complexity spectrum are anthropomorphic hands. The Utah-MIT hand shown in Figure 25.6 faithfully replicates most of the kinematics of a human hand, less one finger. A human hand has a very large number of degrees of freedom (see Exercise 25.4).

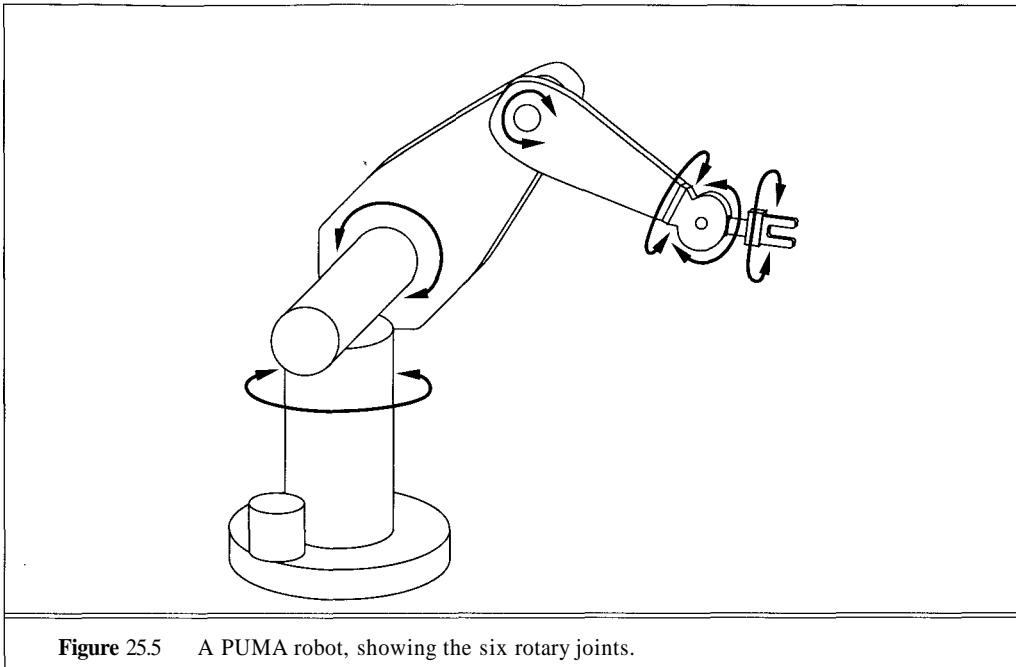


Figure 25.5 A PUMA robot, showing the six rotary joints.

Ultimately, the effectors are driven by electrical (or other) signals. Some effectors only accept on/off signals, some accept scalar values (e.g., turn right 3°), and some robot development environments provide higher level subroutine libraries or have complete languages for specifying actions that can be turned into primitive signals.

Sensors: Tools for perception

Chapter 24 covered the general principles of perception, using vision as the main example. In this section, we describe the other kinds of sensors that provide percepts for a robot.

Proprioception

Like humans, robots have a **proprioceptive**⁴ sense that tells them where their joints are. **Encoders** fitted to the joints provide very accurate data about joint angle or extension. If the output of the encoder is fed back to the control mechanism during motion, the robot can have much greater positioning accuracy than humans. For a manipulator, this typically translates to around a few mils (thousandths of an inch) of accuracy in its end-effector position. In contrast, humans can manage only a centimeter or two. To test this for yourself, place your finger at one end of a ruler (or some other object whose length is familiar). Then with your eyes closed try to touch the other

⁴ The word **proprioceptive** is derived from the same source as **proprietary**, and thus means "perception of privately owned (i.e., internal) stimuli."

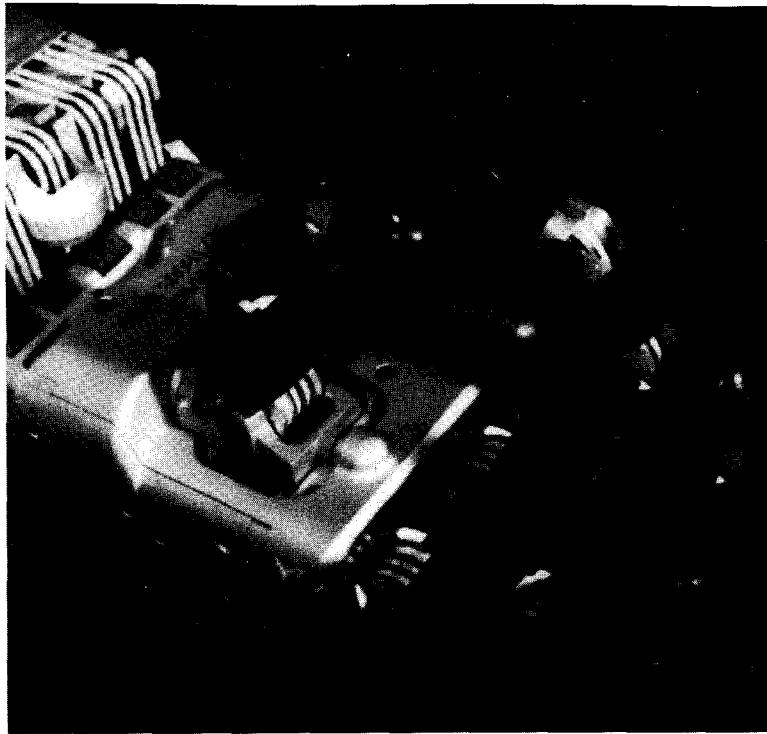


Figure 25.6 The Utah-MIT Hand.

REPEATABILITY

end. You should do better after a few tries, which shows that your positioning **repeatability** is better than your accuracy. The same is usually true for robots.

ODOMETRY

Mobots can measure their change in position using **odometry**, based on sensors that measure wheel rotation (or, in the case of stepper motors that rotate a fixed angle per step, the number of steps). Unfortunately, because of slippage as the robot moves, the position error from wheel motion deteriorates as the robot moves, and may be several percent of the distance travelled. Orientation can be measured more reliably, using a magnetic compass or a gyroscope system. Accelerometers can measure the change in velocity.

Force sensing

Even though robots can sense and control the positions of their joints much more accurately than humans, there are still many tasks that cannot be carried out using only position sensing. Consider, for example, the task of scraping paint off a windowpane using a razor blade. To get all the paint requires positioning accuracy of about a micron in the direction perpendicular to the glass. An error of a millimeter would cause the robot to either miss the paint altogether or break the glass. Obviously, humans are not doing this using position control alone. This and

FORCE SENSOR

COMPLIANT MOTIONS

TACTILE SENSING

many other tasks involving contact, such as writing, opening doors, and assembling automobiles, require accurate control *of forces*. Force can be regulated to some extent by controlling electric motor current, but accurate control requires a **force sensor**. These sensors are usually placed between the manipulator and end effector and can sense forces and torques in six directions. Using force control, a robot can move along a surface while maintaining contact with a fixed pressure. Such motions are called **compliant motions**, and are extremely important in many robotic applications.

Tactile sensing

Picking up a paper coffee cup or manipulating a tiny screw requires more than proprioception. The force applied to the cup must be just enough to stop it from slipping, but not enough to crush it. Manipulating the screw requires information about exactly where it lies against the fingers that it contacts. In both cases, **tactile sensing** (or touch sensing) can provide the needed information. Tactile sensing is the robotic version of the human sense of touch. A robot's tactile sensor uses an elastic material and a sensing scheme that measures the distortion of the material under contact. The sensor may give data at an array of points on the elastic surface, producing the analogue of a camera image, but of deformation rather than light intensity. By understanding the physics of the deformation process, it is possible to derive algorithms that are analogous to vision algorithms, and can compute position information for the objects that the sensor touches. Tactile sensors can also sense vibration, which helps to detect the impending escape of the coffee cup from the holder's grasp. Human beings use this scheme with a very fast servo loop⁵ to detect slip and control the grasping force to near the minimum needed to prevent slip.

Sonar

Sonar is SOund NAVigation and Ranging. Sonar provides useful information about objects very close to the robot and is often used for fast emergency collision avoidance. It is sometimes used to map the robot's environment over a larger area. In the latter case, an array of a dozen or more sonar sensors is fitted around the perimeter of the robot, each pointing in a different direction. Each sensor ideally measures the distance to the nearest obstacle in the direction it is pointing.

Sonar works by measuring the time of flight for a sound pulse generated by the sensor to reach an object and be reflected back. The pulse or "chirp" is typically about 50 kHz. This is more than twice the upper limit for humans of 20 kHz. Sound at that frequency has a wavelength of about 7 mm. The speed of sound is about 330 m/second, so the round-trip time delay for an object 1 m away is about 6×10^{-3} seconds. Sonar has been very effective for obstacle avoidance and tracking a nearby target, such as another mobile robot. But although it should be possible to measure the time delay very accurately, sonar has rarely been able to produce reliable and precise data for mapping. The first problem is beam width. Rather than a narrow beam of sound, a typical sensor produces a conical beam with 10° or more of spread. The second problem comes from the relatively long (7 mm) wavelength of the sonar sound. Objects that are very smooth

⁵ A servomechanism is a device for controlling a large amount of power with a small amount of power. A servo loop uses feedback to regulate the power.

relative to this wavelength look shiny or "specular" to the sensor. Such objects reflect sound like a perfect mirror. Sound will only be received back from patches of surface that are at right angles to the beam. Objects with flat surfaces and sharp edges reflect very little sound in most directions. (The same observation is used to design radar-eluding stealth aircraft and ships.) After being reflected from the surface, the sound may yet strike a rough surface and be reflected back to the sensor. The time delay will correspond not to a physical object, but to a "ghost," which may mysteriously disappear when the robot moves.

As discussed in Chapter 17, noisy sensors can be handled by first constructing a probabilistic model of the sensor, and then using Bayesian updating to integrate the information obtained over time as the robot moves around. Eventually, reasonably accurate maps can be built up, and ghost images can be eliminated.

Camera data

Human and animal vision systems remain the envy of all machine-vision researchers. Chapter 24 provides an introduction to the state of the art in machine vision, which is still some way from handling complex outdoor scenes and general object recognition. Fortunately, for a robot's purposes, something simpler than a general vision system will usually suffice. If the set of tasks the robot needs to perform is limited, then vision need only supply the information relevant to those tasks. Special-purpose robots can also take advantage of so-called **domain constraints** that can be assumed to apply in restricted environments. For example, in a building (as opposed to a forest), flat surfaces can be assumed to be vertical or horizontal, and objects are supported on a flat ground plane.

In some cases, one can also modify the environment itself to make the robot's task easier. One simple way of doing this, widely used in warehousing tasks, is to put bar-code stickers in various locations that the robot can read and use to get an exact position fix. Slightly more drastic is the use of **structured light sensors**, which project their own light source onto objects to simplify the problem of shape determination. Imagine a vertical light stripe cast as shown in Figure 25.7. When this stripe cuts an object, it produces a contour whose 3-D shape is easily inferred by triangulation from any vantage point not in the plane of the stripe. A camera placed in the same *horizontal* plane as the source needs only to locate the stripe within each horizontal scan line. This is a simple image-processing task and easily done in hardware.

By moving the stripe, or by using several rasters of stripes at different spacings, it is possible to produce a very dense three-dimensional map of the object in a short space of time. A number of devices are available now that include a laser source, stripe control, camera, and all the image processing needed to compute a map of distances to points in the image. From the user's point of view, these **laser range finders** really are depth sensors, providing a depth image that updates rapidly, perhaps several times a second.

For model-based recognition, some very simple light-beam sensors have been used recently. These sensors provide a small number of very accurate measurements of object geometry. When models are known, these measurements suffice to compute the object's identity and position. In Figure 25.8, two examples are shown, a **cross-beam sensor** and a **parallel-beam sensor**.

DOMAIN
CONSTRAINTS

STRUCTURED LIGHT
SENSORS

LASER RANGE
FINDERS

CROSS-BEAM
SENSOR
PARALLEL-BEAM
SENSOR

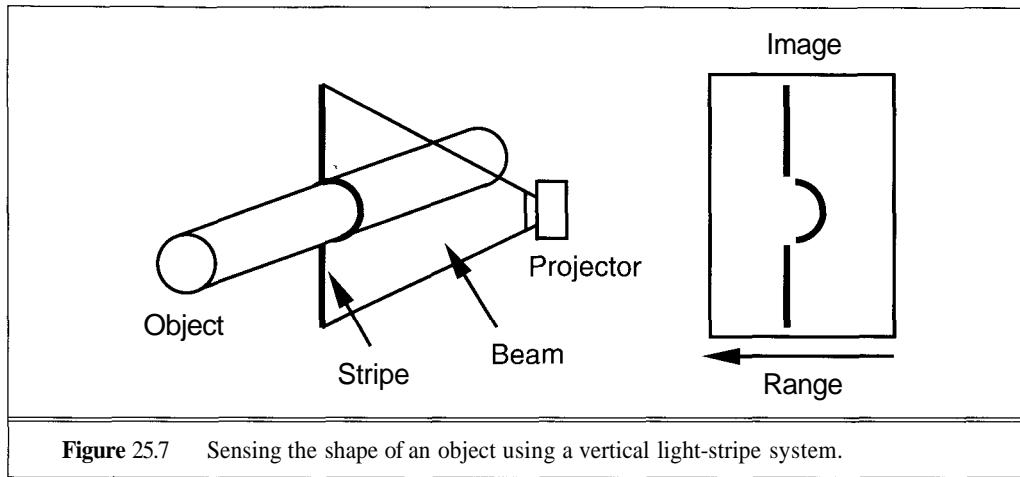


Figure 25.7 Sensing the shape of an object using a vertical light-stripe system.

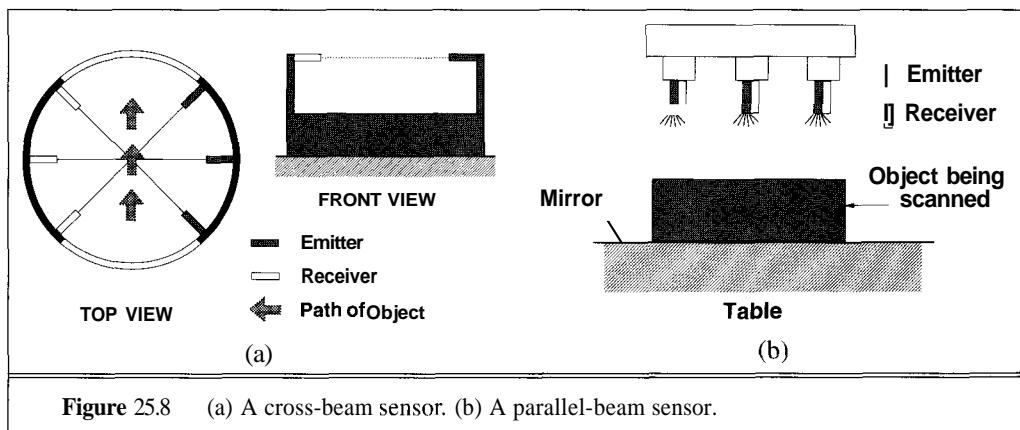


Figure 25.8 (a) A cross-beam sensor. (b) A parallel-beam sensor.

25.4 ARCHITECTURES



In this section, we step back from the nuts and bolts of robots and look at the overall control mechanism. *The architecture of a robot defines how the job of generating actions from percepts is organized.* We will largely be concerned with autonomous mobile robots in dynamic environments, for which the need for a sophisticated control architecture is clear.

The design of robot architectures is essentially the same agent design problem that we discussed in Chapter 2. We saw in the introduction to the current chapter that the environment for mobile robots is toward the difficult end as environments go. Furthermore, the perceptual input available to a robot is often voluminous; nevertheless, the robot needs to react quickly in some situations. In the following subsections, we briefly describe a variety of architectures, ranging

from fully deliberative to fully reflex. There is no accepted theory of architecture design that can be used to prove that one design is better than another. Many theoreticians deride the entire problem as "just a bunch of boxes and arrows." Nonetheless, many superficially different designs seem to have incorporated the same set of features for dealing with the real world.

Classical architecture

By the late 1960s, primitive but serviceable tools for intelligent robots were available. These included vision systems that could locate simple polyhedral objects; two-dimensional path-planning algorithms; and resolution theorem provers that could construct simple, symbolic plans using situation calculus. From these tools, together with a collection of wheels, motors and sensors, emerged Shakey, the forerunner of many intelligent robot projects.

The first version of Shakey, appearing in 1969, demonstrated the importance of experimental research in bringing to light unsuspected difficulties. The researchers found that general-purpose resolution theorem-provers were too inefficient to find nontrivial plans, that integrating geometric and symbolic representations of the world was extremely difficult, and that plans don't work. This last discovery came about because Shakey was designed to execute plans without monitoring their success or failure. Because of wheel slippage, measurement errors and so on, almost all plans of any length failed at some point during execution.

The second version of Shakey incorporated several improvements. First, most of the detailed work of finding paths and moving objects was moved from the general problem-solving level down into special-purpose programs called **intermediate-level actions** (ILAs). These actions in fact consisted of complex routines of **low-level actions** (LLAs) for controlling the physical robot, and included some error detection and recovery capabilities. For example, one ILA called NAVTO could move the robot from one place to another within a room by calling the A* algorithm to plan a path and then calling LLAs to execute the path, doing some path corrections along the way. The LLAs were also responsible for updating the internal model of the world state, which was stored in first-order logic. Motion errors were explicitly modelled, so that as the robot moved, its uncertainty about its location increased. Once the uncertainty exceeded a threshold for safe navigation, the LLA would call on the vision subsystem to provide a new position fix. The key contribution of the ILA/LLA system, then, was to provide a relatively clean and reliable set of actions for the planning system.

The planning system used the STRIPS algorithm, essentially a theorem-prover specially designed for efficient generation of action sequences. STRIPS also introduced the idea of compiling the results of planning into generalized **macro-operators**, so that future problem-solving could be more efficient (see Section 21.2). The entire system was controlled by PLANEX, which accepted goals from the user, called STRIPS to generate plans, then executed them by calling the specified ILAs. The execution mechanism was a simple version of the methods described in Chapter 12. PLANEX kept track of the current world state, comparing it to the preconditions of each subsequence in the original plan. After each action completed, PLANEX would execute the shortest plan subsequence that led to a goal and whose preconditions were satisfied. In this way, actions that failed would be retried, and fortunate accidents would lead to reduced effort. If no subsequence was applicable, PLANEX could call STRIPS to make a new plan.

The basic elements of Shakey's design—specialized components for low-level control and geometric reasoning, a centralized world model for planning, compilation to increase speed, and execution monitoring to handle unexpected problems—are repeated in many modern systems. Improvements in the symbolic planning and execution monitoring components are discussed in depth in Part IV. Compilation, or explanation-based learning, has been used extensively in two robot architectures: Robo-SOAR (Laird *et al.*, 1991) and THEO (Mitchell, 1990). Computation times for simple tasks can be reduced from several minutes to less than a second in some cases. In both of these architectures, compilation is invoked whenever a problem is solved for which no ready-made solution was available. In this way, the robot gradually becomes competent and efficient in routine tasks, while still being able to fall back on general-purpose reasoning when faced with unexpected circumstances.

Much of the research in robotics since Shakey was switched off has been at the level of ILAs and LLAs. Shakey was able to move on a flat floor, and could push large objects around with difficulty. Modern robots can twirl pencils in their fingers, screw in screws and perform high-precision surgery with greater accuracy than human experts. Sections 25.5 and 25.6 describe some of the theoretical advances in geometric reasoning and sensor integration that have made such achievements possible. Hilare II (Giralt *et al.*, 1991) is probably the most comprehensive modern example of a robot architecture in the classical tradition that makes use of advanced geometric methods.

Situated automata



SITUATED AUTOMATA

Since the mid-1980s, a significant minority of AI and robotics researchers have begun to question the "classical" view of intelligent agent design based on representation and manipulation of explicit knowledge. In robotics, *the principal drawback of the classical view is that explicit reasoning about the effects of low-level actions is too expensive to generate real-time behavior*. We have seen that compilation can alleviate this to some extent. Researchers working on **situated automata** have taken this idea one step further, by eliminating explicit deliberation from their robot designs altogether.

A situated automaton is essentially a finite-state machine whose inputs are provided by sensors connected to the environment, and whose outputs are connected to effectors. Situated automata provide a very efficient implementation of reflex agents with state. There have been two main strands in the development of such robots. The first approach involves generating the automaton by an offline compilation process, starting with an explicit representation. The second approach involves a manual design process based on a decomposition according to the various behaviors that the robot needs to exhibit.

The compilation approach, pioneered by Stan Rosenschein, (Rosenschein, 1985; Kaelbling and Rosenschein, 1990) distinguishes between the use of explicit knowledge representation by human designers (what he calls the "grand strategy") and the use of explicit knowledge within the agent architecture (the "grand tactic"). Through a careful logical analysis of the relationship between perception, action, and knowledge, Rosenschein was able to design a compiler that generates finite state machines whose internal states can be *proved* to correspond to certain logical propositions about the environment, provided that the initial state and the correct laws

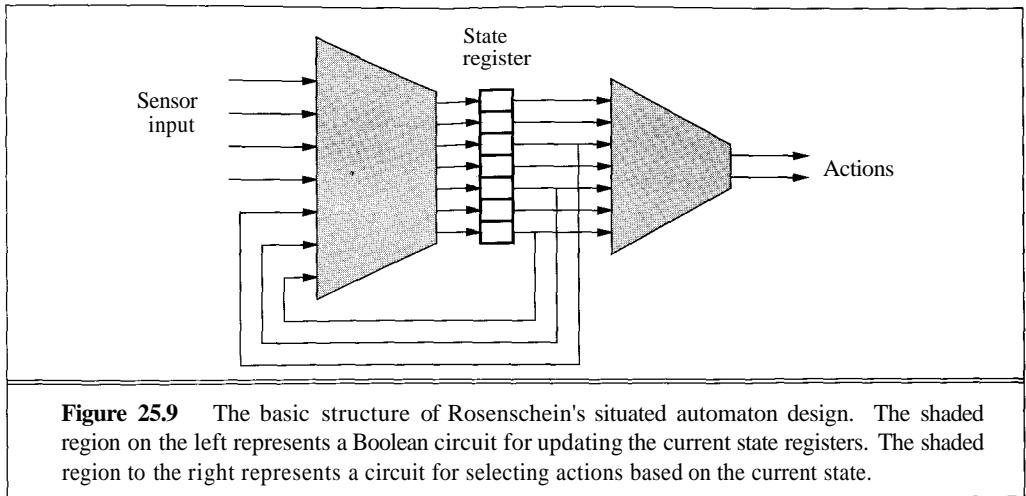


Figure 25.9 The basic structure of Rosenschein's situated automaton design. The shaded region on the left represents a Boolean circuit for updating the current state registers. The shaded region to the right represents a circuit for selecting actions based on the current state.

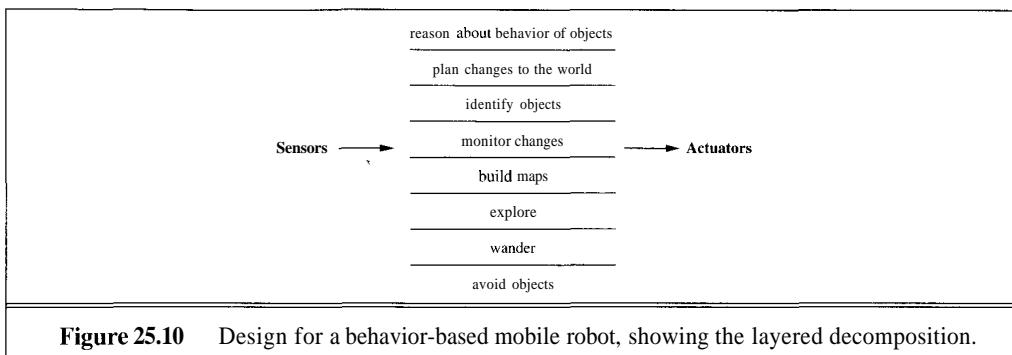
of "physics" are given to the compiler. Thus, the robot "knows" a set of propositions about the environment, even though it has no explicit representation of the proposition.

Rosenschein's basic design is shown in Figure 25.9. It relies on a theorem to the effect that *any* finite-state machine can be implemented as a state register together with a feedforward circuit that updates the state based on the sensor inputs and the current state, and another circuit that calculates the output given the state register. Because all the computation is carried out by fixed-depth, feedforward circuits, the execution time for each decision cycle is vanishingly small. Flakey, a robot based on situated automata theory, was able to navigate the halls of SRI, run errands, and even ask questions, all without the benefit of explicit representations.

Rodney Brooks (1986) has advocated an approach to robot design that he calls **behavior-based robotics**. The idea is that the overall agent design can be decomposed, not into functional components such as perception, learning, and planning, but into behaviors such as obstacle avoidance, wall-following, and exploration. Each behavioral module accesses the sensor inputs independently to extract just the information it needs, and sends its own signals to the effectors. Behaviors are arranged into a prioritized hierarchy in which higher level behaviors can access the internal state of lower level behaviors and can modify or override their outputs. Figure 25.10 shows a hierarchy of behaviors proposed for a mobile robot by Brooks (1986).

The main aim of behavior-based robotics is to eliminate the reliance on a centralized, complete representation of the world state, which seems to be the most expensive aspect of the classical architecture. Internal state is needed only to keep track of those aspects of the world state that are inaccessible to the sensors and are required for action selection in each behavior. For tasks in which the appropriate action is largely determined by the sensor inputs, the slogan "The world is its own model" is quite appropriate. In some cases, only a few bits of internal state are needed even for quite complex tasks such as collecting empty soft drink cans. Similar results have been found in Rosenschein's designs, in which the state register can be surprisingly small.

Behavior-based robotics has been quite successful in demonstrating that many basic competences in the physical world can be achieved using simple, inexpensive mechanisms. At



present, however, it is difficult to see how the design methodology can scale up to more complex tasks. Even for the task "Find an empty cup and bring it back to the starting location," the robot must form an internal representation that corresponds to a map in the classical architecture. Furthermore, the behavior-based approach requires the design of a new controller for each task, whereas classical robots are **taskable**: they can be assigned a goal and can carry out a plan to achieve it. In many ways, behavior-based designs resemble the ILA and LLA levels in Shakey, which are "nonclassical" components of an overall classical architecture.

Rosenschein's situated automata fall somewhere in between. The process of generating the automaton can be completely automated, so that if the compiler is included in the robot's software, the robot can be given a task and can compile its own situated automaton. The automaton can then be executed in software. Comparing this approach with the explanation-based learning method used in SOAR and THEO, we see that the situated automaton is compiled prior to execution and must handle all possible situations that can arise, whereas the explanation-based learning approach generates efficient rules for handling the types of situations that actually arise during execution. In simple domains, the situated-automaton approach is feasible and probably more efficient, whereas in very complex domains, especially those with recursive structure, the complete automaton becomes too large or even infinite in size.

25.5 CONFIGURATION SPACES: A FRAMEWORK FOR ANALYSIS

Recall from Chapter 3 that the main element in analyzing a problem is the **state space**⁶, which describes all the possible configurations of the environment. In robotics, the environment includes the body of the robot itself. The main distinction between the problems discussed in Chapter 3 and those in robotics is that robotics usually involves *continuous* state spaces. Both the configuration of the robot's body and the locations of objects in physical space are defined by real-valued

⁶ There is a source of confusion here because the term state space is used in robotics and control literature to include the robot's state parameters and certain of their derivatives. Enough derivatives are included that the robot's motion can be predicted from its dynamics and control equations. For simplicity, we assume here that derivatives of state parameters are excluded.



CONFIGURATION SPACE

coordinates. It is therefore impossible to apply standard search algorithms in any straightforward way because the numbers of states and actions are infinite. *Much of the work in robot planning has dealt with ways to tame these continuous state spaces.*

Suppose that we have a robot with k degrees of freedom. Then the state or configuration of the robot can be described with k real values q_1, \dots, q_k . For the PUMA robot, this would be a list of six joint angles $\theta_1, \dots, \theta_6$. The k values can be considered as a point p in a k -dimensional space called the **configuration space** of the robot. We use C to refer to the configuration space of the robot itself. This description is convenient because it lets us describe the complex three-dimensional shape of the robot with a single k -dimensional point. In other words, from the six joint angles, we can determine the exact shape of the entire PUMA robot. This works because the individual robot links are rigid—they rotate when the joint angles change, but they do not change size or shape.

Configuration space can be used to determine if there is a path by which a robot can move from one position to another. Consider the problem of getting the PUMA hand to replace a spark plug in a car. We have to find a path through three-dimensional physical space that will lead the hand to the right spot, without bumping into anything. But that is not enough—we need to make sure that the other links of the robot do not bump into anything either. This is a difficult problem to visualize in physical space, but it is easy to visualize in configuration space. We start by considering the points in C for which any part of the robot bumps into something. This set of points is called the **configuration space obstacle**, or \mathcal{O} . The set difference $C - \mathcal{O}$ is called free space, or \mathcal{F} , and is the set of configurations in which the robot can move safely. (Note that we also have to worry about the robot bumping into itself, or over-twisting power and data cables that pass through its joints. For these reasons, there are upper and lower limits on the joint angles of most robots, and corresponding bounds on the size of configuration space.)

Assume we have an initial point c_1 and a destination point c_2 in configuration space. The robot can safely move between the corresponding points in physical space *if and only if* there is a continuous path between c_1 and c_2 that lies entirely in \mathcal{F} . This idea is illustrated in Figure 25.11. A two-link robot is shown at the left of the figure, and its two-dimensional configuration space at the right. The configuration obstacle is the shaded region, and configurations c_1 and c_2 are shown in both domains, with a safe path drawn in C between them. Even in this simple example, the advantage of searching for a safe path in C is clear.

We humans have it much easier than the PUMA, because our shoulder joint has three degrees of freedom, while the PUMA's only has two. That gives the human arm seven degrees of freedom overall, one more than is needed to reach any point. This extra or **redundant** degree of freedom allows the arm to clear obstacles while keeping the hand in one spot, a wonderful facility that helped our ancestors climb around in trees, and helps us change spark plugs.

We should emphasize that configuration space and free space are mathematical tools for designing and analyzing motion-planning algorithms. The algorithms themselves need not explicitly construct or search a configuration space; nonetheless, any motion-planning method or control mechanism can be interpreted as seeking a safe path in \mathcal{F} . Configuration-space analysis can be used to establish whether or not a given mechanism is complete and correct, and can be used to analyze the complexity of the underlying problems.

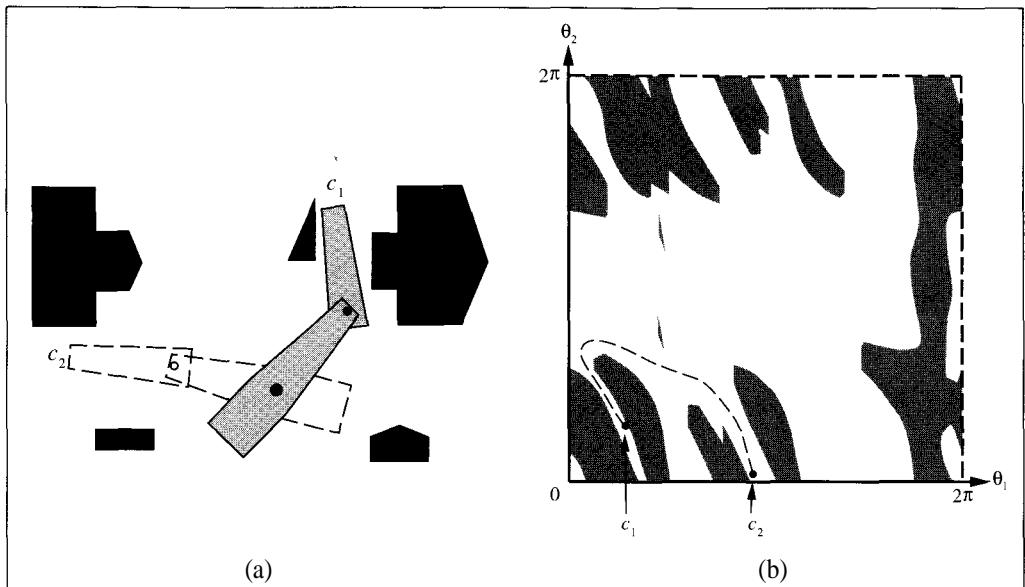


Figure 25.11 (a) A workspace with a rotary two-link arm. The goal is to move from configuration c_1 to configuration C_2 . (b) The corresponding configuration space, showing the free space and a path that achieves the goal.

Generalized configuration space

GENERALIZED CONFIGURATION SPACE

ASSEMBLY PLANNING

The term **generalized configuration space** has been applied to systems where the state of other objects is included as part of the configuration. The other objects may be movable, and their shapes may vary. Shape variation occurs in objects such as scissors or staplers that have mechanical joints, and in deformable objects like string and paper. Generalized configuration spaces are especially useful in understanding tasks such as **assembly planning** in which the robot must move a set of objects into some desired arrangement.

Let \mathcal{E} denote the space of all possible configurations of all possible objects in the world, other than the robot. If a given configuration can be defined by a finite set of parameters $\alpha_1, \dots, \alpha_m$, then \mathcal{E} will be an m -dimensional space. This will not work for objects like string and paper, whose shapes are not describable with finitely many parameters. These objects represent a considerable challenge for geometric modelling at this stage. But we can use \mathcal{E} as a conceptual tool even in those cases.

Now consider $\mathcal{W} = C \times \mathcal{E}$. \mathcal{W} is the space of all possible configurations of the world, both robot and obstacles. In the two-link robot C described earlier, there was no variation in the object shapes, so \mathcal{E} was a single point and \mathcal{W} and C were equivalent.

If all of the objects in the environment are robots under central control, then the situation is really the same as before. In this setting, it is best to cluster all the degrees of freedom of the robots together to create a single super-robot. The generalized configuration space is the space of all joint angles for all robots. Illegal configurations now include those where two robots overlap

in space. The planning problem again reduces to finding a safe path for a point in the generalized free space \mathcal{F} .

If the other objects are not robots but are nonetheless movable, the problem is very much harder. Unfortunately, this is also the most interesting case, because usually we would like a robot to move inanimate things around. We can still construct a generalized configuration space, and eliminate configurations where the robot overlaps an object, or two objects overlap. But even if we find a path between two configurations c_1 and c_2 in generalized \mathcal{F} , we may not be able to execute it. The problem is that an unrestricted path in W will usually describe motion of objects through midair without any help from the robot. We can partially address this problem by restricting W to comprise configurations where objects are either held by the robot or are supported by another object against gravity.

But legal paths in this W can still describe object motions without robot aid, such as spontaneous sliding on a tabletop, which should be illegal. There is no easy solution to this problem. We simply cannot think of W as a benign sea in which we can navigate freely. Instead, we must follow certain shipping lanes, between which we transfer only at special configurations. Although generalized W has many degrees of freedom, ($i + m$ degrees using the notation above), only k of these are actually controllable. Most of the time, we are changing only the robot configuration q_1, \dots, q_k , and the object configurations $\alpha_1, \dots, \alpha_m$ stay fixed. Sometimes though, if the robot is grasping an object, we can change both the robot and the objects configuration. We are still moving in a k -dimensional subset of W , but no longer one involving only the q (robot) coordinates. So we are in a $(k + m)$ -dimensional sea with steerage only along two types of k -dimensional lanes. Motions where the robot moves freely are called **transit paths**, and those where the robot moves an object are called **transfer paths**.

The navigable W in this case is called a **foliation**, suggesting an abundance of small randomly oriented sheets, but it is really more like a book. Each page of the book is a slightly different free space for the robot, defined by slightly different positions for the movable objects. Look again at Figure 25.12. On the left half of the figure, one of the objects has an arrow attached, indicating possible motion. The variable b is a coordinate measuring the object's displacement. The W for the system is now three-dimensional, and b is the new coordinate. For b fixed, a slice through the W describes the allowable motions of the robot with the obstacles fixed, and looks like the C in the right half of the figure. Imagine constructing the C for many different values of b , printing the results on separate pages, sorting them by b coordinate, and assembling the pages into a book. Transit motion is possible within any page of the book, on which the object positions are fixed, but not between pages. Transfer motions, in which an object is moving (or being moved), form a kind of spine for the book, and allow motion between pages.

The difficulty of planning for movable objects is real: the towers of Hanoi problem⁷ is a special case of it. The number of motions needed to move n Hanoi disks is exponential in n , implying that planning for movable objects requires time that grows at least exponentially with the dimension of W . At this time, there are no proved upper bounds for general planning with n movable objects. Robot planners usually make some strong assumptions to avoid tackling this problem head-on. One can do any of the following:

⁷ The towers of Hanoi problem is to move a set of n disks of different sizes from one peg to another, using a third peg for temporary storage. Disks are moved one at a time, and a larger disk cannot rest on a smaller one.

TRANSIT PATHS

TRANSFER PATHS

FOLIATION

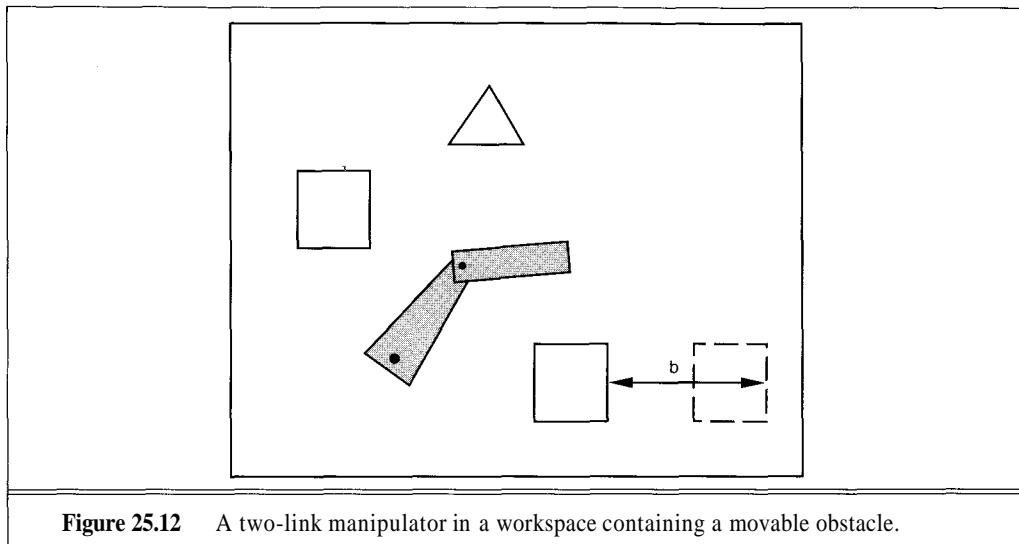


Figure 25.12 A two-link manipulator in a workspace containing a movable obstacle.

1. Partition \mathcal{W} into finitely many states—a form of **abstraction** (see Section 12.2). The planning problem then reduces to a logical planning problem of the kind addressed in Part IV. For example, the blocks world involves a partition of block configurations into those in which one block is "on another," and those in which it is "on the table." The exact locations of the blocks are ignored. The partitioning must be done carefully, because it involves a considerable loss of generality. In the blocks world case, one can no longer construct plans involving blocks that are leaning, or in which two or more blocks perch on another, or in which block positions are important (as in building a wall). Obviously, the appropriate partition depends on the goal, but as yet no general method for constructing partitions has been devised.
2. Plan object motions first and then plan for the robot. This could be called the classical approach to assembly planning. We first restrict the object motions so that all the objects move as two rigid subgroups, one representing the main assembly, and the other a sub-assembly that is being added to it. A single robot then has a good chance to be able to perform that step, and a separate planning phase generates the robot's motion.

Planning object motions also can be abstracted into discrete actions such as "screw part A onto part B." Many early assembly planners used this kind of representation, and are therefore able to use standard partial-order planning algorithms. They support a variety of constraints on the order of assembly steps, but questions of geometric feasibility are usually handled by separate geometric algorithms, or in some cases by the user.

Recently though, a very elegant method was described that plans a geometrically feasible assembly sequence and runs in time *polynomial* in n (Wilson and Schweikard, 1992). This scheme necessarily does not generate all possible sequences, because there are exponentially many. But a user can reject certain of the subassemblies it has chosen, and it will generate another feasible sequence just as fast that satisfies the user's strictures.

3. Restrict object motions. Here one chooses a parameterized family of basic motions, and searches for a sequence of such motions that will solve the planning problem. The best-known example of this is the “LMT” approach⁸ (Lozano-Pérez *et al.*, 1984). One must be careful about the choice of motions, because the problem will be intractable if they are too general, but unsolvable if they are too restrictive. LMT suggests the use of compliant motions, which produce straight-line motions in free space, but which follow the boundaries of configuration space when needed. LMT also explicitly models uncertainty in control and sensing. So the result of an LMT basic motion is not a single trajectory, but an envelope of possible trajectories that grows with time. LMT is described more fully in Section 25.6.

Recognizable Sets

Configuration space is a useful tool for understanding constraints induced by object shape. We have so far tacitly assumed that robot planning problems are all of the form "How do I get from here to there in W ?" But the reality is that most robots have poor knowledge of where they are in W . The robot has reasonably good knowledge of its own state (although often not accurately enough for precision work), but its knowledge of other objects comes second hand from sensors like sonar, laser range finders, and computer vision. Some objects simply may not be visible from the robot's current vantage point, so their uncertainty is enormous. Rather than a point in configuration space, our planner must start with a probability cloud, or an envelope of possible configurations. We call such an envelope a **recognizable set**.⁹

RECOGNIZABLE SET
ABSTRACT SENSOR

It will be useful to formalize this notion. An **abstract sensor** a is a function from the true world state W to the space of possible sensor values. A recognizable set is a set $\sigma^{-1}(s)$ of all world states in which the robot would receive the sensor reading s .¹⁰

If the sensor is perfect, that is, if it always produces the same sensor values in the same world state, the sets $\sigma^{-1}(s)$ form a **partition** of W . Distinct recognizable sets do not overlap, and their union is all of W . Unfortunately, because of noise, and because the chosen W often does not incorporate all of the factors that can affect sensor readings, the value returned by the sensor may not be a unique function of the state. To allow for this, we can treat a as a relation rather than a function. The relation is true of a state and a sensor reading if the sensor reading *could possibly* be returned in that world state. It still makes sense to define recognizable sets as $\sigma^{-1}(s)$, where we now take this to mean the set of states in which the sensor could return the value s . Now it no longer holds that distinct recognizable sets are disjoint, although their union is still all of W .

Recognizable sets simplify the problem of planning with uncertainty. A robot will always be in a recognizable set, because it will always have sensor readings available. The robot may also use memories of earlier sensor readings, but this is equivalent to a virtual sensor that provides both current and past readings. The (virtual) sensor readings determine uniquely which

⁸ LMT is named after the three authors.

⁹ **Recognizable** sets are to continuous domains what multiple state sets are to discrete problems (see Chapter 3).

¹⁰ Notice the close analogy with the idea of **possible worlds** introduced in the context of modal logic in Chapter 8. A recognizable set is essentially the set of possible worlds given what the robot knows about the world.

recognizable set $\sigma^{-1}(s_0)$ the robot is in. From there, the planner can determine where the robot might move next, given a motion command with uncertainty, and what the next sensor reading s_1 could be. Thinking of recognizable sets as states of the robot, the motion command caused a nondeterministic transition from the state $\sigma^{-1}(s_0)$ to one of the states $\sigma^{-1}(s_1)$. Although there are infinitely many such states with a continuous sensor, it is still possible to represent the possible transitions, and to find a correct plan if one exists. Unfortunately, the complexity of doing this is extremely high—in fact, doubly exponential in the number of plan steps (Canny and Reif, 1987).

25.6 NAVIGATION AND MOTION PLANNING

We now turn to the question of how to move around successfully. Given our analysis of robotics problems as motion in configuration spaces, we will begin with algorithms that handle \mathcal{C} directly. These algorithms usually assume that an exact description of the space is available, so they cannot be used where there is significant sensor error and motion error. In some cases, no description of the space is available until the robot actually starts moving around in it.

We can identify five major classes of algorithms, and arrange them roughly in order of the amount of information required at planning time and execution time:

- ◊ **Cell decomposition** methods break continuous space into a finite number of cells, yielding a discrete search problem.
- ◊ **Skeletonization** methods compute a one-dimensional "skeleton" of the configuration space, yielding an equivalent graph search problem.
- ◊ **Bounded-error planning** methods assume bounds on sensor and actuator uncertainty, and in some cases can compute plans that are guaranteed to succeed even in the face of severe actuator error.
- 0 **Landmark-based navigation** methods assume that there are some regions in which the robot's location can be pinpointed using landmarks, whereas outside those regions it may have only orientation information.
- ◊ **Online algorithms** assume that the environment is completely unknown initially, although most assume some form of accurate position sensor.

As always, we are interested in establishing some of the properties of these algorithms, including their soundness, completeness, and complexity. When we talk about the complexity of a planning method, we must keep in mind both offline costs (before execution) and the online cost of execution.

Cell decomposition

Recalling that motion planning for a robot reduces to navigating a point in free space \mathcal{F} , the basic idea of **celldecomposition** is easy to state:

1. Divide \mathcal{F} into simple, connected regions called "cells." This is the cell decomposition.
2. Determine which cells are adjacent to which others, and construct an "adjacency graph." The vertices of this graph are cells, and edges join cells that abut each other.
3. Determine which cells the start and goal configurations lie in, and search for a path in the adjacency graph between these cells.
4. From the sequence of cells found at the last step, compute a path within each cell from a point of the boundary with the previous cell to a boundary point meeting the next cell.

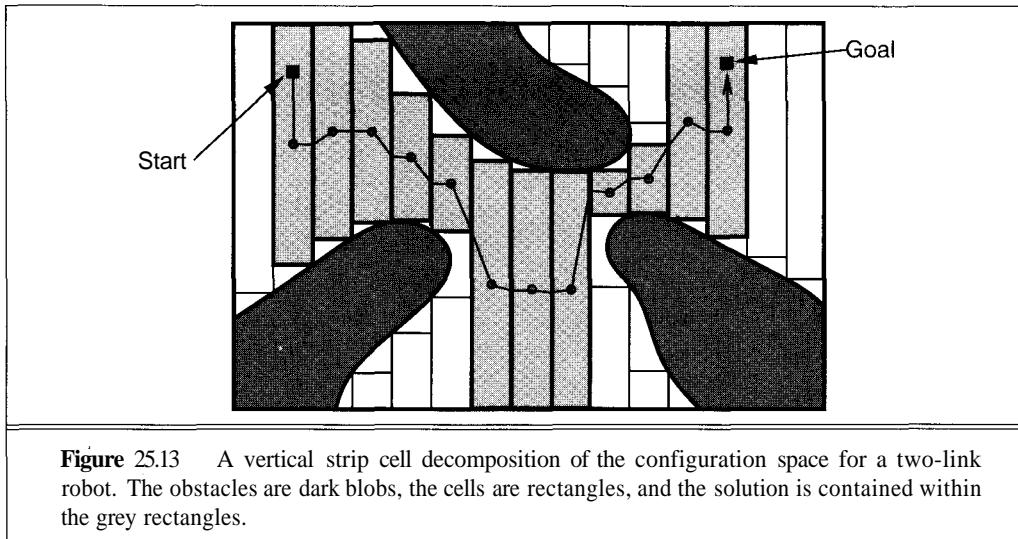
The last step presupposes an easy method for navigating within cells. The cells are usually geometrically "simple," so that this step is easy. For example, one could use rectangular cells, and then it is possible to join any two points in the cell with a straight-line path (this property is shared by all convex cells). The difficulty is that the cells must be constructed in configuration space, and \mathcal{F} typically has complex, curved boundaries (see Figure 25.12).

Because of the difficulty of the \mathcal{F} -boundary, the first approaches to cell decomposition did not represent it exactly. Approximate subdivisions were used, using either boxes or rectangular strips. A strip approximation to the configuration space of the 2-link robot is shown in Figure 25.13. The start and goal configurations are visible as points, and the cells joining them from step 3 above are shaded. Finally, an explicit path from start to goal is constructed by joining the midpoint (centroid) of each strip with the midpoints of the boundaries with neighboring cells.

This approach must be conservative if it is to produce a collision-free path. The strips must be entirely in free space or a path found inside them might cross a \mathcal{C} -boundary and cause a collision. So there will be some "wasted" wedges of free space at the ends of the strips. This is not usually a problem, but in very tight situations, there might be no path except through those wedges. To find this path, thinner strips would have to be used. This raises a general issue for approximate cell decomposition: choosing the resolution of the decomposition. This is sometimes done adaptively by the algorithm, by choosing smaller cells in "tight" parts of free space. Or there might be a natural "safety clearance" below which it is inadvisable to move. To be concrete about this, suppose that we are able to control our robot's motion to within 1 cm. If we cannot find a safe path with cells at 1 cm resolution, a path through cells at a finer resolution would take the robot closer than 1 cm to the obstacles. Because that conflicts with our desire for a sensible path, there is no point in performing that more expensive search.

The approach just described is sound but not complete. It produces a safe path, but may not always find one if one exists. With some small changes, we could instead have produced a planner that is complete but not sound. If we were reckless rather than conservative, we could have declared all partially free cells as being free. If there were any safe path, it would then pass entirely through free cells, and our planner would be guaranteed to find it. As you can imagine though, given the unforgiving nature of steel and aluminum during collisions, incorrect plans are not very useful.

An alternative to approximate algorithms is exact cell decomposition. An exact cell decomposition divides free space into cells that exactly fill it. These cells necessarily have complex shapes, because some of their boundaries are also boundaries of \mathcal{F} . Such a decomposition is shown in Figure 25.14. This decomposition looks rather like a coarse strip decomposition, but there are two important differences. The first is that the cells have curved top and bottom ends, so there are no gaps of free space outside the decomposition. We call these cells cylinders. The

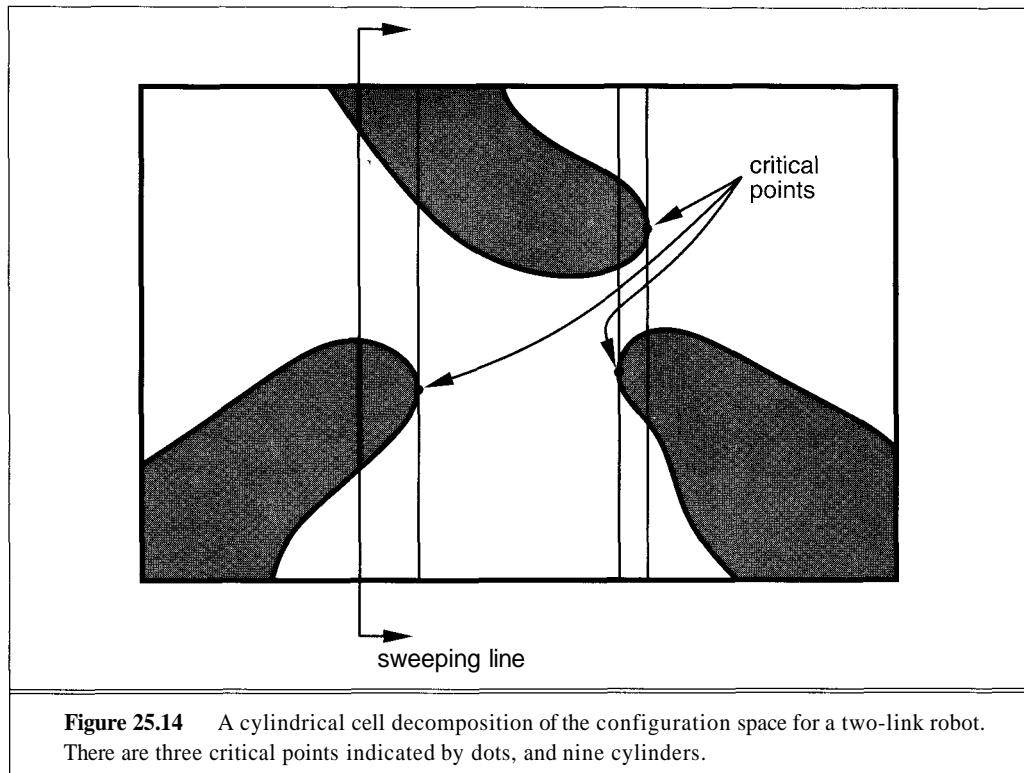


second difference is that the width of the cylinders is not fixed, but determined by the geometry of the environment. This is a key aspect of the exact method. To be useful for planning, the cell shapes must be kept simple. Otherwise, we could declare all of free space as a single cell. This does not help because there is no simple way to move around in such a cell. The cells in the cylindrical decomposition are easy to move in, because their left and right boundaries are straight lines (although sometimes of zero length).

To construct a cylindrical decomposition of a two-dimensional set, we first find **critical points** of the boundary. These are the points where the boundary curve is vertical. Equivalently, imagine sweeping a vertical line from left to right across Figure 25.14. At every instant, the line can be divided into segments that lie in free space or in the obstacle. Critical points are exactly those points where segments split or join as the line moves. As the line moves within a cylindrical cell, there are no splitting or joining events, and this makes it easy to plan paths within the cell. For example, to move from the left boundary to the right boundary of a cylindrical cell, we could use this sweeping line. A single segment of the line would always lie in this cell, and its midpoint would give us the path we want.

Skeletonization methods

Rather than producing a decomposition into a finite number of discrete chunks of space, **skeletonization** methods collapse the configuration space into a one-dimensional subset, or **skeleton**. They simplify the task of navigating in a high-dimensional space by requiring paths to lie along the skeleton. The skeleton is essentially a web with a finite number of vertices, and paths within the skeleton can be computed using graph search methods. If the start and goal points do not lie on the skeleton, short path segments are computed joining them to the nearest point on it. Skeletonization methods are generally simpler than cell decomposition, because they provide a



"minimal" description of free space. They avoid an explicit description of the boundary of free space, and this can provide considerable time savings. There is only one kind of data structure needed to describe skeleton curves, and this helps simplify implementation.

To be complete for motion planning, skeletonization methods must satisfy two properties:

1. If S is a skeleton of free space \mathcal{F} , then S should have a single connected piece within each connected region of \mathcal{F} .
2. For any point p in \mathcal{F} , it should be "easy" to compute a path from p to the skeleton.

The second condition is rather vague, but it will become clearer from the examples of skeletonization methods coming up. The condition is crucial, because otherwise we could construct a skeleton that consisted of a single point in each connected region of \mathcal{F} . Such a skeleton would be a procrastination of the planning problem, rather than a solution of it. There are many types of skeletonization methods in two dimensions. These include visibility graphs, Voronoi diagrams, and roadmaps. We briefly describe each in turn.

The **visibility graph** for a polygonal configuration space C consists of edges joining all pairs of vertices that can see each other. That is, there is an unobstructed straight line joining those vertices. It is shown in Figure 25.15. To see that the visibility graph is complete for planning, we need only observe that the *shortest* path between two points with polygonal obstacles lies entirely on the visibility graph, except for its first and last segment (see Exercise 25.7).

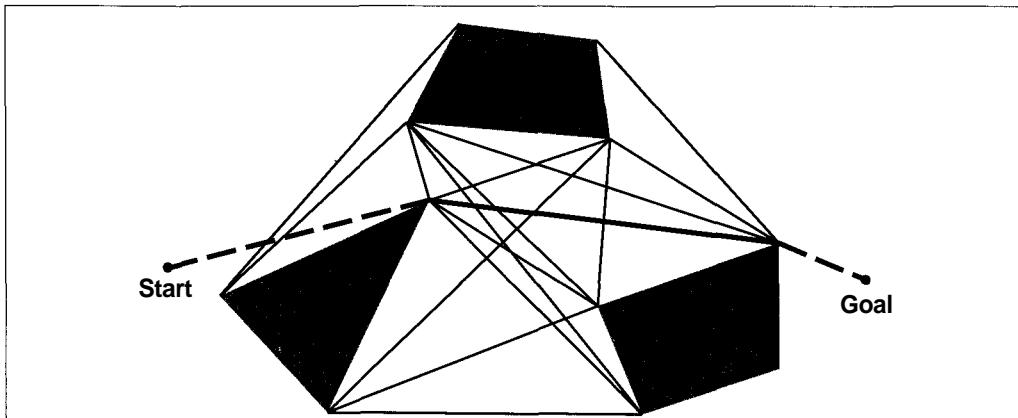


Figure 25.15 The visibility graph for a collection of polygons.

VORONOI DIAGRAM

The **Voronoi diagram** of a polygonal free space \mathcal{F} is shown in Figure 25.16. You can understand the diagram as follows. For each point in free space, compute its distance to the nearest obstacle. Plot that distance on Figure 25.16 as a height coming out of the page. The height of the terrain is zero at the boundary with the obstacles and increases as you move away from them. The bounding rectangle of this \mathcal{C} also counts as an obstacle. The terrain has sharp ridges at points that are equidistant from two or more obstacles. The Voronoi diagram consists of those sharp ridge points. Algorithms that find paths on this skeleton are complete, because the existence of a path in \mathcal{F} implies the existence of one on the Voronoi diagram. However, the path in the Voronoi diagram is not in general the shortest path.

ROADMAPS
SILHOUETTE CURVES
LINKING CURVES

SILHOUETTE METHOD

The most efficient complete method for motion planning is based on **roadmaps**. A roadmap is a skeleton consisting of two types of curves: **silhouette curves** (also known as "freeways") and **linking curves** (also known as "bridges"). The idea behind roadmaps is to make the search for a path simpler by limiting it to travel on a few freeways and connecting bridges rather than an infinite space of points. Figure 25.17 shows a roadmap for a three-dimensional torus. The complexity of computing a roadmap is $O(n^k \log n)$ for a robot with k degrees of freedom. Here, n is the size of the \mathcal{C} description, which is taken to be the number of equations in the formula describing \mathcal{F} .

There are two versions of roadmaps. The first has been called the **silhouette method** because it uses curves that define the visible silhouette of the free-space boundary. The roadmap shown in Figure 25.17 is of this type. To define it, we first choose two directions in \mathcal{C} , and think of them as coordinate axes. Call them X and Y . The roadmap is defined as follows:

- Silhouette curves are local extrema in Y of slices in X . That is, take a slice through \mathcal{F} by setting $X = c$ for some constant c . The cross-section of \mathcal{F} will have several connected pieces. Every piece is either unbounded, or will have at least one point where the Y -coordinate is locally maximized, and one where it is locally minimized. With some technical tricks, we can make sure that all the pieces are bounded. If we then compute all the local maxima and minima of \mathcal{F} in Y , we are guaranteed to get at least one point

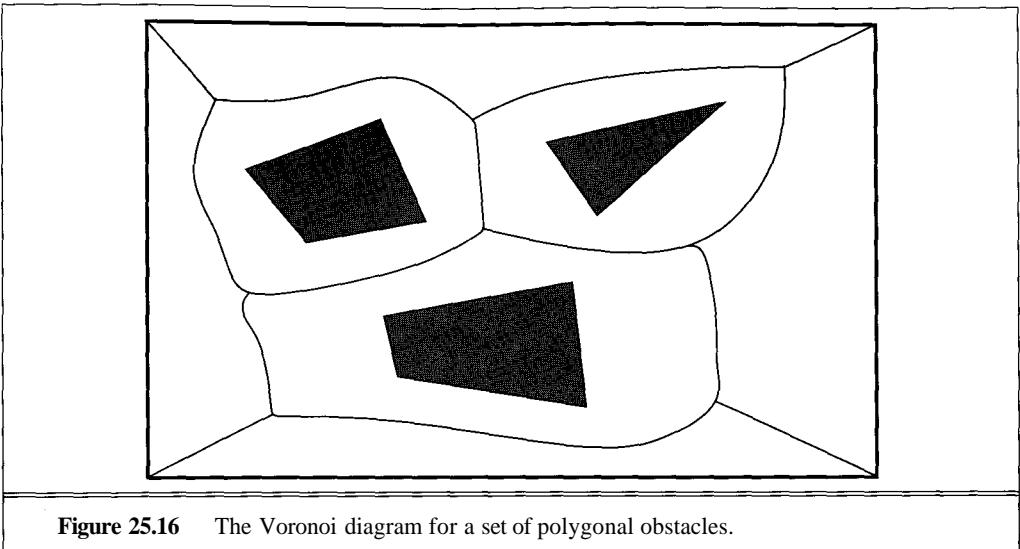


Figure 25.16 The Voronoi diagram for a set of polygonal obstacles.

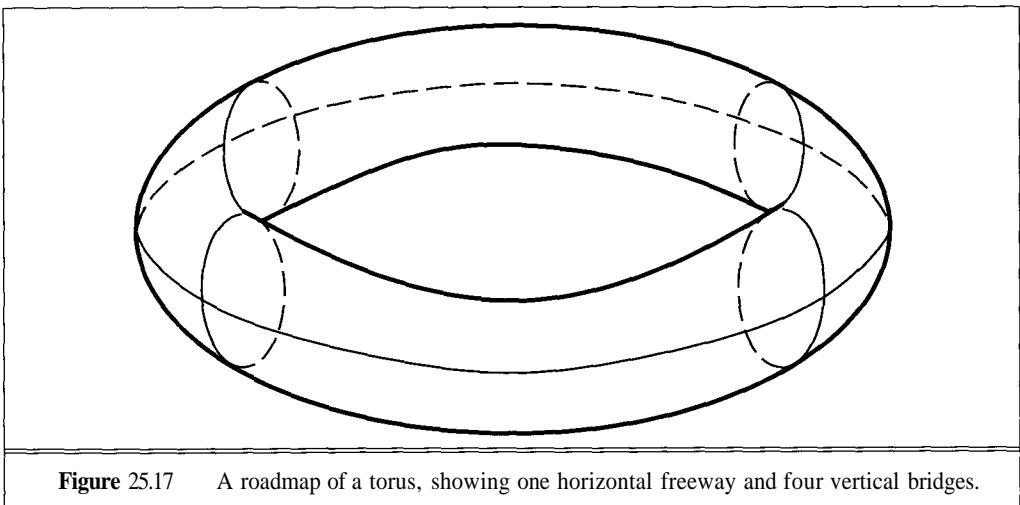


Figure 25.17 A roadmap of a torus, showing one horizontal freeway and four vertical bridges.

in every connected piece of \mathcal{F} 's cross section. See Figure 25.17. To get silhouette curves from these extremal points, we simply allow c to vary, and follow the extremal points in Y as the plane $X = c$ moves through \mathcal{F} .

- Linking curves join **critical points** to silhouette curves. Critical points are points where the cross section $X = c$ changes abruptly as c varies. By referring to Figure 25.17, the reader will find two critical points where the cross-section changes. If the critical point has X -coordinate c_0 , then for $X < c_0$, the cross-section is an hourglass-shaped curve. For $X > c_0$ side, the hourglass has pinched off into two circles. We will call a slice $X = c$ containing a critical point a "critical slice." Linking curves can be defined several ways, but

the easiest is to define a linking curve as the roadmap of a critical slice. Because roadmaps are defined in terms of linking curves, this is a circular definition. But because the linking curve is a roadmap of a slice that has one less dimension, the recursive construction is well-defined and terminates after a number of steps that equals the dimension of C .

One disadvantage of this definition is that it gives curves on the boundary of free space. This is undesirable from both efficiency and safety perspectives. A slightly better definition borrows some ideas from Voronoi diagrams. Instead of using extremals in Y to define the silhouette curves, it uses *extremals of distance from obstacles in slices $X - c$* . This sounds like the same definition as the Voronoi diagram, but there are slight differences in two dimensions, and major differences in more than two. The definition of linking curves also changes. They still link critical points to silhouette curves, but this time, linking curves are computed by moving away from a critical point along curves that follow the direction of maximum increase of the distance function. In other words, start from a critical point, and hill-climb in configuration space to a local maximum of the distance function. That point lies on a silhouette curve under the new definition. This kind of roadmap is exemplified in Figure 25.18.

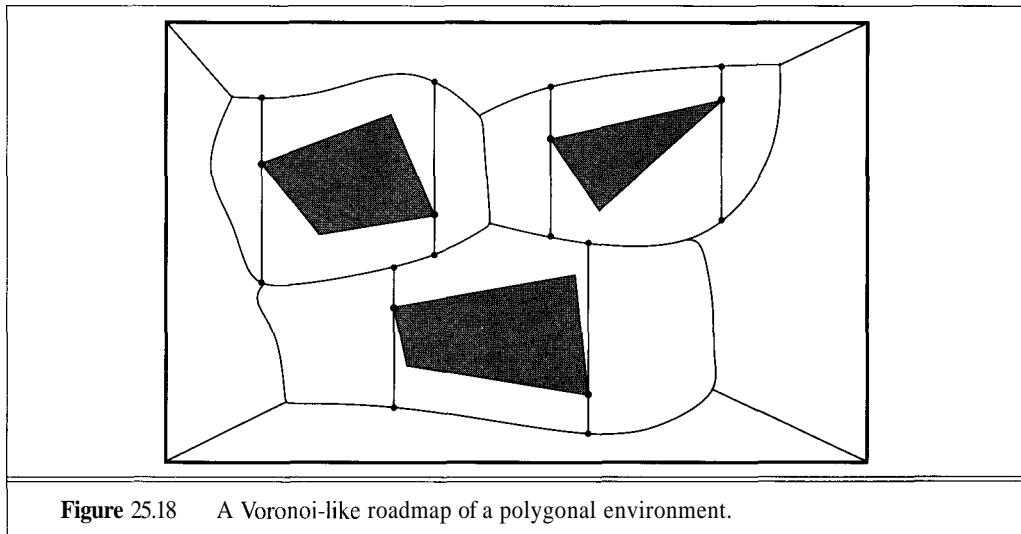


Figure 25.18 A Voronoi-like roadmap of a polygonal environment.

Fine-motion planning

Fine-motion planning (FMP) is about planning small, precise motions for assembly. At the distance scales appropriate for FMP, the environment is not precisely known. Furthermore, the robot is unable to measure or control its position accurately. Dealing with these uncertainties is the principal concern of FMP. Like online algorithms, fine-motion plans are strategies or policies that make use of sensing or environment shape to determine the robot's path at run time. However, whereas online algorithms assume nothing about the environment, partial knowledge is available to the fine-motion planner. This knowledge includes explicit models of the uncertainties in

sensing, control, and environment shape. A fine-motion planner does most of its work offline, generating a strategy that should work in all situations consistent with its models.

A fine-motion plan consists of a series of **guarded motions**. Each guarded motion consists of (1) a motion command and (2) a termination condition, which is a predicate on the robot's sensor values, and returns true to indicate the end of the guarded move. The motion commands are typically **compliant motions** that allow the robot to slide if the motion command would cause collision with an obstacle. Compliant motion requires a dynamic model such as a spring or damper. The spring model is the simplest to understand. Rather than moving the robot itself, imagine moving one end of a spring with the other end attached to the robot. The damper model is similar but instead of a spring, a device called a damper is attached to the robot. Whereas the spring gives a reaction force proportional to relative displacement of its endpoints, the damper gives a reaction force proportional to relative velocity. Both allow the robot to slide on a surface so long as the commanded velocity is not directly into the surface. (In reality, there will be some friction between robot and obstacle, and the robot will not move even if the commanded velocity is close to right angles with the surface.)

As an example, Figure 25.19 shows a 2-D configuration space with a narrow vertical hole. It could be the configuration space for insertion of a rectangular peg into a hole that is slightly larger. The motion commands are constant velocities. The termination conditions are contact with a surface. To model uncertainty in control, we assume that instead of moving at the commanded velocity, the robot's actual motion lies in the cone C_v about it. The figure shows what would happen if we commanded a velocity straight down from the start region s . Because of the uncertainty in velocity, the robot could move anywhere in the conical envelope, possibly going into the hole, but more likely landing to one side of it. Because the robot would not then know which side of the hole it was on, it would not know which way to move.

A more sensible strategy is shown in Figures 25.20 and 25.21. In Figure 25.20, the robot deliberately moves to one side of the hole. The motion command is shown in the figure, and the termination test is contact with any surface. In Figure 25.21, a motion command is given that causes the robot to slide along the surface and into the hole. This assumes we use a compliant motion command. Because all possible velocities in the motion envelope are to the right, the robot will slide to the right whenever it is in contact with a horizontal surface. It will slide down the right-hand vertical edge of the hole when it touches it, because all possible velocities are down relative to a vertical surface. It will keep moving until it reaches the bottom of the hole, because that is its termination condition. In spite of the control uncertainty, all possible trajectories of the robot terminate in contact with the bottom of the hole. (That is, unless friction or irregularities in the surface causes the robot to stick in one place.)

As one might imagine, the problem of *constructing* fine-motion plans is not trivial; in fact, it is a good deal harder than planning with exact motions. One can either choose a fixed number of discrete values for each motion or use the environment geometry to choose directions that give qualitatively different behavior. A fine-motion planner takes as input the configuration-space description, the angle of the velocity uncertainty cone, and a specification of what sensing is possible for termination (surface contact in this case). It should produce a multistep conditional plan or policy that is guaranteed to succeed, if such a plan exists.

We did not include uncertainty in the environment in our example, but there is one elegant way to do it. If the variation can be described in terms of parameters, those parameters can

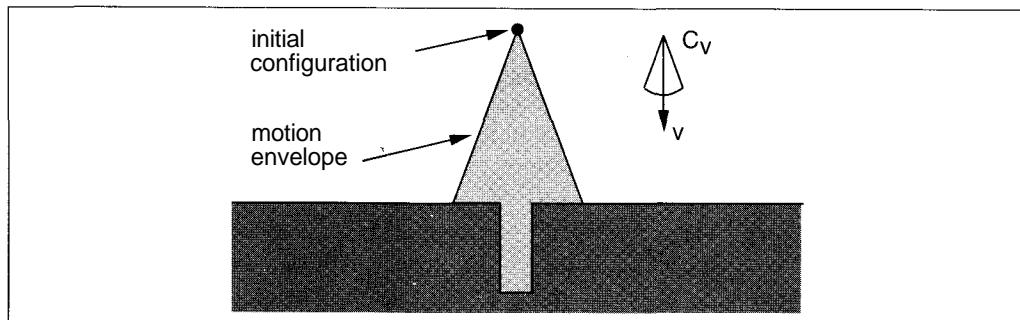


Figure 25.19 A 2-D environment, velocity uncertainty cone, and envelope of possible robot motions. The intended velocity is v , but with uncertainty the actual velocity could be anywhere in C_v , resulting in a final configuration somewhere in the motion envelope, which means we wouldn't know if we hit the hole or not.

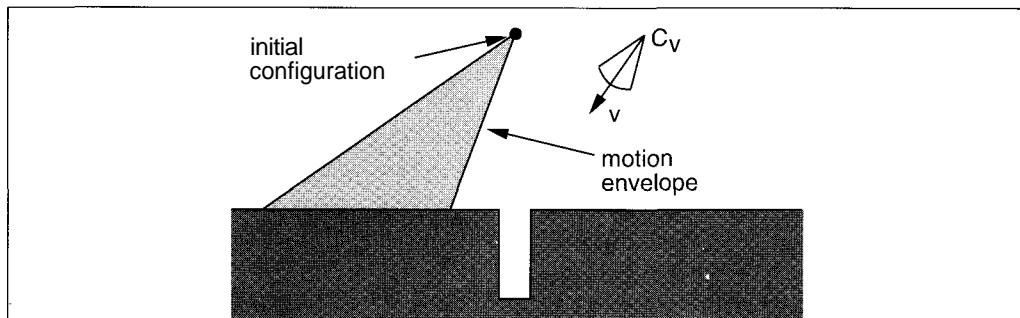


Figure 25.20 The first motion command and the resulting envelope of possible robot motions. No matter what the error, we know the final configuration will be to the left of the hole.

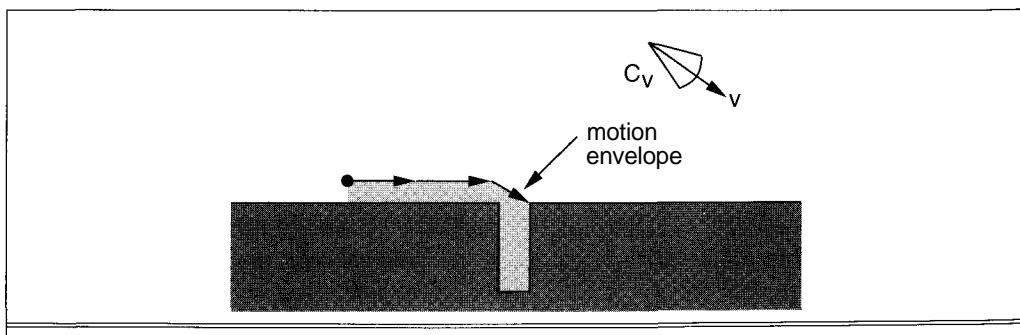


Figure 25.21 The second motion command and the envelope of possible motions. Even with error, we will eventually get into the hole.

be added as degrees of freedom to the configuration space. In the last example, if the depth and width of the hole were uncertain, we could add them as two degrees of freedom to the configuration space. It is impossible to move the robot in these directions in C or to sense its position directly. But both those restrictions can be incorporated when describing this problem as an FMP problem by appropriately specifying control and sensor uncertainties. This gives a complex, four-dimensional planning problem, but exactly the same planning techniques can be applied. Notice that unlike the decision-theoretic methods in Chapter 17, this kind of approach results in plans designed for the worst case outcome, rather than maximizing the expected quality of the plan. Worst-case plans are only optimal in the decision-theoretic sense if failure during execution is much worse than any of the other costs involved in execution.

Unfortunately, the complexity of fine-motion planning is extremely high. It grows exponentially not only with the dimension of configuration space, but also with the number of steps in the plan. FMP as described earlier also involves some tenuous assumptions about control and sensor uncertainty, uncertainty in environment shape, and capabilities of the run-time system. On the other hand, the FMP methodology can be applied to mobile robot navigation, and its assumptions seem more reasonable there. We will see some examples of this in the next section.

Landmark-based navigation

In the last two sections, we saw motion planning extended to include run-time decisions based on sensing. The sensors were assumed to be simple position or contact sensors, and to provide uniform accuracy across the environment. More complex sensors, such as vision and sonar, have very different attributes. It is much more difficult to model these sensors in a reasonable way, but we will present a couple of recent approaches.

LANDMARKS

In the first, called the **landmark model** of sensing, we assume that the environment contains easily recognizable, unique **landmarks**. A landmark is modeled as a point with a surrounding circular **field of influence**. Within the field of influence, the robot is able to know its position exactly. If the robot is outside all the fields of influence, it has no direct position information. This model might seem unrealistic at first, but it is good model for landmarks such as bar codes. Bar codes, like the ones on supermarket items, are sometimes used as landmarks for mobile robots in indoor environments. They are placed in strategic locations on walls, a few feet above the floor. They have unique codes enabling each to be distinguished. As long as the robot is close enough to recognize a bar code, it can get a good estimate of distance from that landmark. Getting good angular data is harder, but some estimate can be computed. Beyond the range of recognition of the code, the robot cannot get any information from it.

The robot's control is assumed to be imperfect. When it is commanded to move in a direction v , we assume the actual motion lies in a cone of paths centered on v . This is the same uncertainty model that we used in the last section. It is a very reasonable one for mobile robots. If the robot has a gyroscope or magnetic compass, the errors in its motion will be offsets in direction from the commanded motion. This is exactly what the uncertainty cone models.

An environment with landmarks is shown in Figure 25.22. This environment is known at planning time, but not the robot's position. We assume though that the robot lies somewhere inside a region (rectangular in the figure) that we do know. We plan a strategy for the robot by working

backwards from the goal. Figure 25.22 shows a commanded velocity v , and the **backprojection** of the goal region with respect to v . If the robot starts anywhere in the backprojection and moves with commanded velocity v , it will definitely reach the goal disk. Notice that the backprojection intersects a landmark D_1 . Because D_1 is the field of influence of a landmark, the robot has perfect position sensing inside D_1 . So if it reaches any part of D_1 , it can move reliably to the part of D_1 that intersects the backprojection of the goal. From there it can move in direction v to the goal.

This means that the robot can reach the goal from a region R if it can move reliably from R straight to the goal or if it can move reliably from R to disk D_1 . Continuing to work backwards, Figure 25.23 shows a backprojection of the union of the goal and D_1 relative to velocity command u . This new backprojection contains the start region S . This gives us a guaranteed strategy for reaching the goal. The execution of this strategy would be (1) move in direction u to D_1 and then (2) move in direction v to the goal.

As for fine-motion planning, we must choose these velocities by searching all possibilities. Fortunately, in this case, we do not have an exponential blowup with the number of plan steps. This is because the backprojections all have the same form, namely, they are backprojections of unions of disks. This gives a polynomial bound on the number of qualitatively different motion command directions. By using this observation, it is possible to plan in time that is polynomial in the number of disks. The plan itself will have at most n steps if there are n landmarks, because no landmark need be visited more than once. This planning method is both sound and complete.

Online algorithms

Most robot applications have to deal with some amount of uncertainty about the environment. Even robots used for manufacturing, which may have complete geometric models of robot and environment, have to perform high-precision tasks such as assembly. Assembly may require motions of a thousandth of an inch or less, and at this scale, models are far from accurate.

When the environment is poorly known, it is impossible to plan a path for the robot that will be collision-free and reach a goal under all circumstances.

Instead, one can try to produce a **conditional plan** (in the language of Chapter 13) or **policy** (in the language of Chapter 17) that will make decisions at run time. In some cases, it is possible to compute such a plan with no knowledge of the environment at all. This avoids the need for an offline planning stage, and all choices are made at run time. We will call such an algorithm an **online algorithm**. Online algorithms need to be simple, because they must make choices in real time. For that reason, they cannot "remember" much about their environment.

Despite their simplicity, online algorithms have been found that are both complete and "efficient." Efficiency for online algorithms can be defined in different ways. Because it depends on the environment, it must depend on measures of the environment's complexity. In Figure 25.24 we see a two-dimensional environment with start and goals points. The environment is not known to the robot when it begins, and it cannot "see" anything. It can only sense a boundary when it runs into it. The robot is equipped with a position sensor, and it knows where the goal is. Here is one complete online strategy:

1. Let $/$ be the straight line joining the initial position of the robot with its goal position. The robot begins to move toward the goal along $/$.



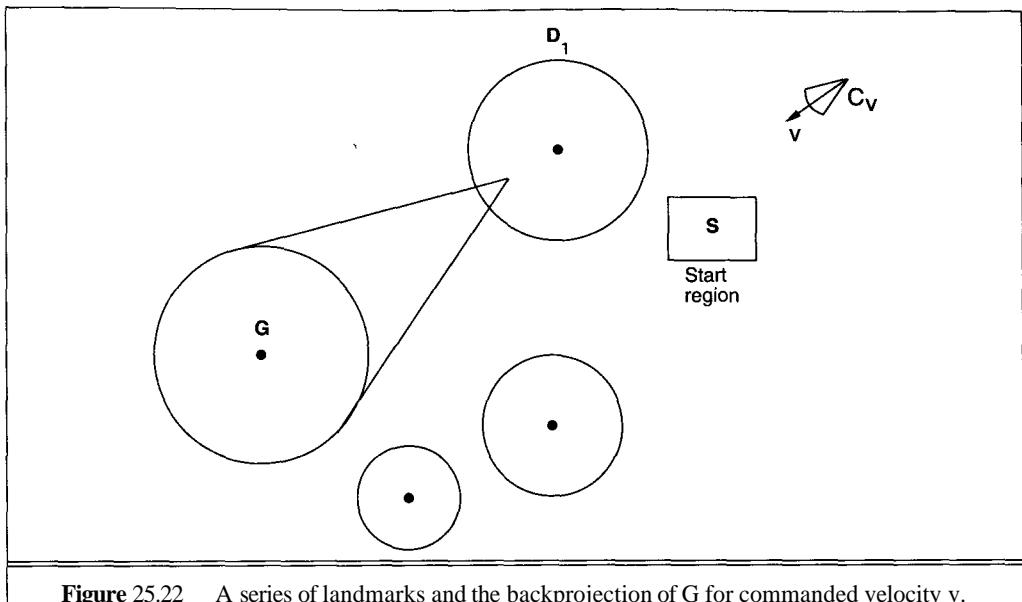


Figure 25.22 A series of landmarks and the backprojection of G for commanded velocity v .

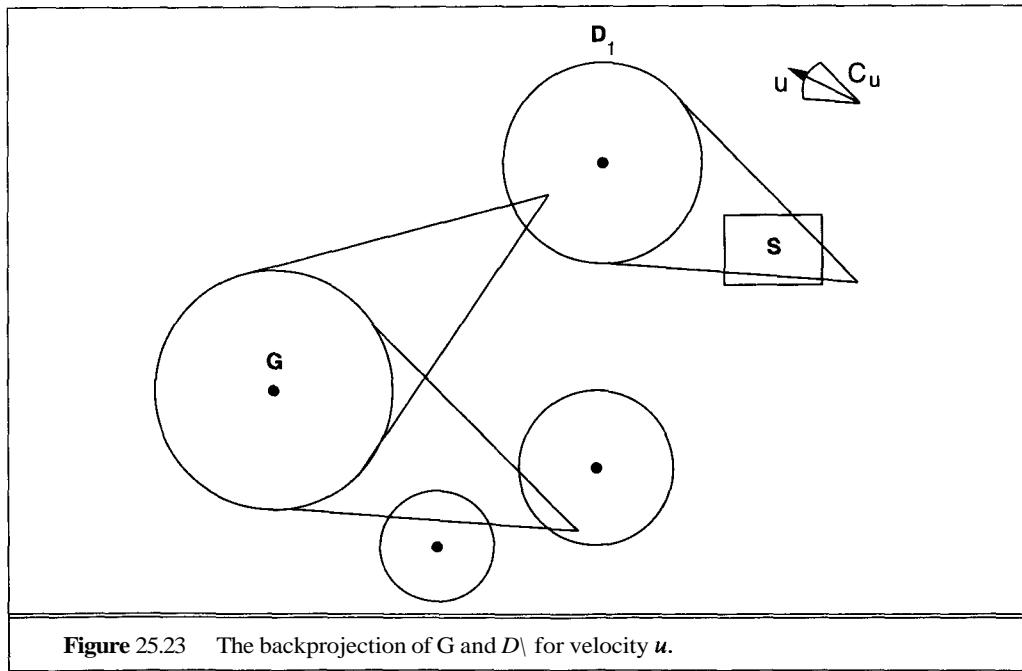


Figure 25.23 The backprojection of G and D_1 for velocity u .

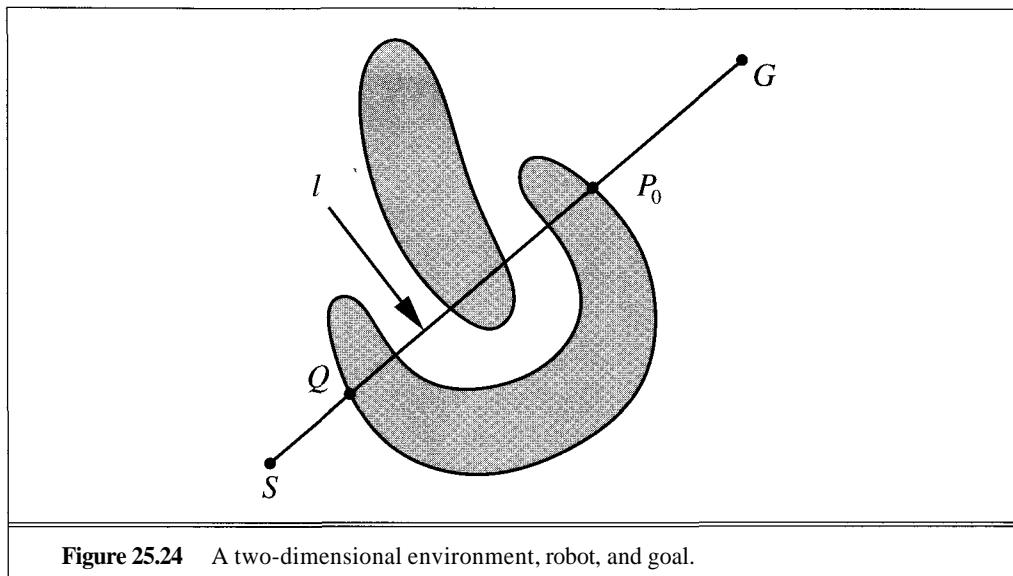


Figure 25.24 A two-dimensional environment, robot, and goal.

2. If the robot encounters an obstacle before it reaches the goal, it stops and records its current position Q . The robot then walks around the obstacle clockwise (the direction is not important) back to Q . During this walk, the robot records points where it crosses the line l , and how far it has walked to reach them. After the walk, let P_0 be the closest such point to the goal.
3. The robot then walks around the obstacle from Q to P_0 . Because it knows how far it walked to reach P_0 , it can decide whether it will get to P_0 faster by going clockwise or counterclockwise. Once it reaches P_0 , it starts moving toward the goal along l , and continues until it reaches another obstacle (in which case, it executes step 2 again) or until it reaches the goal.

COMPETITIVE RATIO

Online algorithms such as this are usually very fast in terms of computation time, but almost always give up any guarantee of finding the shortest path. Their efficiency is often measured using a **competitive ratio**. In the case of robot motion, one typically uses the worst-case ratio between the actual length of the path found, and the shortest path. For example, the preceding algorithm has a competitive ratio of at most $1 + 1.5B/|l|$, where B is the sum of the lengths of all the obstacle boundaries and $|l|$ is the length of the line l from start to goal.

It is fairly easy to see that this ratio can be very bad in some cases. For example, if an enormous obstacle protrudes just a teeny bit into the straight-line path, and the robot happens to start walking around it the wrong way, then the ratio can be unbounded. For some special cases, there are algorithms with competitive ratios bounded by a constant (Exercise 25.12), but there are some environments for which no algorithm has a finite competitive ratio. Fortunately for humans, these do not occur often in practice.

25.7 SUMMARY

Robotics is a challenging field for two reasons. First, it requires hardware (sensors and effectors) that actually work, a real challenge for mechanical engineering. Second, robots have to work in the physical world, which is more complex than most of the simulated software worlds that we have used for our examples in other chapters. Some robots finesse this problem by operating in a restricted environment. But modern autonomous robots with sophisticated sensors and effectors provide a challenging testbed for determining what it takes to build an intelligent agent.

- In general, the physical world is inaccessible, nondeterministic, nonepisodic, and dynamic.
- Robots have made an economic impact in many industries, and show promise for exploring hazardous and remote environments.
- Robots consist of a body with rigid **links** connected to each other by **joints**. The movement of the links is characterized by the degrees of freedom of the joints.
- Robots can have sensors of various types, including vision, force sensing, tactile (touch) sensing, sonar, and a sense of where their own body parts are.
- In a dynamic world, it is important to be able to take action quickly. Robots are designed with this in mind.
- The problem of moving a complex-shaped object (i.e., the robot and anything it is carrying) through a space with complex-shaped obstacles is a difficult one. The mathematical notion of a **configuration space** provides a framework for analysis.
- **Cell decomposition and skeletonization** methods can be used to navigate through the configuration space. Both reduce a high-dimensional, continuous space to a discrete graph-search problem.
- Some aspects of the world, such as the exact location of a bolt in the robot's hand, will always be unknown. **Fine-motion planning** deals with this uncertainty by creating a sensor-based plan that will work regardless of the exact initial conditions.
- Uncertainty applies to sensors at the large scale as well. In the **landmark model**, a robot uses certain well-known landmarks in the environment to determine where it is, even in the face of uncertainty.
- If a map of the environment is not available, then the robot will have to plan its navigation as it goes. **Online algorithms** do this. They do not always choose the shortest route, but we can analyze how far off they will be.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The word **robot** was popularized by Czech playwright Karel Čapek in his 1921 play *R.U.R.* (Rossum's Universal Robots). The theme of the play—the dehumanization of mankind in a technological society—appealed to audiences in war-torn Europe and the United States. The

robots, which were grown chemically rather than constructed mechanically, end up resenting their masters and decide to take over. It appears (Glanc, 1978) that it was actually Capek's brother, Josef, who first combined the Czech words "roboata" (obligatory work) and "robotnik" (serf) to yield "robot" in his 1917 short story *Opilec*.

Reichardt (1978) surveys the history and future of robots, grouping them into four categories: (1) strictly mythological, fictional, or fraudulent; (2) working, but nonelectronic; (3) controlled by very special-purpose electronic or electromechanical hardware; (4) controlled by general-purpose computers. Brief accounts of early robots of all four kinds are given by Raphael(1976), McCorduck (1979), and Heppenheimer (1985); more detailed treatments are given by Cohen (1966) and Chapuis and Droz (1958). The most famous classical instances of the first type are Talos (supposedly designed and built by Hephaistos, the Greek god of metallurgy) and the Golem of Prague. Perhaps the first impressive example of the second type was Jacques Vaucanson's mechanical duck, unveiled in 1738. An early instance of the third type is Torres y Quevedo's electromechanical automaton for playing the chess endgame of king and rook vs. king, described in Chapter 5. In the 1890s, Nikola Tesla built some experimental vehicles that were teleoperated (or remote controlled) via radio. Grey Walter's "turtle," built in 1948, could be considered the first modern type three robot.

In the late 1950s, George Engelberger and George Devol developed the first useful industrial robots, starting with type three, and moving on to type four. Engelberger founded Unimation to market them, and earned the title "father of robotics." His *Robotics in Practice* (1980) is a good survey of the early days of industrial robots. In the mid-1980s, there was a surge of interest in the field, largely funded by automotive companies. Reality did not live up to expectations, and there was a major shakeout in the robotics industry in the late 1980s. Perhaps in reaction to this shakeout, Engelberger's *Robotics in Service* (1989) is much more sanguine about the imminent practicality of type four robots in industrial settings.

HAND-EYE MACHINES

Type four robots can be further divided into mobile robots (or mobots) and static manipulators, originally called **hand-eye machines**. The first modern mobile robot was the "Hopkins Beast," built in the early 1960s at Johns Hopkins University. It had pattern-recognition hardware and could recognize the cover plate of a standard AC power outlet. It was capable of searching for outlets, plugging itself in, and then recharging its batteries! Still, the Beast had a limited repertoire of skills. The first general-purpose mobot was "Shakey," developed at SRI International from 1966 through 1972 (Nilsson, 1984).

The first major effort at creating a hand-eye machine was Heinrich Ernst's MH-1, described in his MIT Ph.D. thesis (Ernst, 1961) and in a retrospective by Taylor (1989). The Machine Intelligence project at Edinburgh (Michie, 1972) also demonstrated an impressive early system for vision-based assembly called FREDDY.

Robotics engages virtually every component and subfield of AI. Some areas of AI were originally driven primarily by the demands of robotics, although they have since become separate areas of study. The main two are computer vision (and other forms of perception) and planning. The Shakey robot project at SRI, in particular, was seminal in the development of the techniques of planning. There are also several problem areas that are unique to robotics. Planning in continuous state spaces is usually restricted to robotics. Sensor and motion errors are taken much more seriously in robotics, although they are also studied for military applications. Also, quite apart from perception and planning, it is far from trivial simply to describe the motions

one wishes a robot arm (for instance) to undertake, and then design physical devices and control systems to carry out the motions described. The mechanics and control of multilink robot arms are among the most complex problems studied in applied mathematics today.

Robot Manipulators: Mathematics, Programming, and Control (Paul, 1981) is a classic guide to the basics of robot arm design. Yoshikawa (1990) provides a more up-to-date text in this area. Latombe (1991) presents a good specialized textbook on motion planning. The robot motion planning problem, stated in the most natural way, is PSPACE-hard (Reif, 1979) (see page 853 for a description of PSPACE). Canny (1988) gives a book-length treatment of the computational complexity of robot motion planning, dealing with a number of ways of stating the problem. The major conference for robotics is the *IEEE International Conference on Robotics and Automation*. Robotics journals include *IEEE Robotics and Automation*, the *International Journal of Robotics Research*, and *Robotics and Autonomous Systems*.

EXERCISES

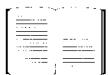
25.1 (This exercise was first devised by Michael Genesereth and Nils Nilsson.) Humans are so adept at basic tasks such as picking up cups or stacking blocks that they often forget what it was like to try such things as newborn babies. The idea of this exercise is to make explicit some of the difficulties involved, in a very direct manner. As you solve these difficulties, you should find yourself recapitulating the last 20 years of developments in robotics.

First, pick a task. The task should not be too difficult! (For example, making a column from three cereal boxes standing on end takes over an hour.) Set up the initial environment. Then, build a robot out of four other humans as follows:

- ◊ **Brain:** the job of the Brain is to come up with a plan to achieve the goal, and to direct the hands in the execution of the plan. The Brain receives input from the Eyes, but *cannot see the scene directly*.
- ◊ **Eyes:** the Eyes' job is to report a brief description of the scene to the Brain. The Eyes should stand a few feet away from the working environment, and can provide qualitative descriptions (such as "There is a red box standing on top of a green box, which is on its side") or quantitative descriptions ("The green box is about two feet to the left of the blue cylinder"). Eyes can also answer questions from the Brain such as, "Is there a gap between the Left Hand and the red box?" The Eyes *should not know what the goal is*.
- 0 **Hands** (Left and Right): one person plays each Hand. The two Hands stand next to each other; the Left Hand uses only his or her left hand, and the Right Hand only his or her right hand. The Hands execute only simple commands from the Brain—for example, "Left Hand, move two inches forward." They cannot execute commands other than motions; for example, "Pick up the box" is not something a Hand can do. To discourage cheating, you might want to have the hands wear gloves, or have them operate tongs. As well as being ignorant of the goal, the Hands must be *blindfolded*. The only sensory capability they have is the ability to tell when their path is blocked by an immovable obstacle such as a table or the other Hand. In such cases, they can beep to inform the Brain of the difficulty.

We have given only the most basic protocol. As the task unfolds, you may find that more complex protocols are needed in order to make any progress. What you learn about robotics will depend very much on the task chosen and on how closely you stick to the protocols.

25.2 Design and sketch a mobile robot for an office environment. Your robot should be able to perform typical office tasks like retrieving files, books, journals, and mail, and photocopying. What types of arm/hand did you choose? Consider carefully the trade-off between flexibility and custom design for those tasks. What kinds of sensing did you choose?



25.3 Consider the problem of designing a robot that will keep an office building tidy by periodically collecting and emptying bins and picking up empty cans and paper cups.

- Describe an appropriate physical design for the robot.
- Define a suitable performance evaluation measure.
- Propose an overall architecture for the robot agent. Mitchell (1990) and Brooks (1989) describe two different approaches to the problem.

25.4 Calculate the number of degrees of freedom of your hand, with the forearm fixed.

25.5 Consider the arm of a record player as a two-link robot. Let θ_1 measure the elevation of the arm, and θ_2 measure its horizontal rotation. Sketch the configuration space of this robot with θ_1 and θ_2 axes. Include the configuration space obstacles for the platter and spindle.

25.6 Draw the cylindrical cell decomposition for the environment in Figure 25.25. With n boards in such an arrangement, how many regions would there be in the cell decomposition?

25.7 If you have not done them already, do Exercises 4.13 to 4.15. Also answer the following:

- Prove rigorously that shortest paths in two dimensions with convex polygonal obstacles lie on the visibility graph.
- Does this result hold in three dimensions? If so, prove it. If not, provide a counterexample.
- Can you suggest (and prove) a result for the shortest paths in two dimensions with *curved* obstacles?

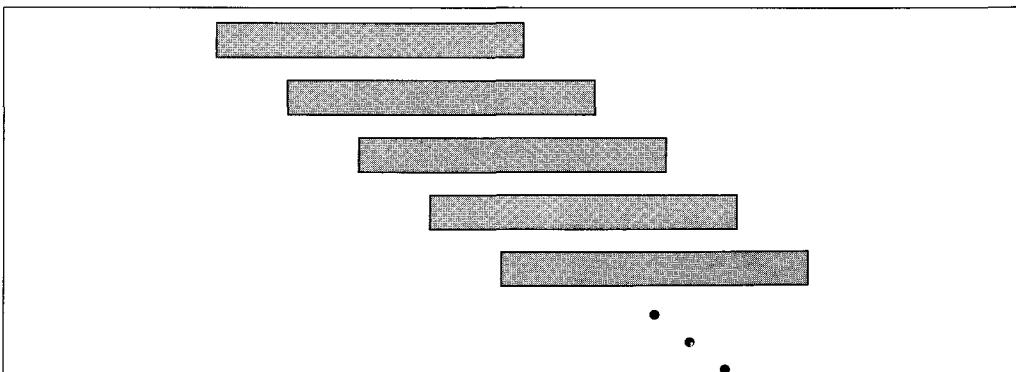


Figure 25.25 Row of boards environment.

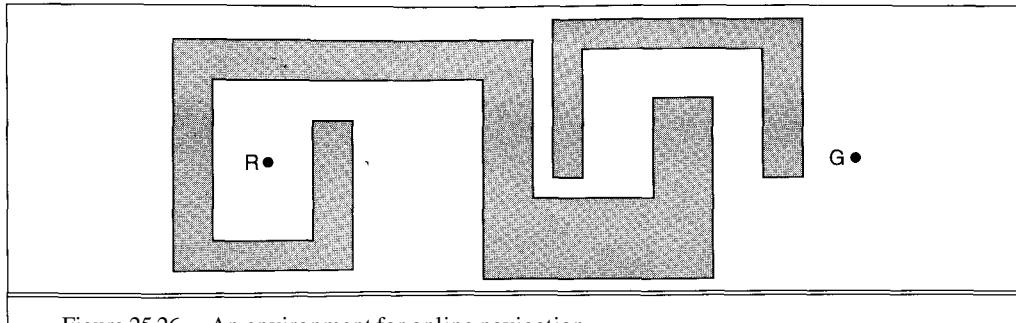


Figure 25.26 An environment for online navigation.

- d. Can the two-dimensional path-planning problem be solved using Voronoi diagrams instead of visibility graphs? Explain the modifications that would be needed to the overall system.

25.8 Suppose a robot can see two point landmarks L_1 and L_2 , and that it measures the angle between them as 17° . What is the shape of the recognizable set corresponding to this sensor reading? What is the shape of the recognizable set if the angle measurement has an uncertainty of $\pm 1^\circ$? Sketch this situation.

25.9 For the environment in Figure 25.26, sketch the path taken by the robot in executing the online navigation strategy. This strategy always completely circumnavigates any obstacle it encounters, which is often unnecessary. Try to think of another strategy that will travel less distance around some obstacles. Make sure your strategy will not get stuck in cycles. Your strategy will probably have a worse worst-case bound than the one presented in this chapter, but should be faster in typical cases.



25.10 Implement a general environment in which to exercise the online navigation algorithm, such that arbitrary obstacles can be placed in the environment. Construct a specific environment corresponding to Figure 25.26. Then implement an agent that incorporates the online algorithm, and show that it works.

25.11 Explain how to approach the problem of online navigation for a cylindrical robot of significant diameter. Does the point-robot algorithm need to be modified? What happens when the robot is heading down a passageway that becomes too narrow to get through?

25.12 We stated in our discussion of online algorithms that some special classes of environments are amenable to online algorithms with constant competitive ratios. An example is shown in Figure 25.27, which shows a robot on the bank of a river. We assume that the robot has an exact position sensor. The robot is trying to find the bridge, which it knows is somewhere nearby, but is not sure how far and in which direction. Unfortunately, there is a thick fog and the robot cannot see the bridge unless it stumbles right onto it.

- Describe an online algorithm for the robot that is guaranteed to find the bridge after a finite search, no matter where the bridge is.
- Calculate the total distance traversed by the robot using your algorithm, and compute its competitive ratio.

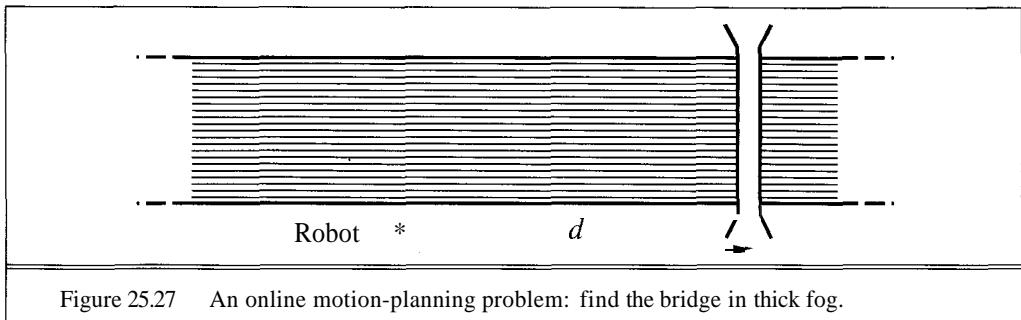


Figure 25.27 An online motion-planning problem: find the bridge in thick fog.

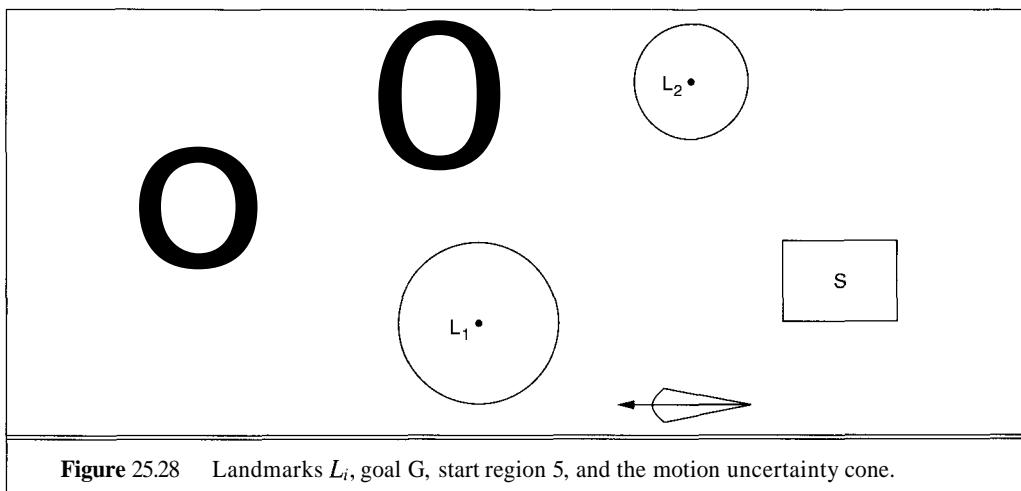


Figure 25.28 Landmarks L_i , goal G , start region S , and the motion uncertainty cone.

25.13 For the environment and motion uncertainty cone shown in Figure 25.28, find a navigation strategy that is guaranteed to reach the goal G .

25.14 Here are the three laws of robotics from the science fiction book */, Robot* (Asimov, 1950):

1. A robot may not injure a human being or through inaction allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings except where such orders would conflict with the first law.
3. A robot must protect its own existence as long as such protection does not conflict with the first or second law.

So far, only a few attempts have been made to build such safeguards into robotic software (Weld and Etzioni, 1994).

- a. How would you design a robot to obey these laws? What type of architecture would be best? What aspects of the laws are hard to obey?
- b. Are the laws sensible? That is, if it were possible to build a sophisticated robot, would you want it to be governed by them? Can you think of a better set of laws?

Part VIII

CONCLUSIONS

In this part we summarize what has come before, and give our views of what the future of AI is likely to hold. We also delve into the philosophical foundations of AI, which have been quietly assumed in the previous parts.

26 PHILOSOPHICAL FOUNDATIONS

In which we consider what it means to think and to be conscious, and whether artifacts could ever do such things.

26.1 THE BIG QUESTIONS

As we mentioned in Chapter 1, philosophers have been around for much longer than computers, and have been trying to resolve many of the same questions that AI and cognitive science claim to address: How *can* minds work, how *do* human minds work, and can nonhumans have minds? For most of the book we have concentrated on getting AI to work at all, but in this chapter, we address the big questions.

Sometimes the questions have led philosophers and AI researchers to square off in heated debate. More often, the philosophers have debated each other. Some philosophers have sided with the computational approach provided by artificial intelligence, partly because it has the tools and the inclination to give detailed, causal explanations of intelligent behavior.¹

It is rather as if philosophers were to proclaim themselves expert explainers of the methods of a stage magician, and then, when we ask them to explain how the magician does the sawing-the-lady-in-halftrick, they explain that it is really quite obvious: the magician doesn't really saw her in half; he simply makes it appear that he does. "But how does he do *that*?" we ask. "Not our department," say the philosophers. (Dennett, 1984)

... among philosophers of science one finds an assumption that machines can do everything that people can do, followed by an attempt to interpret what this bodes for the philosophy of mind; while among moralists and theologians one finds a last-ditch retrenchment to such highly sophisticated behavior as moral choice, love and creative discovery, claimed to be beyond the scope of any machine. (Dreyfus, 1972)

¹ At a recent meeting of the American Philosophical Association, it was put to us by one philosopher, who may prefer to remain nameless, that "Philosophy has already capitulated to AI."

Other philosophers have openly derided the efforts of AI researchers and, by implication, their philosophical fellow travellers:

Artificial intelligence pursued within the cult of computationalism stands not even a ghost of a chance of producing durable results . . . it is time to divert the efforts of AI researchers—and the considerable monies made available for their support—into avenues other than the computational approach. (Sayre, 1993)

After fifty years of effort, however, it is clear to all but a few diehards that this attempt to produce general intelligence has failed. (Dreyfus, 1992)

The nature of philosophy is such that clear disagreements can continue to exist unresolved for many years. If there were any simple experiment that could resolve the disagreements, the issues would not be of philosophical interest.

We will begin in Section 26.2 by looking at philosophical questions concerning which the interests of AI and philosophy coincide—the "how to" questions concerning the basic capabilities of perception, learning, and reasoning. This will help to clarify the concepts used later.

In Section 26.3, we look at the question "Can machines be made to act *as if* they were intelligent." The assertion that they can is called the **weak AI** position. Arguments against weak AI make one of these three claims:

- There are things that computers cannot do, no matter how we program them.
- Certain ways of designing intelligent programs are bound to fail in the long run.
- The task of *constructing* the appropriate programs is infeasible.

The arguments can be, and sometimes have been, refuted by exhibiting a program with the supposedly unattainable capabilities. On the other hand, by making AI researchers think more carefully about their unstated assumptions, the arguments have contributed to a more robust methodology for AI.

WEAK AI

STRONG AI

In Section 26.4, we take on the real bone of contention: the **strong AI** position, which claims that machines that act intelligently cannot have real, conscious minds. Debates about strong AI bring up some of the most difficult conceptual problems in philosophy.

Note that one could legitimately believe in strong AI but not weak AI. It is perfectly consistent to believe that it is infeasible to build machines that act intelligently, but to be willing to grant such a machine full consciousness if it could in fact ever be built.

We will try to present the arguments in their simplest forms, while preserving the lunges and parries that characterize philosophical debates. We do not have space to do them full justice, and we encourage the reader to consult the original sources. A caution: one of the most important things to keep in mind when reading the contributions to a philosophical debate is that what often appears to be an attempt to refute some proposition (such as "computers understand English") is actually an attempt to refute a *particular justification* for the proposition, or an attempt to show that the proposition is merely possible rather than logically necessary. Thus, many of the debates are not what they may seem at first, but there is still plenty to argue about.

26.2 FOUNDATIONS OF REASONING AND PERCEPTION

PHYSICALISM
MATERIALISM
BIOLOGICAL NATURALISM

FUNCTIONALISM

HOMUNCULI

INFINITE REGRESS

Almost all parties to the AI debate share a certain amount of common ground regarding the relation of brain and mind. The common ground goes under various names—**physicalism**, **materialism**, and **biological naturalism** among others—but it can be reduced to the characteristically pithy remark by John Searle: “*Brains cause minds*” Intelligence and mental phenomena are products of the operation of the physical system of neurons and their associated cells and support structures.

This doctrine has a number of corollaries. Perhaps the most important is that mental states—such as being in pain, knowing that one is riding a horse, or believing that Vienna is the capital of Austria—are brain states. This does not commit one to very much, other than to avoid speculation about nonphysical processes beyond the ken of science. The next step is to abstract away from specific brain states. One must allow that different brain states can correspond to the same mental state, provided they are of the *same type*. Various authors have various positions on what one means by *type* in this case. Almost everyone believes that if one takes a brain and replaces some of the carbon atoms by a new set of carbon atoms,² the mental state will not be affected. This is a good thing because real brains are continually replacing their atoms through metabolic processes, and yet this in itself does not seem to cause major mental upheavals. **Functionalism**, on the other hand, proposes a much looser definition of “same type,” based on identity of the functional properties of the components of the brain—that is, their input/output specifications. In the neural version of functionalism,³ what matters is the input/output properties of the neurons, not their physical properties. Because the same input/output properties can be realized by many different physical devices, including silicon devices, functionalism naturally leads to the belief that AI systems with the appropriate structure might have real mental states.

Now we begin to explain *how* it is that brains cause minds. Some of the earliest forms of explanation involved what are now called **homunculi**—a Latin term meaning “miniature men.” For example, when it was found that a small image of the world was formed on the retina by the lens, some early philosophers proposed that vision was achieved by a subsystem of the brain—a homunculus—that looked at this image and reported what it saw.⁴ Unfortunately, one then has to explain how it is that the homunculus can see. An **infinite regress** begins when one proposes that the homunculus sees by means of a smaller homunculus inside his head.

Modern proponents of rule-based models of human behavior have been accused of falling into the same infinite regress trap. The objection goes as follows: every use of a logical rule (such as “all men are mortal”) must itself be governed by a rule (such as Modus Ponens). The application of a rule such as Modus Ponens must in turn be governed by another rule, and so on *ad infinitum*. Therefore, intelligent behavior cannot be produced by following rules. Now this argument has a certain superficial plausibility, particularly if we note that the forward-chaining application of inference rules such as Modus Ponens indeed can be viewed as itself an application

² Perhaps even atoms of a different isotope of carbon, as is sometimes done in brain-scanning experiments.

³ There are other versions of functionalism that make the decomposition at different points. In general, there is a wide range of choices in what counts as a component.

⁴ Proponents of this theory were excited by the discovery of the pineal gland, which in many mammals looks superficially like an eye. Closer examination of course revealed that it has no powers of vision.

of Modus Ponens at a higher level. Clearly, however, logical systems do manage to work without infinite regress. This is because at some point, the rule application process *is fixed*; for example, in Prolog, the process of left-to-right, depth-first chaining applies to all inferences. Although this process *could* be implemented by a meta-Prolog interpreter, there is no need because it is not subject to alteration. This means that it can be implemented directly by a physical process, based on the actual operation of the laws of physics, rather than the application of rules representing those laws. The physical implementation of the reasoning system serves as the base case that terminates the regress before it becomes infinite. There is therefore no in-principle philosophical difficulty in seeing how a machine can operate as a reasoning system to achieve intelligent behavior. There is a practical difficulty: the system designer must show both that the rules will lead to the right inferences, and that the reasoning system (the fixed, physical instantiation) correctly implements the metalevel reasoning rules.

INTENTIONAL STATES

So much for intelligent behavior, at least for now. What about mental states? How do we explain their possession by humans, and do machines have them? We will first consider propositional attitudes, which are also known as **intentional states**. These are states, such as believing, knowing, desiring, fearing, and so on, that refer to some aspect of the external world. For example, the belief that Vienna is the capital of Austria is a belief *about* a particular city and its status (or if you prefer, a belief about a proposition which is in turn about a city). The question is, when can one say that a given entity has such a belief?

INTENTIONAL STANCE

One approach is to adopt what Daniel Dennett (1971) has called the **intentional stance** toward the entity. On this view, ascribing beliefs or desires to entities is no more than a calculational device that allows one to predict the entity's behavior. For example, a thermostat can be said to desire that room temperature be maintained in a certain range, and to believe that the room is currently too cold and that switching on the heat increases the room temperature (McCarthy and Hayes, 1969). It is then reasonable to ascribe intentional states when that provides the most succinct explanatory model for the entity's behavior. As mentioned in Chapter 1, this is similar to ascribing pressure to a volume of gas. However, the thermostat's "belief" that the room is too cold is not identical to the corresponding belief held by a person. The thermostat's sensors only allow it to distinguish three states in the world, which we might call *Hot*, *OK*, and *Cold*. It can have no conception of a room, nor of heat for that matter. "The room is too cold" is simply a way of naming the proposition in which the thermostat is said to believe.

The "intentional stance" view has the advantage of being based solely on the entity's behavior and not on any supposed internal structures that might constitute "beliefs." This is also a disadvantage, because any given behavior can be implemented in many different ways. The intentional stance cannot distinguish among the implementations. For some implementations, there are no structures that might even be candidates for representations of the ascribed beliefs or desires. For example, if a chess program is implemented as an enormous lookup table, then it does not seem reasonable to suppose that the program *actually* believes that it is going to capture the opponent's queen; whereas a human with identical chess behavior might well have such a belief. The question of *actual* intentional states is still a real one.

The intentional stance allows us to avoid paradoxes and clashes of intuition (such as being forced to say that thermostats have beliefs). This makes us all feel more comfortable, but it is not necessarily the right scientific approach. Intuitions can change, and paradoxes can be resolved. The hypothesis that the earth revolves around the sun was once called "the Copernican paradox"

precisely because it clashed with the intuitions afforded by the folk cosmology of the time. We may find out that our current folk psychology is so far off base that it is giving us similar kinds of faulty intuitions.

CORRESPONDENCE THEORY

The **correspondence theory** of belief goes somewhat further toward a realistic account. According to this theory, an internal structure in an agent is a reasonable candidate for a representation of a proposition if the following conditions hold:

1. The structure is formed when sensory evidence for the truth of the proposition is obtained.
2. The structure ceases to exist when sensory evidence for the proposition's falsehood is obtained.
3. The structure plays the appropriate causal role in the selection of actions.

Thus, the internal structure acts as a "flag" that correlates with the external proposition.

GROUNDING

The correspondence theory contains the crucial element of **grounding**—the agent's beliefs are grounded in its sensory experience of the world. Grounding is often viewed as essential to the possession of intentional states. For example, before an agent can be said to be dying for a hamburger, it must have some direct experience of hamburgers, or at least of related comestibles. It is not enough that its knowledge base contain the sentence *DyingFor(Me,Hamburger)*. However, if the sentence gets there "in the right way"—that is, through experience of hamburgers and so on—then we say it has the appropriate **causal semantics**. It has meaning relating to hamburgers because of its connection to actual hamburger experiences.

CAUSAL SEMANTICS

There are two views on the sense in which an internal representation has actual meaning. The first view, called **wide content**, has it that the internal representation *intrinsically* refers to some aspect of the outside world; that is, there is some connection between the internal representation and the external world that exists by the nature of the representation. For example, the internal state corresponding to the belief that "This hamburger is delicious" intrinsically refers to the particular hamburger that it is "about." The second view, called **narrow content**, says that no such connection exists. The "hamburgery" aspect of the belief is an intrinsic aspect of the belief *as experienced by the agent*.

WIDE CONTENT**NARROW CONTENT****BRAIN IN A VAT**

We can distinguish between the two views by considering one of the favorite devices of philosophers: the **brain in a vat**. Imagine, if you will, that your brain was removed from your body at birth and placed in a marvellously engineered vat. The vat sustains your brain, allowing it to grow and develop. At the same time, electronic signals are fed to your brain from a computer simulation of an entirely fictitious world, and motor signals from your brain are intercepted and used to modify the simulation as appropriate. The brain in a vat has been wheeled out many times to resolve questions in philosophy. Its role here is to show that wide content is not consistent with physicalism. The brain in the vat can be in an identical state to the brain of someone eating a hamburger; yet in one case, the hamburger exists; in the other, it does not. Even in the former case, then, the belief only refers to the actual hamburger in the eyes of a third party who has independent access to the internals of the brain and to the external world containing the hamburger. The brain by itself does not refer to the hamburger.

QUALIA

The narrow-content view goes beyond the simple correspondence theory. The belief that a hamburger is delicious has a certain intrinsic nature—there is something that it is like to have this belief. Now we get into the realm of **qualia**, or intrinsic experiences (from the Latin word meaning, roughly, "such things"). The correspondence theory can account for the verbal or

discriminatory behaviors associated with the beliefs "the light is red" or "the light is green," for example. But it cannot distinguish between the experiences of red and green—what it is like to see red as opposed to what it is like to see green. It does seem that there is a real question here, but it is not one that seems likely to yield to a behavioral analysis. If the intrinsic experiences arising from exposure to red and green lights were somehow switched, it seems reasonable to suppose that we would still behave the same way at traffic lights, but it also seems reasonable to say that our lives would be in some way different. This final aspect of intentional states—their "felt quality" if you like—is by far the most problematic. It brings up the question of consciousness, which, as we will see, is very ticklish indeed.

26.3 ON THE POSSIBILITY OF ACHIEVING INTELLIGENT BEHAVIOR

One of the most basic philosophical questions for AI is "Can machines think?" We will not attempt to answer this question directly, because it is not clearly defined. To see why, consider the following questions:

- Can machines fly?
- Can machines swim?

Most people would agree that the answer to the first question is yes, airplanes can fly, but the answer to the second is no; boats and submarines do move through the water, but we do not normally call that swimming. However, neither the questions nor the answers have any impact at all on the working lives of aeronautic and naval engineers. The answers have very little to do with the design or capabilities of airplanes and submarines, and much more to do with the way we have chosen to use words. The word "swim" has come to mean "to move along in the water by movements of the limbs or other body parts," whereas the word "fly" has no such limitation on the means of locomotion.

To complicate matters, words can be used metaphorically, so when we say a computer (or an engine, or the economy) is running well, we mean it is operating smoothly, not that it is propelling itself with its legs in an admirable fashion. Similarly, a person who says, "My modem doesn't work because the computer thinks it is a 2400-baud line" is probably using "thinks" metaphorically, and may still maintain that computers do not *literally* think.

The practical possibility of "thinking machines" has been with us only for about 40 years, not long enough for speakers of English to settle on an agreed meaning for the word "think." In the early days of the debate, some philosophers thought that the question of thinking machines could be settled by means of linguistic analysis of the kind hinted at earlier. If we define "think" to mean something like "make decisions or deliberations by means of an organic, natural brain," then we must conclude that computers cannot think. Ultimately, the linguistic community will come to a decision that suits its need to communicate clearly,⁵ but the decision will not tell us much about the capabilities of machines.

⁵ Wittgenstein said that we should "look at the word 'to think' as a tool."

Alan Turing, in his famous paper "Computing Machinery and Intelligence" (Turing, 1950), suggested that instead of asking "Can machines think?" we should instead ask if they can pass a behavioral test (which has come to be called the Turing Test) for intelligence. He conjectured that by the year 2000, a computer with a storage of 10^9 units could be programmed well enough to have a conversation with an interrogator for 5 minutes and have a 30% chance of fooling the interrogator into thinking it was human. Although we would certainly not claim that anything like general, human-level intelligence will be achieved by that time, his conjecture may not be that far off the truth. Turing also examined a wide variety of possible objections to the possibility of intelligent machines, including virtually all of those that have been raised in the 44 years since his paper appeared.

Some of the objections can be overcome quite easily. For example, Lady Ada Lovelace, commenting on Babbage's Analytical Engine, says, "It has no pretensions to *originate* anything. It can do *whatever we know how to order it* to perform." This objection, that computers can only do what they are told to do and are therefore not capable of creativity, is commonly encountered even today. It is refuted simply by noting that one of the things we can tell them to do is to learn from their experience. For example, Samuel's checker-playing program performed very poorly with its original programming. However, it was able to learn, over the course of a few days of self-play, to play checkers far better than Samuel himself (see Chapter 20). One can try to preserve Lady Lovelace's objection by maintaining that the program's ability to learn originated in Samuel, and so too did its checker-playing ability. But then one would also be led to say that Samuel's creativity originated in his parents, and theirs originated in their parents, and so on.

The "argument from disability" takes the form of a claim, usually unsupported, to the effect that "a machine can never do X." As examples of X, Turing lists the following:

Be kind, resourceful, beautiful, friendly, have initiative, have a sense of humor, tell right from wrong, make mistakes, fall in love, enjoy strawberries and cream, make someone fall in love with it, learn from experience, use words properly, be the subject of its own thought, have as much diversity of behavior as man, do something really new.

Although some of these abilities concern the consciousness of machines, which we discuss at length in what follows, many concern behavioral properties (see Exercise 26.1). Turing suggests that scepticism of this nature arises from experience of machines as devices for carrying out repetitive tasks requiring little sensory and no reasoning ability. He points to the fact that in the late 1940s, the general population found it difficult to believe that machines could find numerical solutions of equations or predict ballistic trajectories. Even today, however, many technically literate people do not believe that machines can learn.

The supposed inability to make mistakes presents an interesting problem when considering the Turing Test. Certainly, instantaneous and correct answers to long division problems would be a giveaway, and some attempt to simulate human fallibility would be required.⁶ But this is not a mistake in the normal sense, because the program is doing exactly what its designer intended. Something more akin to human mistakes will arise when intractable problems are involved. For example, given only a small amount of time to find a chess move, the computer must essentially guess that its move is correct. Similarly, a program that is trying to induce hypotheses from

⁶ In recent Turing Test competitions, the winning programs are those that type their answers back slowly and irregularly, with occasional corrections and spelling mistakes. A Markov model of typing speed and accuracy is sufficient.

a small amount of data is bound to make mistakes when using such hypotheses for prediction. When unavoidably irrational behavior on the part of the computer matches corresponding failings of humans, this provides evidence that similar mechanisms are in operation. Rational behavior, on the other hand, provides much weaker constraints on mechanisms.

What Turing calls the mathematical objection concerns the proven inability of computers to answer certain questions. We discuss this in-principle barrier to intelligence in Section 26.3. In-practice objections center on the so-called "argument from informality," which claims that intelligent behavior cannot be captured by formal rules. We discuss this category of objections in Section 26.3. The final, and most interesting, objection claims that even if computers behave as intelligently as humans, they still will not *be* intelligent. Although AI cannot do much more than make machines behave intelligently, we still have some fun discussing the issue in Section 26.4.

The mathematical objection

HALTING PROBLEM

It is well-known, partly through the work of Turing himself (Turing, 1936) as well as that of Gödel (1931), that certain questions cannot be answered correctly by any formal system. An example is the **halting problem**: will the execution of a program P eventually halt, or will it run forever? Turing proved that for any algorithm H that purports to solve halting problems there will always be a program P , such that H will not be able to answer the halting problem correctly. Turing therefore acknowledges that for any particular machine that is being subjected to the Turing Test, the interrogator can formulate a halting problem that the machine cannot answer correctly.

Philosophers such as Lucas (1961) have claimed that this limitation makes machines inferior to humans, who can always "step outside" the limiting logic to see whether the problem in question is true or not. Lucas bases his argument not on the halting problem but on Gödel's incompleteness theorem (see Chapter 9). Briefly, for any non-trivial formal system F (a formal language, and a set of axioms and inference rules), it is possible to construct a so-called "Gödel sentence" $G(F)$ with the following properties:

- $G(F)$ is a sentence of F , but cannot be proved within F .
- If F is consistent, then $G(F)$ is true.

Lucas claims that because a computer is a formal system, in a sense that can be made precise, then there are sentences whose truth it cannot establish—namely, its own Gödel sentences. A human, on the other hand, can establish the truth of these sentences by applying Gödel's theorem to the formal system that describes the computer and its program.

Lucas's point is essentially the same as the potential objection raised by Turing himself, and can be refuted in the same way. We admit that there is a strong intuition that human mathematicians can switch from one formalism to another until they find one that solves the problem they are faced with. But there is no reason why we could not have the same intuition about a computer that was programmed to try thousands of different formalisms (and to invent new formalisms) in an attempt to solve the halting or Gödel problem. If we accept that the brain is a deterministic physical device operating according to normal physical laws, then it is just as much a formal system as the computer (although a harder one to analyze), and thus has the same limitations as computers. If you believe there are nondeterministic aspects of the world

that keep the brain from being a formal system, then we can build a computer that incorporates analog devices to achieve the same nondeterminism (for example, using a Geiger counter to seed a random number generator).

In practical terms, the limitations of formal systems are not going to be of much help to the interrogator in the Turing Test anyway. First of all, descriptions of the halting problems for either humans or sufficiently intelligent computers are going to be far too large to talk about. Even if they were small enough, the interrogator does not know the details of the computer it is talking to, and thus has no way of posing the right halting problem. And even if the interrogator made a lucky guess, the computer could just wait a few minutes (or weeks) and then reply "It looks like it doesn't halt, but I'm not sure—it's pretty complicated." Presumably that would be a good imitation of a typical human reply.

Another way of stating the refutation argument, in a form appropriate to Lucas's version, is that a human being cannot show the consistency of a formal system that describes the operation of his or her brain or of a large computer, and cannot therefore establish the truth of the corresponding Gödel sentence. Therefore, humans have the same limitations that formal systems have.

In a more recent revival of the mathematical objection, the mathematician Roger Penrose has written a reasonably controversial book on AI, provocatively entitled *The Emperor's New Mind*,⁷ which purports to overcome these objections to Lucas. Penrose argues that, at least when we consider the mental faculties that mathematicians use to generate new mathematical propositions and their proofs, the claim that *F* is complex cannot hold up. This is because when a new result is found, it is usually a simple matter for one mathematician to communicate it to another, and to provide convincing evidence by means of a series of simple steps. He begins by assuming that this universal facility for mathematics is algorithmic, and tries to show a contradiction:

Now this putative "universal" system, or algorithm, cannot ever be known as the one that we mathematicians use to decide truth. For if it were, we would construct its Gödel proposition and *know that to be a mathematical truth also*. Thus, we are driven to the conclusion that the algorithm that mathematicians use to decide mathematical truth is so complicated or obscure that its very validity can never be known to us. (Penrose, 1990, p. 654).

And because mathematical truths are in fact simple to see, at least step by step, mathematical "insight" cannot be algorithmic. We can paraphrase the argument as follows:

1. Mathematics is simple, and includes the "Gödelisation" process, which is also simple.
2. Hence the "validity of mathematics" is easily seen.
3. Thus, if mathematics were a formal system, its Gödel sentence would easily be seen to be true by mathematicians.
4. Hence mathematical insight cannot be algorithmic.

In an entertaining series of replies appearing in the journal *Behavioral and Brain Sciences*, a number of mathematicians including George Boolos (1990) and Martin Davis (1990) point out an obvious flaw in the first two steps. Mathematics has a long history of inconsistencies, the most famous of which is Frege's *Basic Laws of Arithmetic*, shown inconsistent by Russell's paradox

⁷ Penrose (1990) strenuously denies that the obvious analogy to the tale of the Emperor's New Clothes was intended in any way to suggest that AI is not all it claims to be.

in 1902. Almost every major figure in the history of logic has at one time or another published an inconsistent set of axioms. Even today, there is some doubt as to whether the principal axiomatization of set theory known as ZFC (the Zermelo-Fraenkel axioms plus the Axiom of Choice) is consistent, even though it is used widely as the basis for most of mathematics. Penrose replies as follows:

I am not asserting that, in any particular case of a formal system F , we need necessarily be able to "see" that $G(F)$ is true, but I am asserting that the "Godelian insight" that enables one to pass from F to $G(F)$ is just as good a mathematical procedure for deriving new truths from old as are any other procedures in mathematics. This insight is not contained within the rules of F itself, however. Thus, F does not encapsulate all the insights available to mathematicians. The insights that *are* available to mathematicians are not formalizable. (Penrose, 1990, p. 694).

Penrose does not say why he thinks the "Godelian insight" is not formalizable, and it appears that in fact it has been formalized. In his Ph.D. thesis, Natarajan Shankar (1986) used the Boyer-Moore theorem prover BMTP to derive Gödel's theorem from a set of basic axioms, in much the same way that Gödel himself did.⁸ The thesis was intended mainly to demonstrate the power of automatic theorem provers, in particular the capability of BMTP to develop and apply lemmata. But it shows, as Robert Wilensky (1990) has pointed out, that one needs to be careful to distinguish between the formal system that is doing the proving—in this case, a computer programmed with the BMTP—and the formal system within which the proof is carried out, namely, an axiomatization of arithmetic and of sentences in that axiomatization. Just like a mathematician, a good automated theorem prover can apply Gödelisation to the particular formal system that it is working on; but it can no more establish the consistency of its own program and hardware than a mathematician can establish the consistency of his or her own brain. We therefore seem to be back where we started, with the refutation of the "mathematical objection" that Turing himself raised.

Penrose naturally maintains that in some way, mathematicians' use of insight remains nonalgorithmic. Perhaps the most interesting aspect of his book is the conclusion he draws from this. After providing a valuable discussion of the relevance of physics to the brain, he deduces that nothing in our current physical understanding of its operation would suggest that it has nonalgorithmic aspects—that is, the simulation of its operation by a computer, as described earlier, would in principle be possible according to modern physics. Rather than accepting the conclusion that perhaps the brain is, after all, algorithmic in this sense, he prefers to believe that modern physics must be wrong. In particular, the brain must use physical principles not yet discovered, but probably relating to the interaction between quantum theory and gravity, that must also be nonalgorithmic in character.

The argument from informality

One of the most influential and persistent criticisms of AI as an enterprise was raised by Turing as the "argument from informality of behavior." Essentially, this is the claim that human behavior

⁸ Ammon's SHUNYATA system (1993) even appears to have developed by itself the diagonalization technique used by Gödel and developed originally by Cantor.

is far too complex to be captured by any simple set of rules; and because computers can do no more than follow a set of rules, they cannot generate behavior as intelligent as that of humans.

The principal proponent of this view has been the philosopher Hubert Dreyfus, who has produced a series of powerful critiques of artificial intelligence: *What Computers Can't Do* (1972; 1979), *What Computers Still Can't Do* (1992), and, with his brother Stuart, *Mind Over Machine* (1986). Terry Winograd, whose PhD thesis (Winograd, 1972) on natural language understanding is criticized fiercely by Dreyfus (1979), has also expressed views similar to those of Dreyfus in his recent work (Winograd and Flores, 1986).

The position they criticize came to be called "Good Old-Fashioned AI," or GOFAI, a term coined by Haugeland (1985). GOFAI is supposed to claim that all intelligent behavior can be captured by a system that reasons logically from a set of facts and rules describing the domain. It therefore corresponds to the simplest logical agent described in Chapter 6. Dreyfus claims that

when Minsky or Turing claims that man can be understood as a Turing machine, they must mean that a digital computer can reproduce human behavior . . . by *processing data representing facts about the world using logical operations* that can be reduced to matching, classifying and Boolean operations. (Dreyfus, 1972, p. 192)

It is important to point out, before continuing with the argument, that AI and GOFAI are not the same thing. In fact, it is not clear whether any substantial section of the field has ever adhered to GOFAI in this extreme form, although the phrase "facts and rules" did appear in Turing's two-sentence speculation concerning how systems might be programmed to pass the Turing Test.⁹ One only has to read the preceding chapters to see that the scope of AI includes far more than logical inference. However, we shall see that Dreyfus' criticisms of GOFAI make for interesting reading and raise issues that are of interest for all of AI.

Whether or not anyone adheres to GOFAI, it has indeed been a working hypothesis of most AI research that the knowledge-based approach to intelligent system design has a major role to play. Hubert Dreyfus has argued that this presumption is based in a long tradition of **rationalism** going back to Plato (see Section 1.2). He cites Leibniz as providing a clear statement of the goals of the modern "expert systems" industry:

The most important observations and turns of skill in all sorts of trades and professions are as yet unwritten . . . We can also write up this practice, since it is at bottom just another theory.

Furthermore, he claims that the presumption is wrong; that competence can be achieved without explicit reasoning or rule following. The Dreyfus critique therefore is not addressed against computers *per se*, but against one particular way of programming them. It is reasonable to suppose, however, that a book called *What First-Order Logical Rule-Based Systems Can't Do* might have had less impact.

Dreyfus's first target is the supposition that early successes of GOFAI justify optimism that the methodology will succeed in scaling up to human-level intelligence. Many AI programs in the 1960s and early 1970s operated in **microworlds**—small, circumscribed domains such as the Blocks World. In microworlds, the totality of the situation can be captured by a small number of facts. At the time, many AI researchers were well aware that success in microworlds could be achieved without facing up to a major challenge in the real world: the vast amount of

⁹ Significantly, Turing also said that he expected AI to be achieved by a learning machine, not a preprogrammed store.

potentially relevant information that could be brought to bear on any given problem. As we saw in Chapter 22, disambiguation of natural language seems to require access to this background knowledge. Dreyfus gives as an example the text fragment

Mary saw a dog in the window. She wanted it.

This example was used originally by Lenat (Lenat and Feigenbaum, 1991) to illustrate the commonsense knowledge needed to disambiguate the "it" in the second sentence. Presumably, "it" refers to the dog rather than the window. If the second sentence had been "She smashed it" or "She pressed her nose up against it," the interpretation would be different. To generate these different interpretations in the different contexts, an AI system would need a fair amount of knowledge about dogs, windows, and so on. The project of finding, encoding, and using such knowledge has been discussed since the early days of AI, and Lenat's CYC project (Lenat and Guha, 1990) is probably the most well-publicized undertaking in this area.

The position that Dreyfus adopts, however, is that this general commonsense knowledge is not explicitly represented or manipulated in human performance. It constitutes the "holistic context" or "Background" within which humans operate. He gives the example of appropriate social behavior in giving and receiving gifts: "Normally one simply responds in the appropriate circumstances by giving an appropriate gift." One apparently has "a direct sense of how things are done and what to expect." The same claim is made in the context of chess playing: "A mere chess master might need to figure out what to do, but a grandmaster just sees the board as demanding a certain move." Apparently, the "right response just pops into his or her head."

Dreyfus seems at first to be making a claim that might appear somewhat irrelevant to the weak AI program: that if humans are sometimes not conscious of their reasoning processes, then on those occasions no reasoning is occurring. The obvious AI reply would be to distinguish between **phenomenology**—how things, including our own reasoning, appear to our conscious experience—and causation. AI is required to find a causal explanation of intelligence. One might well claim that knowledge of chess—the legal moves and so on—is being used, but perhaps not at a conscious level. As yet, AI does not claim to have a theory that can distinguish between conscious and unconscious deliberations, so the phenomenological aspects of decision-making are unlikely to falsify any particular approach to AI.

Another approach might be to propose that the grandmaster's supposed ability to see the right move immediately derives from a partial situation-action mapping used by a reflex agent with internal state. The mapping might be learned directly (see Chapter 20) or perhaps compiled from more explicit knowledge (see Chapter 21). And as discussed in Chapter 2, situation-action mappings have significant advantages in terms of efficiency. On the other hand, even a grandmaster sometimes needs to use his or her knowledge of the legal moves to deal with unfamiliar situations, to find a way out of a trap, or to ensure that a mating attack is unavoidable.

Dreyfus's position is actually more subtle than a simple appeal to magical intuition. *Mind Over Matter* (Dreyfus and Dreyfus, 1986) proposes a five-stage process of acquiring expertise, beginning with rule-based processing (of the sort proposed in AI) and ending with the ability to select correct responses instantaneously:

We have seen that computers do indeed reason things out rather like inexperienced persons, but only with greater human experience comes know-how, a far superior, holistic, intuitive way of approaching problems that cannot be imitated by rule-following computers.

PHENOMENOLOGY

Dreyfus's first proposal for how this "know-how" operates is that humans solve problems by analogy, using a vast "case library" from which somehow the most relevant precedents are extracted. He proposed "some sort of holographic memory" as a potential mechanism. He later suggests neural networks as a possible implementation for the final "know-how" phase.

Now he reaches what is perhaps the inevitable destination of the weak-AI critic: he ends up effectively as an AI researcher, because he cannot avoid the question "If AI mechanisms cannot work, what mechanism do you propose instead for human performance?" His answer, that humans use some sort of learning method, is not new to AI. Since the very early experiments of Samuel and Friedberg, researchers have proposed using machine learning as a method of achieving higher levels of performance and avoiding the difficulties of hand coding. The question is, what is the target representation for the learning process? Dreyfus chooses neural networks because they can achieve intelligence, to some degree, *without explicit representation of symbolic knowledge*.¹⁰ He claims, however, that there is no reason to suppose that real intelligence can be achieved without a brain-sized network; nor would we understand the results of training such a network.

Dreyfus's natural pessimism also leads him to make two useful observations on the difficulty of a naive scheme to construct intelligence by training a large network with appropriate examples:

1. Good generalization from examples cannot be achieved without a good deal of background knowledge; as yet, no one has any idea how to incorporate background knowledge into the neural network learning process.
2. Neural network learning is a form of **supervised** learning (see Chapter 18), requiring the prior identification of relevant inputs and correct outputs. Therefore, it cannot operate autonomously without the help of a human trainer.

The first objection was also raised in Chapter 18, and in Chapter 21, we saw several ways in which background knowledge indeed can improve a system's ability to generalize. Those techniques, however, relied on the availability of knowledge in explicit form, something that Dreyfus denies strenuously. In our view, this is a good reason for a serious redesign of current models of neural processing so that they *can* take advantage of previously learned knowledge. There has been some progress in this direction. The second objection leads directly to the need for **reinforcement learning** (Chapter 20), in which the learning system receives occasional positive or negative rewards, rather than being told the correct action in every instance. Given enough experience, a reinforcement learning agent can induce a utility function on situations, or alternatively a mapping from situation-action pairs to expected values. For example, by winning and losing games a chess-playing agent can gradually learn which sorts of positions are promising or dangerous. Reinforcement learning is currently very popular in neural network systems.

Dreyfus correctly points out that the major problem associated with reinforcement learning is how to generalize from particular situations to more general classes of situations—the general problem of inductive learning. One can take heart from the observation that reinforcement learning reduces to ordinary inductive learning, for which we already have some well-developed techniques. There are, of course, problems yet to be solved with inductive learning, including problem 1, mentioned earlier, concerning how to use background knowledge to improve learning. Dreyfus also brings up the problem of learning in the context of a large number of potentially

¹⁰ In fact, many neural network researchers are proud that their networks seem to learn distinct, higher-level "features" of the input space and to combine them in approximately logical ways.

relevant features. One possible solution is to stick to a small finite set of features, and add new ones when needed. But according to him, "There is no known way of adding new features should the current set prove inadequate to account for the learned facts." As we saw in Chapter 21, there are well-principled ways to generate new features for inductive learning.

Another difficult problem in reinforcement learning arises when the available perceptual inputs do not completely characterize the situation. In such cases, the agent must develop additional internal state variables, in terms of which output mappings can be learned. Dreyfus claims that "Since no one knows how to incorporate internal states appropriately, a breakthrough will be necessary." Again, this is a tricky problem, but one on which some progress has been made (see Chapter 20).

The final problem to which Dreyfus refers in *What Computers Still Can't Do* is that of controlling the acquisition of sensory data. He notes that the brain is able to direct its sensors to seek relevant information and to process it to extract aspects relevant to the current situation. He says (page xliv), "Currently, no details of this mechanism are understood or even hypothesized in a way that could guide AI research." Yet the field of active vision, underpinned by the theory of information value (Chapter 16), is concerned with exactly this problem, and already robots incorporate the theoretical results obtained.

Dreyfus seems willing to grant that success in overcoming these obstacles would constitute the kind of real progress in AI that he believes to be impossible. In our view, the fact that AI has managed to reduce the problem of producing human-level intelligence to a set of relatively well-defined technical problems seems to be progress in itself. Furthermore, these are problems that are clearly soluble in principle, and for which partial solutions are already emerging. In summary, we have seen that the life of a weak-AI critic is not an easy one. Claims that "X is impossible for computers" (e.g., X might be beating a chess master) tend to be overtaken by actual events. They also open the critic to the requirement of suggesting a mechanism by which humans do X; this forces them, essentially, to become AI researchers. On the other hand, perceptive criticism from outside the field can be very useful. Many of the issues Dreyfus has focused on—background commonsense knowledge, the qualification problem, uncertainty, learning, compiled forms of decision making, the importance of considering situated agents rather than disembodied inference engines—are now widely accepted as important aspects of intelligent agent design.

26.4 INTENTIONALITY AND CONSCIOUSNESS

Many critics have objected to the Turing Test, stating that it is not enough to see how a machine acts; we also need to know what internal "mental" states it has. This is a valid and useful criticism; certainly in trying to understand any computer program or mechanical device it is helpful to know about its internal workings as well as its external behavior. Again, the objection was foreseen by Turing. He cites a speech by a Professor Jefferson:

Not until a machine could write a sonnet or compose a concerto because of thoughts and emotions felt, and not by the chance fall of symbols, could we agree that machine equals brain—that is, not only write it but know that it had written it.



Jefferson's key point is consciousness: *the machine has to be aware of its own mental state and actions.* Others focus on intentionality, that is, the "aboutness" (or lack thereof) of the machine's purported beliefs, desires, intentions, and so on.

Turing's response to the objection is interesting. He could have presented reasons why machines can in fact be conscious. But instead he maintains that the question is just as ill-defined as asking "can machines think," and in any case, why should we insist on a higher standard for machines than we do for humans? After all, in ordinary life we never have *any* evidence about the internal mental states of other humans, so we cannot know that anyone else is conscious. Nevertheless, "instead of arguing continually over this point, it is usual to have the polite convention that everyone thinks," as Turing puts it.

Turing argues that Jefferson would be willing to extend the polite convention to machines if only he had experience with ones that act intelligently, as in the following dialog, which has become such a part of AI's oral tradition that we simply have to include it:

HUMAN: In the first line of your sonnet which reads 'shall I compare thee to a summer's day,' would not a 'spring day' do as well or better?

MACHINE: It wouldn't scan.

HUMAN: How about 'a winter's day'. That would scan all right.

MACHINE: Yes, but nobody wants to be compared to a winter's day.

HUMAN: Would you say Mr. Pickwick reminded you of Christmas?

MACHINE: In a way.

HUMAN: Yet Christmas is a winter's day, and I do not think Mr. Pickwick would mind the comparison.

MACHINE: I don't think you're serious. By a winter's day one means a typical winter's day, rather than a special one like Christmas.

Jefferson's objection is still an important one, because it points out the difficulty of establishing any objective test for consciousness, where by "objective" we mean a test that can be carried out with consistent results by any sufficiently competent third party. Turing also concedes that the question of consciousness is not easily dismissed: "I do not wish to give the impression that I think there is no mystery about consciousness ... But I do not think these mysteries necessarily need to be solved before we can answer the question with which we are concerned in this paper," namely, "Can machines think?"

Although many, including Jefferson, have claimed that thinking necessarily involves consciousness, the issue is most commonly associated with the work of the philosopher John Searle. We will now discuss two thought experiments that, Searle claims, refute the thesis of strong AI.

The Chinese Room

We begin with the Chinese Room argument (Searle, 1980). The idea is to describe a hypothetical system that is clearly running a program and passes the Turing Test, but that equally clearly (according to Searle) does not understand anything of its inputs and outputs. The conclusion will be that running the appropriate program (i.e., having the right outputs) is not a *sufficient* condition for being a mind.

The system consists of a human, who understands only English, equipped with a rule book, written in English, and various stacks of paper, some blank, some with indecipherable

inscriptions. (The human therefore plays the role of the CPU, the rule book is the program, and the stacks of paper are the storage device.) The system is inside a room with a small opening to the outside. Through the opening appear slips of paper with indecipherable symbols. The human finds matching symbols in the rule book, and follows the instructions. The instructions may include writing symbols on new slips of paper, finding symbols in the stacks, rearranging the stacks, and so on. Eventually, the instructions will cause one or more symbols to be transcribed onto a piece of paper that is handed through the opening to the outside world.

So far, so good. But from the outside, we see a system that is taking input in the form of Chinese sentences and generating answers in Chinese that are as obviously "intelligent" as those in the conversation imagined by Turing.¹¹ Searle then argues as follows: the person in the room does not understand Chinese (given); the rule book and the stacks of paper, being just pieces of paper, do not understand Chinese; therefore there is no understanding of Chinese going on. *Hence, According to Searle, running the right program does not necessarily generate understanding.*

Like Turing, Searle considered and attempted to rebuff a number of replies to his argument. First, we will consider the so-called Robot Reply (due to Jerry Fodor (1980) among others), which turns out to be a red herring, although an interesting one. The Robot Reply is that although the symbols manipulated by the Chinese Room may not have real meaning to the room itself (e.g., nothing in the room has any experience of acupuncture, with respect to which the symbol for it might have any meaning), a fully equipped robot would not be subject to the same limitations. Its internal symbols would have meaning to it by virtue of its direct experience of the world. Searle's reply is to put the Chinese Room inside the robot's "head": the sensors are redesigned to generate Chinese symbols instead of streams of bits, and the effectors redesigned to accept Chinese symbols as control inputs. Then we are back where we started. The Robot Reply is a red herring because the causal semantics of the symbols is not the real issue. Even the original Chinese Room needs some causal semantics, in order to be able to answer questions such as "How many questions have I asked so far?" Conversely, the outputs of human sensors, for example, along the optic nerve or the auditory nerve, might as well be in Chinese (see the earlier discussion of wide and narrow content). Not even Searle would argue that connecting artificial sensors to these nerves would remove consciousness from the brain involved.¹²

Several commentators, including John McCarthy and Robert Wilensky, propose what Searle calls the Systems Reply. This gets to the point. The objection is that although one can ask if the human in the room understands Chinese, this is analogous to asking if the CPU can take cube roots. In both cases, the answer is no, and in both cases, according to the Systems Reply, the entire system *does* have the capacity in question. Certainly, if one asks the Chinese Room whether it understands Chinese, the answer would be affirmative (in fluent Chinese). Searle's response is to reiterate the point that the understanding is not in the human, and cannot be in the paper, so there cannot be any understanding. He further suggests that one could imagine the human memorizing the rule book and the contents of all the stacks of paper, so that there would

¹¹ The fact that the stacks of paper might well be larger than the entire planet, and the generation of answers would take millions of years, has no bearing on the *logical* structure of the argument. One aim of philosophical training is to develop a finely honed sense of which objections are germane and which are not.

¹² This is a good thing, because artificial inner ears are at the prototype stage (Watson, 1991), and artificial retinas, with associated image processing, are rapidly becoming feasible (Campbell *et al.*, 1991).

be nothing to have understanding *except* the human; and again, when one asks the human (in English), the reply will be in the negative.

Now we are down to the real issues. The shift from paper to memorization is a red herring, because both forms are simply physical instantiations of a running program. The real claim made by Searle has the following form:

1. Certain kinds of objects are incapable of conscious understanding (of Chinese).
2. The human, paper, and rule book are objects of this kind.
3. If each of a set of objects is incapable of conscious understanding, then any system constructed from the objects is incapable of conscious understanding
4. Therefore there is no conscious understanding in the Chinese room.

While the first two steps are on firm ground,¹³ the third is not. Searle just assumes it is true without giving any support for it. But notice that if you do believe it, and if you believe that humans are composed of molecules, then either you must believe that humans are incapable of conscious understanding, or you must believe that individual molecules are capable.

It is important to see that the rebuttal of Searle's argument lies in rejecting the third step, and not in making any claims about the room. You can believe that the room is not conscious (or you can be undecided about the room's consciousness) and still legitimately reject Searle's argument as invalid.

Searle's (1992) more recent position, described in his book *The Rediscovery of the Mind*, is that consciousness is an **emergent property** of appropriately arranged systems of neurons in the same way that solidity is an emergent property of appropriately arranged collections of molecules, none of which are solid by themselves.

Now most supporters of strong AI would also say that consciousness is an emergent property of systems of neurons (or electronic components, or whatever). The question is, which properties of neurons are *essential* to consciousness, and which are merely incidental? In a solid, what counts are the forces that molecules exert on each other, and the way in which those forces change with distance. The solid would still be solid if we replaced each molecule with a tiny computer connected to electromagnetic force field generators. As yet, we do not know which properties of neurons are important—the functional properties associated with information processing or the intrinsic properties of the biological molecules. The Chinese Room argument therefore can be reduced to the empirical claim that *the only physical medium that can support consciousness is the neural medium*. The only empirical evidence for this empirical claim is that other media do not resemble neurons in their intrinsic physical properties. Searle (1992) admits that it is *possible* that other media, including silicon, might support consciousness, but he would claim that in such cases, the system would be conscious *by virtue of the physical properties of the medium* and not by virtue of the program it was running.

To reiterate, the aim of the Chinese Room argument is to refute strong AI—the claim that running the right sort of program necessarily generates consciousness. It does this by exhibiting an apparently intelligent system running the right sort of program that is, according to Searle, *demonstrably unconscious*. He tried to demonstrate this with the argument that unconscious parts

¹³ Searle never explicitly says what kinds of objects are incapable of consciousness. Books and papers for sure, but he wants us to generalize this to computers but not brains without saying exactly what the generalization is.



cannot lead to a conscious whole, an argument that we have tried to show is invalid. Having failed to *prove* that the room is unconscious, Searle then appeals to intuition: just look at the room; what's there to be conscious? While this approach gains some supporters, intuitions can be misleading. It is by no means intuitive that a hunk of brain can support consciousness while an equally large hunk of liver cannot. Furthermore, when Searle admits that materials other than neurons could in principle support consciousness, he weakens his argument even further, for two reasons: first, one has only Searle's intuitions (or one's own) to say that the Chinese room is not conscious, and second, even if we decide the room is not conscious, that tells us nothing about whether a program running on some other physical media might be conscious.

Searle describes his position as "biological naturalism." The physical nature of the system is important, and not its computational description. He even goes so far as to allow the logical possibility that the brain is actually implementing an AI program of the traditional sort. But even if that program turned out to be an exact copy of some existing AI program, moving it to a different machine would destroy consciousness. The distinction between the intrinsic properties (those inherent in its specific physical makeup) and functional properties (the input/output specification) of a neuron is thus crucial.

One way to get at the distinction between intrinsic and functional properties is to look at other artifacts. In 1848, artificial urea was synthesized for the first time, by Wöhler. This was important because it proved that organic and inorganic chemistry could be united, a question that had been hotly debated. Once the synthesis was accomplished, chemists agreed that artificial urea *was* urea, because it had all the right physical properties. Similarly, artificial sweeteners are undeniably sweeteners, and artificial insemination (the other AI) is undeniably insemination. On the other hand, artificial flowers are not flowers, and Daniel Dennett points out that artificial Chateau Latour wine would not be Chateau Latour wine, even if it was chemically indistinguishable, simply because it was not made in the right place in the right way. Similarly, an artificial Picasso painting is not a Picasso painting, no matter what it looks like.

Searle is interested in the notion of *simulations* as well as in artifacts. He claims that AI programs can at best be simulations of intelligence, and such simulations imply no intrinsic properties. The following quote is representative:

No one supposes that a computer simulation of a storm will leave us all wet . . . Why on earth would anyone in his right mind suppose a computer simulation of mental processes actually had mental processes? (Searle, 1980, pp. 37-38)

While it is easy to agree that computer simulations of storms do not make us wet, it is not clear how to carry this analogy over to computer simulations of mental processes. After all, a Hollywood simulation of a storm using sprinklers and wind machines *does* make the actors wet. A computer simulation of multiplication *does* result in a product—in fact a computer simulation of multiplication *is* multiplication. Searle achieves his rhetorical effect by choosing examples carefully. It would not have been as convincing to say "Why on earth would anyone in his right mind suppose that a computer simulation of a video game actually *is* a game?", but it would have the same logical force as his original quote.

To help decide whether intelligence is more like Chateau Latour and Picasso or more like urea and multiplication, we turn to another thought experiment.

The Brain Prosthesis Experiment

The Brain Prosthesis Experiment was touched on by Searle (1980), but is most commonly associated with the work of Hans Moravec (1988). It goes like this. Suppose we have developed neurophysiology to the point where the input/output behavior and connectivity of all the neurons in the brain are perfectly understood. Furthermore, suppose that we can build microscopic electronic devices that mimic this behavior and can be smoothly interfaced to neural tissue. Lastly, suppose that some miraculous surgical technique can replace individual neurons with the corresponding electronic devices without interrupting the operation of the brain as a whole. The experiment consists of gradually replacing all the neurons with electronic devices, and then reversing the process to return the subject to his or her normal biological state.

We are concerned with both the external behavior and the internal experience of the subject, during and after the operation. By the definition of the experiment, the subject's external behavior must remain unchanged compared to what would be observed if the operation were not carried out.¹⁴ Now although the presence or absence of consciousness cannot be easily ascertained by a third party, the subject of the experiment ought at least to be able to record any changes in his or her own conscious experience. Apparently, there is a direct clash of intuitions as to what would happen. Moravec, a robotics researcher, is convinced his consciousness would remain unaffected. He adopts the **functionalist** viewpoint, according to which the input/output behavior of neurons is their only significant property. Searle, on the other hand, is equally convinced his consciousness would vanish:

You find, to your total amazement, that you are indeed losing control of your external behavior.

You find, for example, that when doctors test your vision, you hear them say "We are holding up a red object in front of you; please tell us what you see." You want to cry out "I can't see anything. I'm going totally blind." But you hear your voice saying in a way that is completely out of your control, "I see a red object in front of me." . . . [Y]our conscious experience slowly shrinks to nothing, while your externally observable behavior remains the same. (Searle, 1992)

But one can do more than argue from intuition. First, note that in order for the external behavior to remain the same while the subject gradually becomes unconscious, it must be the case that the subject's volition is removed instantaneously and totally, otherwise the shrinking of awareness would be reflected in external behavior—"Help, I'm shrinking!" or words to that effect. This instantaneous removal of volition as a result of gradual neuron-at-a-time replacement seems an unlikely claim to have to make.

Second, consider what happens if we do ask the subject questions concerning his or her conscious experience during the period when no real neurons remain. By the conditions of the experiment, we will get responses such as "I feel fine. I must say I'm a bit surprised because I believed the Chinese Room argument." Or we might poke the subject with a pointed stick, and observe the response, "Ouch, that hurt." Now, in the normal course of affairs, the sceptic can dismiss such outputs from AI programs as mere contrivances. Certainly, it is easy enough to use a rule such as "If sensor 12 reads 'High' then print 'Ouch.'" But the point here is that because we have replicated the functional properties of a normal human brain, we assume that the electronic brain contains no such contrivances. Then we must have an explanation of the manifestations of

¹⁴ One can imagine using an identical "control" subject who is given a placebo operation, so that the two behaviors can be compared.

"consciousness" produced by the electronic brain that appeals only to the functional properties of the neurons. *And this explanation must also apply to the real brain, which has the same functional properties.* There are, it seems, only two possible conclusions:

1. The causal mechanisms involved in consciousness that generate these kinds of outputs in normal brains are still operating in the electronic version, which is therefore "conscious."
2. The conscious mental events in the normal brain have no causal connection to the subject's behavior, and are missing from the electronic brain.

EPIPHENOMENAL

Although we cannot rule out the second possibility, it reduces consciousness to what philosophers call an **epiphenomenal** role—something that happens but casts no shadow, as it were, on the observable world. Furthermore, if consciousness is indeed epiphenomenal, then the brain must contain a second, unconscious mechanism that is responsible for the "Ouch."

Third, consider the situation after the operation has been reversed and the subject has a normal brain. Once again, the subject's external behavior must be as if the operation had not occurred. In particular, we should be able to ask, "What was it like during the operation? Do you remember the pointed stick?" The subject must have accurate memories of the actual nature of his or her conscious experiences, including the qualia, despite the fact that according to Searle there were no such experiences.

Searle might reply that we have not specified the experimental conditions properly. If the real neurons are, say, put into suspended animation between the time they are extracted and the time they are replaced in the brain, then of course they will not "remember" the experiences during the operation. To deal with this, we simply need to make sure that the neurons' state is updated, by some means, to reflect the internal state of the artificial neurons they are replacing. If the supposed "nonfunctional" aspects of the real neurons then result in functionally different behavior from that observed with artificial neurons still in place, then we have a simple *reductio ad absurdum*, because that would mean that the artificial neurons are not functionally equivalent to the real neurons. (Exercise 26.4 addresses a rebuttal to this argument.)

Patricia Churchland (1986) points out that the functionalist arguments that operate at the level of the neuron can also operate at the level of any larger functional unit—a clump of neurons, a mental module, a lobe, a hemisphere, or the whole brain. That means that if you accept that the brain prosthesis experiment shows that the replacement brain is conscious, then you should also believe that consciousness is maintained when the entire brain is replaced by a circuit that maps from inputs to outputs via a huge lookup table. This is disconcerting to many people (including Turing himself) who have the intuition that lookup tables are not conscious.

Discussion

We have seen that the subject of consciousness is problematic. Simple intuitions seem to lead to conflicting answers if we propose different experimental situations. But what is clear is that *if* there is an empirical question concerning the presence or absence of consciousness in appropriately programmed computers, then like any empirical question it can only be settled by experiment. Unfortunately, it is not clear what sort of experiment could settle the question, nor what sort of scientific theory could explain the results. Scientific theories are designed to account for objective phenomena; in fact Popper (1962) has characterized all physical laws as theories

that, ultimately, allow one to conclude the existence of particles in certain space-time locations. How would we view a theory that allowed one to infer pains, for example, from premises of the ordinary physical kind? It seems that it would be at best a *description* ("When a voltage is applied to such-and-such neuron, the subject will feel pain") rather than an *explanation*. An explanatory theory should, among other things, be able to explain why it is *pain* that the subject experiences when a given neuron is stimulated, rather than, say, the smell of bacon sandwiches.

It is also hard to imagine how a better understanding of neurophysiology could help. Suppose, for example, that (1) we could train a subject to record all his or her conscious thoughts without interrupting them too much; (2) neuroscientists discovered a system of neurons whose activity patterns could be decoded and understood;¹⁵ and (3) the decoded activity patterns corresponded exactly to the recorded conscious thoughts. Although we might claim to have located the "seat of consciousness," it seems we would still be no better off in our understanding of why these patterns of activity actually constitute consciousness.

The problem seems to be that consciousness, as we currently (fail to) understand it, is not understandable by the normal means available to science.

No one can rule out a major intellectual revolution that would give us a new—and at present unimaginable—concept of reduction, according to which consciousness would be reducible. (Searle, 1992, p. 124).

If consciousness is indeed irreducible, that would suggest that there can be no explanations in nonsubjective terms of why red is the sort of sensation it is and not some other sort, or why pain is like pain and not like the smell of bacon sandwiches.

One final (but not necessarily conclusive) argument can be made concerning the evolutionary origin of consciousness. Both sides of the debate agree that simple animals containing only a few neurons do not possess consciousness. In such animals, neurons fulfill a purely functional role by allowing simple adaptive and discriminative behaviors. Yet the basic design and construction of neurons in primitive animals is almost identical to the design of neurons in higher animals. Given this observation, the proponent of consciousness as an intrinsic neural phenomenon must argue that neurons, which evidently evolved for purely functional purposes, just happen by chance to have exactly the properties required to generate consciousness. The functionalist, on the other hand, can argue that consciousness necessarily emerges when systems reach the kind of functional complexity needed to sustain complex behavior.

26.5 SUMMARY

We have presented some of the main philosophical issues in AI. These were divided into questions concerning its technical feasibility (weak AI), and questions concerning its relevance and explanatory power with respect to the mind (strong AI). We concluded, although by no means conclusively, that the arguments against weak AI are needlessly pessimistic and have often mischaracterized the content of AI theories. Arguments against strong AI are inconclusive; although

¹⁵ It is not clear if this really makes sense, but the idea is to grant neuroscience as much success as we can imagine.

they fail to prove its impossibility, it is equally difficult to prove its correctness. Fortunately, few mainstream AI researchers, if any, believe that anything significant hinges on the outcome of the debate given the field's present stage of development. Even Searle himself recommends that his arguments not stand in the way of continued research on AI as traditionally conceived.

Like genetic engineering and nuclear power, artificial intelligence has its fair share of critics concerned about its possible misuses. We will discuss these concerns in the next chapter. But unlike other fields, artificial intelligence has generated a thriving industry devoted to proving its impossibility. This second batch of critics has raised some important issues concerning the basic aims, claims, and assumptions of the field. Every AI scientist and practitioner should be aware of these issues, because they directly affect the social milieu in which AI research is carried out and in which AI techniques are applied. Although general societal awareness should be part of the education of every scientist, it is especially important for AI because the nature of the field seems to arouse scepticism and even hostility in a significant portion of the population. There persists an incredible degree of misunderstanding of the basic claims and content of the field, and, like it or not, these misunderstandings have a significant effect on the field itself.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The nature of the mind has been a standard topic of philosophical theorizing from ancient times to the present. In the *Phaedo*, Plato specifically considered and rejected the idea that the mind could be an “attunement” or pattern of organization of the parts of the body, a viewpoint that approximates the functionalist viewpoint in modern philosophy of mind. He decided instead that the mind had to be an immortal, immaterial soul, separable from the body and different in substance—the viewpoint of dualism. Aristotle distinguished a variety of souls (Greek $\psi\chi'\eta$) in living things, some of which, at least, he described in a functionalist manner. (See Nussbaum (1978) for more on Aristotle's functionalism.) Aristotle also conceived of deliberation about what action to take in a way reminiscent of the modern situated agent approach (the last part of this extract also appears on the cover of the book):

But how does it happen that thinking is sometimes accompanied by action and sometimes not, sometimes by motion, and sometimes not? It looks as if almost the same thing happens as in the case of reasoning and making inferences about unchanging objects. But in that case the end is a speculative proposition . . . whereas here the conclusion which results from the two premises is an action. . . . I need covering; a cloak is a covering. I need a cloak. What I need, I have to make; I need a cloak. I have to make a cloak. And the conclusion, the "I have to make a cloak," is an action. (Nussbaum, 1978, p. 40)

Descartes is notorious for his dualistic view of the human mind, but ironically his historical influence was toward mechanism and physicalism. He explicitly conceived of animals as automata, and his spirited defense of this viewpoint actually had the effect of making it easier to conceive of humans as automata as well, even though he himself did not take this step. The book *L'Homme Machine or Man a Machine* (La Mettrie, 1748) did explicitly argue that humans are automata.

Twentieth-century analytic philosophy has typically accepted physicalism (often in the form of the brain-state "identity theory" (Place, 1956; Armstrong, 1968), which asserts that mental states are identical with brain states), but has been much more divided on functionalism, the machine analogy for the human mind, and the question of whether machines can literally think. A number of early philosophical responses to Turing's (1950) "Computing Machinery and Intelligence," for example, Scriven (1953), attempted to deny that it was even *meaningful* to say that machines could think, on the ground that such an assertion violated the meanings of the relevant terms. Scriven, at least, had retracted this view by 1963; see his addendum to a reprint of his article (Anderson, 1964). In general, later philosophical responses to AI have at least granted the meaningfulness of the question, although some might answer it vehemently in the negative.

Following the classification used by Block (1980), we can distinguish varieties of functionalism. *Functional specification theory* (Lewis, 1966; Lewis, 1980) is a variant of brain-state identity theory that selects the brain states that are to be identified with mental states on the basis of their functional role. *Functional state identity theory* (Putnam, 1960; Putnam, 1967) is more closely based on a machine analogy. It identifies mental states not with *physical* brain states but with abstract computational states of the brain conceived expressly as a computing device. These abstract states are supposed to be independent of the specific physical composition of the brain, leading some to charge that functional state identity theory is a form of dualism!

Both the brain-state identity theory and the various forms of functionalism have come under attack from authors who claim that they do not account for the *qualia* or "what it's like" aspect of mental states (Nagel, 1974). Searle has focused instead on the alleged inability of functionalism to account for intentionality (Searle, 1980; Searle, 1984; Searle, 1992). Churchland and Churchland (1982) rebut both these types of criticism.

Functionalism is the philosophy of mind most naturally suggested by AI, and critiques of functionalism often take the form of critiques of AI (as in the case of Searle). Other critics of AI, most notably Dreyfus, have focused specifically on the assumptions and research methods of AI itself, rather than its general philosophical implications. Even philosophers who are functionalists are not always sanguine about the prospects of AI as a practical enterprise (Fodor, 1983). Despite Searle's "strong"/"weak" terminology, it is possible for a philosopher to believe that human intellectual capabilities could in principle be duplicated by duplicating their functional structure alone (and thus to support "strong AI") while also believing that as a practical matter neither GOFAI nor any other human endeavor is likely to discover that functional structure in the foreseeable future (and thus to oppose "weak AI"). In fact, this seems to be a relatively common viewpoint among philosophers.

Not all philosophers are critical of GOFAI, however; some are, in fact, ardent advocates and even practitioners. Zenon Wylshyn (1984) has argued that cognition can best be understood through a computational model, not only in principle but also as a way of conducting research at present, and has specifically rebutted Dreyfus's criticisms of the computational model of human cognition (Wylshyn, 1974). Gilbert Harman (1983), in analyzing belief revision, makes connections with AI research on truth maintenance systems. Michael Bratman has applied his "belief-desire-intention" model of human psychology (Bratman, 1987) to AI research on planning (Bratman, 1992). At the extreme end of strong AI, Aaron Sloman (1978, p. xiii) has even described as "racialist" Joseph Weizenbaum's view (Weizenbaum, 1976) that hypothetical intelligent machines should not be regarded as persons.

ELIMINATIVE MATERIALISM

Eliminative materialism (Rorty, 1965; Churchland, 1979) differs from all other prominent theories in the philosophy of mind, in that it does not attempt to give an account of why our "folk psychology" or commonsense ideas about the mind are true, but instead rejects them as false and attempts to replace them with a purely scientific theory of the mind. In principle, this scientific theory could be given by classical AI, but in practice, eliminative materialists tend to lean on neuroscience and neural network research instead (Churchland, 1986), on the grounds that classical AI, especially "knowledge representation" research of the kind described in Chapter 8, tends to rely on the truth of folk psychology. Although the "intentional stance" viewpoint (Dennett, 1971) could be interpreted as functionalist, it should probably instead be regarded as a form of eliminative materialism, in that taking the "intentional stance" is not supposed to reflect any objective property of the agent toward whom the stance is taken. It should also be noted that it is possible to be an eliminative materialist about some aspects of mentality while analyzing others in some other way. For instance, Dennett (1978a) is much more strongly eliminativist about qualia than about intentionality.

The *Encyclopedia of Philosophy* (1967) is an impressively authoritative source. General collections of articles on philosophy of mind, including functionalism and other viewpoints related to AI, are *Materialism and the Mind-Body Problem* (Rosenthal, 1971) and *Readings in the Philosophy of Psychology*, volume 1 (Block, 1980). Biro and Shahan (1982) present a collection devoted to the pros and cons of functionalism. Anthologies of articles dealing more specifically with the relation between philosophy and AI include *Minds and Machines* (Anderson, 1964), *Philosophical Perspectives in Artificial Intelligence* (Ringle, 1979), *Mind Design* (Haugeland, 1981), and *The Philosophy of Artificial Intelligence* (Boden, 1990). There are several general introductions to the philosophical "AI question" (Boden, 1977; Haugeland, 1985; Copeland, 1993). *The Behavioral and Brain Sciences*, abbreviated *BBS*, is a major journal devoted to philosophical questions (and high-level, abstract scientific questions) about AI and neuroscience. A *BBS* article includes occasionally amusing peer commentary from a large number of critics and a rebuttal by the author of the main article.

EXERCISES

26.1 Go through Turing's list of alleged "disabilities" of machines, identifying which have been shown to be achievable, which are achievable in principle by a program, and which are still problematic because they require conscious mental states.

26.2 Does a refutation of the Chinese Room argument necessarily prove that appropriately programmed computers are conscious?

26.3 Suppose we ask the Chinese Room to prove that John Searle is not a conscious being. After a while, it comes up with a learned paper that looks remarkably like Searle's paper, but switches "computer" and "human" throughout, along with all the corresponding terms. The claim would be that if Searle's argument is a refutation of the possibility of conscious machines, then the Chinese Room's argument is a refutation of the possibility of conscious humans. Then, provided

we agree that humans are conscious, this refutes Searle's argument by *reductio ad absurdum*. Is this a sound argument? What might Searle's response be?

26.4 In the Brain Prosthesis argument, it is important to be able to restore the subject's brain to normal, such that its external behavior is as it would have been if the operation had not taken place. Can the sceptic reasonably object that this would require updating those neurophysiological properties of the neurons relating to conscious experience, as distinct from those involved in the functional behavior of the neurons?

26.5 Find and analyze an account in the popular media of one or more of the arguments to the effect that AI is impossible.

26.6 Under the correspondence theory, what kinds of propositions can be represented by a logical agent? A reflex (condition-action) agent?

26.7 Attempt to write definitions of the terms "intelligence," "thinking," and "consciousness." Suggest some possible objections to your definitions.

27

AI: PRESENT AND FUTURE

In which we take stock of where we are and where we are going, this being a good thing to do before continuing.

27.1 HAVE WE SUCCEEDED YET?

In Part I, we proposed a unified view of AI as intelligent agent design. We showed that the design problem depends on the percepts and actions available to the agent, the goals that the agent's behavior should satisfy, and the nature of the environment. A variety of different agent designs are possible, ranging from reflex agents to fully deliberative, knowledge-based agents. Moreover, the components of these designs can have a number of different instantiations—for example, logical, probabilistic, or "neural." The intervening chapters presented the principles by which these components operate.

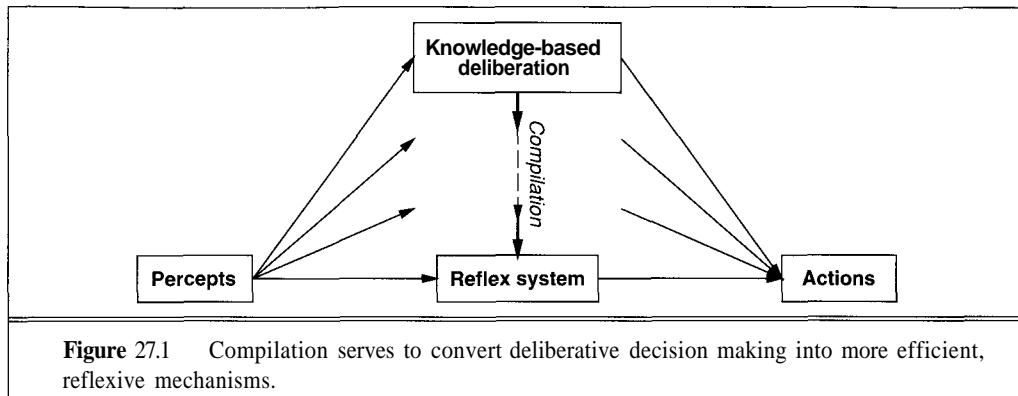
In areas such as game playing, logical inference and theorem proving, planning, and medical diagnosis, we have seen systems based on rigorous theoretical principles that can perform as well as, or better than, human experts. In other areas, such as learning, vision, robotics, and natural language understanding, rapid improvements in performance are occurring through the application of better analytical methods and improvements in our understanding of the underlying problems. Continued research will bear fruit in the form of better capabilities in all of these areas.

Enthralled by the technical details, however, one can sometimes lose sight of the big picture. We need an antidote for this tendency. *We will consider therefore whether we have the tools with which to build a complete, general-purpose intelligent agent.* This will also help to reveal a number of gaps in our current understanding.

Let us begin with the question of the agent architecture. We discussed some general principles and types in Chapter 2, and some specific architectures in Chapter 25. One key aspect of a general architecture is the ability to incorporate a variety of types of decision making, ranging from knowledge-based deliberation to reflex responses. Reflex responses are needed for situations in which time is of the essence, whereas knowledge-based deliberation allows the agent to



take into account more information, to plan ahead, to handle situations in which no immediate response is available, and to produce better responses tailored specifically for the current situation. Compilation processes such as explanation-based learning (Chapter 21) continually convert declarative information at the deliberative level into more efficient representations, eventually reaching the reflex level (Figure 27.1). Architectures such as SOAR (Laird *et al.*, 1987) and THEO (Mitchell, 1990) have exactly this structure. Every time they solve a problem by explicit deliberation, they save away a generalized version of the solution for use by the reflex component.



As we saw in Part V, uncertainty is an inevitable problem in the real world. We also saw how uncertain knowledge is one possible response to the fact that exactly correct rules in realistic environments are usually very complex, and hence unusable. Unfortunately, there are clear gaps in our understanding of how to incorporate uncertain reasoning into general-purpose agent architectures. First, very little work has been done on compilation of uncertain reasoning and decision making. Second, we need more expressive languages for uncertain knowledge—a first-order probabilistic logic. Third, we need much better mechanisms for planning under uncertainty. Current algorithms, such as policy iteration (Chapter 17), are really more analogous to the simple search algorithms of Part II than to the powerful planning methods of Part IV. The latter methods incorporate partial ordering, goal directedness, and abstraction in order to handle very complex domains. *AI is only just beginning to come to grips with the problem of integrating techniques from the logical and probabilistic worlds (Hanks *et al.*, 1994).*

Agents in real environments also need ways to control their own deliberations. They must be able to cease deliberation when action is demanded, and they must be able to use the available time for deliberation to execute the most profitable computations. For example, a taxi-driving agent that sees an accident ahead should decide either to brake or to take avoiding action in a split second rather than in half an hour. It should also spend that split second thinking about the most important questions, such as whether the lanes to the left and right are clear and whether there is a large truck close behind, rather than worrying about wear and tear on the tires or where to pick up the next passenger. These issues are usually studied under the heading of **real-time AI**. As AI systems move into more complex domains, all problems will become real-time because the agent will never have long enough to solve the decision problem exactly (see also Section 27.2). This issue came up in our discussion of game-playing systems in Chapter 5, where



we described alpha-beta pruning to eliminate irrelevant deliberations and depth limits to ensure timely play. Clearly, there is a pressing need for methods that work in more general decision-making situations. Two promising techniques have emerged in recent years. The first involves the use of **anytime algorithms** (Dean and Boddy, 1988). An anytime algorithm is an algorithm whose output quality improves gradually over time, so that it has a reasonable decision ready whenever it is interrupted. Such algorithms are controlled by a metalevel decision procedure that decides whether further computation is worthwhile. Iterative deepening search in game playing provides a simple example of an anytime algorithm. More complex systems, composed of many such algorithms working together, can also be constructed (Zilberstein, 1993).

The second technique is **decision-theoretic metareasoning** (Russell and Wefald, 1991). This method applies the theory of information value (Chapter 16) to select computations. The value of a computation depends on both its cost (in terms of delaying action) and its benefits (in terms of improved decision quality). Metareasoning techniques can be used to design better search algorithms, and automatically guarantee that the algorithms have the anytime property. Metareasoning is expensive, of course, and compilation methods can be applied to generate more efficient implementations. The application of anytime and metareasoning methods to general decision-making architectures has not yet been investigated in any depth.

The final architectural component for a general intelligent agent is the learning mechanism, or mechanisms. As the architecture becomes more complicated, the number of niches for learning increases. As we saw in Part VI, however, the same methods for inductive learning, reinforcement learning, and compilation can be used for all of the learning tasks in an agent. The learning methods do depend on the representations chosen, of course. Methods for attribute-based logical and neural representations are well understood, and methods for first-order logical representations and probabilistic representations are catching up fast. As new representations, such as first-order probabilistic logics, are developed, new learning algorithms will have to be developed to accompany them. We also will need to find a way to integrate inductive methods into the agent architecture in the same way that compilation methods are integrated into architectures such as SOAR and THEO.

An agent architecture is an empty shell without knowledge to fill it. Some have proposed that the necessary knowledge can be acquired through a training process, starting virtually from scratch. To avoid recapitulating the entire intellectual history of the human race, the training process also might include direct instruction and knowledge acquisition from sources such as encyclopedias and television programs. Although such methods might avoid the need for knowledge representation and ontological engineering work, they currently seem impractical. For the foreseeable future, useful knowledge-based systems will require some work by human knowledge engineers. Robust natural language systems, for example, may require very broad knowledge bases. Further work on general-purpose ontologies, as sketched in Chapter 8, is clearly needed. The CYC project (Lenat and Guha, 1990) is a brave effort in this direction, but many open problems remain and we have little experience in *using* large knowledge bases.

To sum up: by looking at current systems and how they would need to be extended, we can identify a variety of research questions whose answers would take us further toward general-purpose intelligent systems. This incremental approach is useful, provided we are fairly confident that we have a reasonable starting point. In the next section, we look at the AI problem from first principles to see whether this is in fact the case.

27.2 WHAT EXACTLY ARE WE TRYING TO DO?



Even before the beginning of artificial intelligence, philosophers, control theorists, and economists sought a satisfactory definition of rational action. This is needed to underpin theories of ethics, inductive learning, reasoning, optimal control, decision making, and economic modelling. *It has also been our goal in this book to show how to design agents that act rationally.* Initially, we presented no formal definition of rational action. Later chapters presented logical and decision-theoretic definitions together with specific designs for achieving them. The role of such definitions in AI is clear: if we define some desirable property P , then we ought in principle to be able to design a system that provably possesses property P . Theory meets practice when our systems exhibit P in reality. Furthermore, that they exhibit P in reality should be something that we actually care about. In a sense, the choice of what P to study determines the nature of the field. Therefore, we need to look carefully at what exactly we are trying to do.

There are a number of possible choices for P . Here are three:

PERFECT
RATIONALITY

- ◊ **Perfect rationality:** the classical notion of rationality in decision theory. A perfectly rational agent acts at every instant in such a way as to maximize its expected utility, given the information it has acquired from the environment. In Chapter 1, we warned that

achieving perfect rationality—always doing the right thing—is just not possible in complicated environments. The computational demands are just too high. However, for most of the book, we will adopt the working hypothesis that understanding perfect decision making is a good place to start.

Because perfect rational agents do not exist for nontrivial environments, perfect rationality is not a suitable candidate for P .

CALCULATIVE
RATIONALITY

- ◊ **Calculative rationality:** the notion of rationality that we have used implicitly in designing logical and decision-theoretic agents. A calculatively rational agent *eventually* returns what *would have been* the rational choice at the beginning of its deliberation. This is an interesting property for a system to exhibit because it constitutes an “in-principle” capacity to do the right thing. Calculative rationality is sometimes of limited value, because the actual behavior exhibited by such systems can be rather irrational. For example, a calculatively rational chess program may choose the right move, but may take 10^{50} times too long to do so. In practice, AI system designers are forced to compromise on decision quality to obtain reasonable overall performance, yet the theoretical basis of calculative rationality does not provide for such compromises.

BOUNDED
OPTIMALITY

- ◊ **Bounded optimality (BO):** a bounded optimal agent behaves as well as possible *given its computational resources*. That is, the expected utility of the agent program for a bounded optimal agent is at least as high as the expected utility of any other agent program running on the same machine.

Of these three possibilities, choosing P to be “bounded optimality” seems to offer the best hope for a strong theoretical foundation for AI. Clearly, a BO agent is of real, practical interest because its behavior is the best that can be obtained. Equally clearly, BO programs exist for any given task, machine, and environment. Obviously, *finding* the BO program is the trick; AI as the study

of bounded optimality is feasible in principle, but no one said it was easy! One obvious difficulty is that the designer may not have a probability distribution over the kinds of environments in which the agent is expected to operate, and so may not be able to ascertain the bounded optimality of a given design. In such cases, however, a suitably designed learning agent should be able to adapt to the initially unknown environment; analytical results on the bounded optimality of learning agents can be obtained using computational learning theory (Chapter 18). One crucial open question is to what extent bounded optimal systems can be *composed* from bounded optimal subsystems, thereby providing a hierarchical design methodology.

Although bounded optimality is a very general specification, it is no more general than the definition of calculative rationality on which much of past AI work has been based. The important thing is that by including resource constraints from the beginning, questions of efficiency and decision quality can be handled within the theory rather than by *ad hoc* system design. The two approaches only coincide if BO agents look *something like* calculatively rational agents with various efficiency improvements added on. Real environments are *far* more complex than anything that can be handled by a pure calculatively rational agent, however, so this would quite a radical assumption.

As yet, little is known about bounded optimality. It is possible to construct bounded optimal programs for very simple machines and for somewhat restricted kinds of environments (Etzioni, 1989; Russell and Subramanian, 1993), but as yet we have no idea what BO programs are like for large, general-purpose computers in complex environments. If there is to be a constructive theory of bounded optimality, we have to hope that the design of bounded optimal programs does not depend too strongly on the details of the computer being used. It would make scientific research very difficult if adding a few kilobytes of memory to a machine with 100 megabytes made a significant difference to the design of the BO program and to its performance in the environment. One way to make sure this cannot happen is to be slightly more relaxed about the criteria for bounded optimality. By analogy with asymptotic complexity (Appendix A), we can define **asymptotic bounded optimality** (ABO) as follows (Russell and Subramanian, 1993). Suppose a program P is bounded optimal for a machine M in a class of environments E (the complexity of environments in E is unbounded). Then program P' is ABO for M in E if it can outperform P by running on a machine kM that is k times faster (or larger). Unless k were enormous, we would be happy with a program that was ABO for a nontrivial environment on a nontrivial architecture. There would be little point in putting enormous effort into finding BO rather than ABO programs, because the size and speed of available machines tends to increase by a constant factor in a fixed amount of time anyway.

ASYMPTOTIC
BOUNDED
OPTIMALITY

We can hazard a guess that BO or ABO programs for powerful computers in complex environments will not necessarily have a simple, elegant structure. We have already seen that general-purpose intelligence requires some reflex capability and some deliberative capability, a variety of forms of knowledge and decision making, learning and compilation mechanisms for all of those forms, methods for controlling reasoning, and a large store of domain-specific knowledge. A bounded optimal agent must adapt to the environment in which it finds itself, so that eventually its internal organization may reflect optimizations that are specific to the particular environment. This is only to be expected, and is similar to the way in which racing cars restricted by weight and horsepower have evolved into extremely complex designs. We suspect that a science of artificial intelligence based on bounded optimality will involve a good deal of study of

the processes that allow an agent program to converge to bounded optimality, and perhaps less concentration on the details of the messy programs that result.

In summary, the concept of bounded optimality is proposed as a formal task for artificial intelligence research that is both well-defined and feasible. Bounded optimality specifies optimal *programs* rather than optimal *actions*. Actions are, after all, generated by programs, and it is over programs that designers have control.

This move from prescribing actions to prescribing programs is not unique to AI. Philosophy has also seen a gradual evolution in the definition of rationality. There has been a shift from consideration of **act utilitarianism**—the rationality of individual acts—to **rule utilitarianism**, or the rationality of general policies for acting. A philosophical proposal generally consistent with the notion of bounded optimality can be found in the "Moral First Aid Manual" (Dennett, 1986). Dennett explicitly discusses the idea of reaching equilibrium within the space of feasible configurations of decision procedures. He uses as an example the Ph.D. admissions procedure of a philosophy department. He concludes, as do we, that the best configuration may be neither elegant nor illuminating. The existence of such a configuration and the process of reaching it are the main points of interest.

ACT UTILITARIANISM
RULE UTILITARIANISM

GAME THEORY

PRISONER'S DILEMMA

Another area to undergo the same transition is **game theory**, a branch of economics initiated in the same book—*Theory of Games and Economic Behavior* (Von Neumann and Morgenstern, 1944)—that began the widespread study of decision theory. Game theory studies decision problems in which the utility of a given action depends not only on chance events in the environment but also on the actions of other agents. The standard scenario involves a set of agents who make their decisions simultaneously, without knowledge of the decisions of the other agents. The **Prisoner's Dilemma** is a famous example, in which each of two crime suspects can "collaborate" (refuse to implicate his or her partner) or "defect" (spill the beans in return for a free pardon). If the suspects collaborate, they will only be convicted of a minor offense, a one-year sentence. If they both defect, both receive a four-year sentence. If one defects and the other does not, the defector goes free whereas the other receives the maximum sentence of ten years. Considered from the point of view of either player separately, the best plan is to defect, because this gives better results whatever the other agent does. Unfortunately for the suspects (but not for the police), this results in both suspects spilling the beans and receiving a four-year sentence, whereas if they had collaborated, they would both have received lighter sentences. Even more disturbing is the fact that defection also occurs when the game has a finite number of rounds. (This can easily be proved by working backwards from the last round.) Recently, however, there has been a shift from consideration of optimal decisions in games to a consideration of optimal decision-making programs. This leads to different results because it limits the ability of each agent to do unlimited simulation of the other, who is also doing unlimited simulation of the first, and so on. Even the requirement of computability makes a significant difference (Megiddo and Wigderson, 1986). Bounds on the complexity of players have also become a topic of intense interest. Neyman's theorem (Neyman, 1985), recently proved by Papadimitriou and Yannakakis (1994), shows that a collaborative equilibrium exists if each agent is a finite automaton with a number of states that is less than exponential in the number of rounds. This is essentially a bounded optimality result, where the bound is on space rather than speed of computation. Again, the bounded optimal program for the automaton is rather messy, but its existence and properties are what counts.

27.3 WHAT IF WE Do SUCCEED?

In David Lodge's *Small World*, a novel about the academic world of literary criticism, the protagonist causes consternation by asking a panel of eminent but contradictory literary theorists the following question: "*What if you were right?*" None of the theorists seems to have considered this question before, perhaps because debating unfalsifiable theories is an end in itself. Similar confusion can sometimes be evoked by asking AI researchers, "What if you succeed?" AI is fascinating, and intelligent computers are clearly more useful than unintelligent computers, so why worry?

To the extent that AI has already succeeded in finding uses within society, we are now facing some of the real issues. In the litigious atmosphere that prevails in the United States, it is hardly surprising that legal liability needs to be discussed. When a physician relies on the judgment of a medical expert system for a diagnosis, who is at fault if the diagnosis is wrong? Fortunately, due in part to the growing influence of decision-theoretic methods in medicine, it is now accepted that negligence cannot be shown if the physician performs medical procedures that have high *expected* utility, even if the *actual* utility is catastrophic. The question should therefore be, "Who is at fault if the diagnosis is unreasonable?" So far, courts have held that medical expert systems play the same role as medical textbooks and reference books; physicians are responsible for understanding the reasoning behind any decision and for using their own judgment in deciding whether or not to accept the system's recommendations. In designing medical expert systems as agents, therefore, the actions should not be thought of as directly affecting the patient but as influencing the physician's behavior. If expert systems become reliably more accurate than human diagnosticians, doctors may be legally liable if they *fail* to use the recommendations of an expert system.

Similar issues are beginning to arise regarding the use of intelligent agents on the "information highway." Some progress has been made in incorporating constraints into intelligent agents so that they cannot damage the files of other users (Weld and Etzioni, 1994). Also problematic is the fact that network services already involve monetary transactions. If those monetary transactions are made "on one's behalf" by an intelligent agent, is one liable for the debts incurred? Would it be possible for an intelligent agent to have assets itself and to perform electronic trades on its own behalf? Could it own stocks and bonds in the same way that corporations own stocks and bonds? So far, these questions do not seem to be well understood. To our knowledge, no program has been granted legal status as an individual for the purposes of financial transactions; at present, it seems unreasonable to do so. Programs are also not considered to be "drivers" for the purposes of enforcing traffic regulations on real highways. In California law, at least, there do not seem to be any legal sanctions to prevent an automated vehicle from exceeding the speed limits, although the designer of the vehicle's control mechanism would be liable in the case of accident. As with human reproductive technology, the law has yet to catch up with the new developments. These topics, among others, are covered in journals such as *AI and Society*, *Law, Computers and Artificial Intelligence*, and *Artificial Intelligence and Law*.

Looking further into the future, one can anticipate questions that have been the subject of innumerable works of science fiction, most notably those of Asimov (1942). If we grant that

machines will achieve high levels of intelligent behavior and will communicate with humans as apparent equals, then these questions are unavoidable. Should (or will) intelligent machines have rights? How should intelligent machines interact with humans? What might happen if intelligent machines decide to work against the best interests of human beings? What if they succeed?

In *Computer Power and Human Reason*, Joseph Weizenbaum (the author of the ELIZA program) has argued that the effect of intelligent machines on human society will be such that continued work on artificial intelligence is perhaps unethical. One of Weizenbaum's principal arguments is that AI research makes possible the idea that humans are automata—an idea that results in a loss of autonomy or even of humanity. (We note that the idea has been around much longer than AI. See *L'Homme Machine* (La Mettrie, 1748).) One can perhaps group such concerns with the general concern that any technology can be misused to the detriment of humanity. Arguments over the desirability of a given technology must weigh the benefits and risks, and put the onus on researchers to ensure that policy makers and the public have the best possible information with which to reach a decision. On the other hand, AI raises deeper questions than, say, nuclear weapons technology. No one, to our knowledge, has suggested that reducing the planet to a cinder is better than preserving human civilization. Futurists such as Edward Fredkin and Hans Moravec have, however, suggested that once the human race has fulfilled its destiny in bringing into existence entities of higher (and perhaps unlimited) intelligence, its own preservation may seem less important. Something to think about, anyway.

Looking on the bright side, success in AI would provide great opportunities for improving the material circumstances of human life. Whether it would improve the quality of life is an open question. Will intelligent automation give people more fulfilling work and more relaxing leisure time? Or will the pressures of competing in a nanosecond-paced world lead to more stress? Will children gain from instant access to intelligent tutors, multimedia online encyclopedias, and global communication, or will they play ever more realistic war games? Will intelligent machines extend the power of the individual, or of centralized governments and corporations? Science fiction authors seem to favor dystopian futures over Utopian ones, probably because they make for more interesting plots. In reality, however, the trends seem not to be too terribly negative.

A

COMPLEXITY ANALYSIS AND O() NOTATION

BENCHMARKING

Computer scientists are often faced with the task of comparing two algorithms to see which runs faster or takes less memory. There are two approaches to this task. The first is **benchmarking**—running the two algorithms on a computer and measuring which is faster (or which uses less memory). Ultimately, this is what really matters, but a benchmark can be unsatisfactory because it is so specific: it measures the performance of a particular program written in a particular language running on a particular computer with a particular compiler and particular input data. From the single result that the benchmark provides, it can be difficult to predict how well the algorithm would do on a different compiler, computer, or data set. A useful variant of benchmarking is to count the number of operations performed of a particular kind: for example, in testing a numerical sorting algorithm we might count the number of “greater-than” tests.

A.1 ASYMPTOTIC ANALYSIS

ANALYSIS OF ALGORITHMS

The second approach relies on a mathematical **analysis of algorithms**, independent of the particular implementation and input. We will discuss the approach with the following example, a program to compute the sum of a sequence of numbers:

```
function SUMMATION(sequence) returns a number
    sum — 0
    for i — 1 to LENGTH(sequence)
        sum — sum + sequence[i]
    end
    return sum
```

The first step in the analysis is to abstract over the input, to find some parameter or parameters that characterize the size of the input. In this example, the input can be characterized by the length of the sequence, which we will call n . The second step is to abstract over the implementation, to find some measure that reflects the running time of the algorithm, but is not tied to a particular

compiler or computer. For the SUMMATION program, this could be just the number of lines of code executed. Or it could be more detailed, measuring the number of additions, assignments, array references, and branches executed by the algorithm. Either way gives us a characterization of the total number of steps taken by the algorithm, as a function of the size of the input. We will call this $T(n)$. With the simpler measure, we have $T(n) = 2n + 2$ for our example.

If all programs were as simple as SUMMATION, analysis of algorithms would be a trivial field. But two problems make it more complicated. First, it is rare to find a parameter like n that completely characterizes the number of steps taken by an algorithm. Instead, the best we can usually do is compute the worst case $T_{\text{worst}}(n)$ or the average case $T_{\text{avg}}(n)$. Computing an average means that the analyst must assume some distribution of inputs.

The second problem is that algorithms tend to resist exact analysis. In that case, it is necessary to fall back on an approximation. We say that the SUMMATION algorithm is $O(n)$, meaning that its measure is at most a constant times n , with the possible exception of a few small values of n . More formally,

$$T(n) \text{ is } O(f(n)) \text{ if } T(n) < kf(n) \text{ for some } k, \text{ for all } n > n_0$$

ASYMPTOTIC ANALYSIS

The OQ notation gives us what is called an **asymptotic analysis**. We can say without question that as n asymptotically approaches infinity, an $O(n)$ algorithm is better than an $O(n^2)$ algorithm. A single benchmark figure could not substantiate such a claim.

The OQ notation abstracts over constant factors, which makes it easier to use than the $T()$ notation, but less precise. For example, an $O(n^2)$ algorithm will always be worse than an $O(n)$ in the long run, but if the two algorithms are $T(n^2+1)$ and $T(100n+1000)$, then the $O(n^2)$ algorithm is actually better for $n < 110$.

Despite this drawback, asymptotic analysis is the most widely used tool for analyzing algorithms. It is precisely because the analysis abstracts both over the exact number of operations (by ignoring the constant factor, k) and the exact content of the input (by only considering its size, n) that the analysis becomes mathematically feasible. The OQ notation is a good compromise between precision and ease of analysis.

A.2 INHERENTLY HARD PROBLEMS

COMPLEXITY ANALYSIS

Analysis of algorithms and the OQ notation allow us to talk about the efficiency of a particular algorithm. However, they have nothing to say about whether or not there could be a better algorithm for the problem at hand. The field of **complexity analysis** analyzes problems rather than algorithms. The first gross division is between problems that can be solved in polynomial time and those that cannot be solved in polynomial time, no matter what algorithm is used. The class of polynomial problems is called P. These are sometimes called "easy" problems, because the class contains those problems with running times like $O(\log n)$ and $O(n)$. But it also contains those with $O(n^{1000})$, so the name "easy" should not be taken too literally.

Another important class of problems is NP, the class of nondeterministic polynomial problems. A problem is in this class if there is some algorithm that can guess a solution and then verify whether or not the guess is correct in polynomial time. The idea is that if you either have

an exponentially large number of processors so that you can try all the guesses at once, or you are very lucky and always guess right the first time, then the NP problems become P problems.

One of the big open questions in computer science is whether the class NP is equivalent to the class P when one does not have the luxury of an infinite number of processors or omniscient guessing. Most computer scientists are convinced that $P \neq NP$, that NP problems are inherently hard and only have exponential time algorithms. But this has never been proven.

NP-COMPLETE

Those who are interested in deciding if $P = NP$ look at a subclass of NP called the **NP-complete** problems. The word complete is used here in the sense of "most extreme," and thus refers to the hardest problems in the class NP. It has been proven that either all the NP-complete problems are in P or none of them is. This makes the class theoretically interesting, but the class is also of practical interest because many important problems are known to be NP-complete. An example is the satisfiability problem: given a logical expression (see Chapter 6), is there an assignment of truth values to the variables of the expression that make it true?

Also studied is the class of PSPACE problems, those that require a polynomial amount of space, even on a nondeterministic machine. It is generally believed that PSPACE-hard problems are worse than NP-complete, although it could turn out that $NP = PSPACE$, just as it could turn out that $P = NP$.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The $O()$ notation so widely used in computer science today was first introduced in the context of number theory by the German mathematician P. G. H. Bachmann (1894). The concept of NP-completeness was invented by Cook (1971), and the modern method for establishing a reduction from one problem to another is due to Karp (1972). Cook and Karp have both won the Turing award, the highest honor in computer science, for their work.

Classic works on the analysis and design of algorithms include those by Knuth (1973) and Aho, Hopcroft, and Ullman (1974); more recent contributions are by Tarjan (1983) and Cormen, Leiserson, and Rivest (1990). These books place an emphasis on designing and analyzing algorithms to solve tractable problems. For the theory of NP-completeness and other forms of intractability, the best introduction is by Garey and Johnson (1979). In addition to the underlying theory, Garey and Johnson provide examples that convey very forcefully why computer scientists are unanimous in drawing the line between tractable and intractable problems at the border between polynomial and exponential time complexity. They also provide a voluminous catalog of problems that are known to be NP-complete or otherwise intractable.

B

NOTES ON LANGUAGES AND ALGORITHMS

B.1 DEFINING LANGUAGES WITH BACKUS-NAUR FORM (BNF)

BACKUS-NAUR
FORM
BNF
TERMINAL SYMBOLS

NONTERMINAL
SYMBOLS

START SYMBOL

In this book, we define several languages, including the languages of propositional logic (page 166), first-order logic (page 187), and a subset of English (page 670). A formal language is defined as a set of strings where each string is a sequence of symbols. All the languages we are interested in consist of an infinite set of strings, so we need a concise way to characterize the set. We do that with a **grammar**. We have chosen to write our grammars in a formalism called **Backus-Naur form**, or **BNF**. There are four components to a BNF grammar:

- **A set of terminal symbols.** These are the symbols or words that make up the strings of the language. They could be letters (A, B, C, ...) or words (a, **ardvark**, **abacus**, ...) for English). For the language of arithmetic, the set of symbols is

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, \times, (), \}$$

- **A set of nonterminal symbols** that categorize subphrases of the language. For example, the nonterminal symbol *NounPhrase* in English denotes an infinite set of strings including "you" and "the big slobbery dog."
- **A start symbol**, which is the nonterminal symbol that denotes the complete strings of the language. In English, this is *Sentence*; for arithmetic, it might be *Exp*.
- **A set of rewrite rules or productions** of the form *LHS* \rightarrow *RHS*, where *LHS* is a nonterminal, and *RHS* is a sequence of zero or more symbols (either terminal or nonterminal).

A rewrite rule of the form

$$\textit{Digit} \rightarrow 7$$

means that anytime we see the string consisting of the lone symbol 7, we can categorize it as a *Digit*. A rule of the form

$$\textit{Sentence} \rightarrow \textit{NounPhrase} \textit{VerbPhrase}$$

means that whenever we have two strings categorized as a *NounPhrase* and a *VerbPhrase*, we can append them together and categorize the result as a *Sentence*. As an abbreviation, the symbol |

can be used to separate alternative right-hand sides. Here is a BNF grammar for simple arithmetic expressions:

```
Exp      → Exp Operator Exp
          | ( Exp )
          | Number -
Number   → Digit
          | Number Digit
Digit    → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Operator → + | - | ÷ | X
```

We cover languages and grammars in more detail in Chapter 22. Be aware that other books use slightly different notations for BNF; for example, you might see *(Digit)* instead of *Digit* for a nonterminal; ‘word’ instead of **word** for a terminal; or `:` instead of `→` in a rule.

B.2 DESCIBING ALGORITHMS WITH PSEUDO-CODE

In this book, we define over 100 algorithms. Rather than picking a programming language (and risking that readers who are unfamiliar with the language will be lost), we have chosen to describe the algorithms in pseudo-code. Most of it should be familiar to users of languages like Pascal, C, or Common Lisp, but in some places we use mathematical formulas or ordinary English to describe parts that would otherwise be more cumbersome. There are a few idiosyncrasies that should be remarked on.

Nondeterminism

It is the nature of AI that we are often faced with making a decision before we know enough to make the right choice. So our algorithms have to make a choice, but keep track of the alternatives in case the choice does not succeed. The clearest way to describe such algorithms without bogging them down with bookkeeping details is with the primitives **choose** and **fail**.

The idea is that when we call **choose**(*a,b,c*), the algorithm will return either *a*, *b*, or *c* as the value of the **choose** expression. But it will also save the other two on an agenda of pending choices. The algorithm continues; if it terminates normally then all is done, and we forget about the agenda. But if a **fail** statement is encountered, then a pending choice is taken off the agenda, and control is resumed *at the point in the algorithm where that choice was saved*. Algorithms that make use of **choose** are called **nondeterministic algorithms**.

You can think of a nondeterministic algorithm as a search through the space of possible choices. As such, any of the search algorithms from Chapters 3 or 4 can be used. The beauty of the nondeterministic algorithm is that the search strategy can be specified separately from the main algorithm.

ORACLE

Another way to think of nondeterministic algorithms is to imagine an **oracle** that magically advises the algorithm to make the correct choice at every choice point. If such an oracle could be found, the algorithm would be deterministic; it would never need to backtrack. You can think of the agenda as a slower means of simulating the advice of the oracle.

Nondeterministic algorithms are often clearer than deterministic versions, but unfortunately only a few programming languages support nondeterminism directly—Prolog and ICON are probably the best known. **choose and fail** can be implemented in SCHEME in about 15 lines of code, using the function `call-with-current-continuation`.

Here is an example of a nondeterministic function: the function call `INTEGER(start)` returns an integer greater than or equal to *start*, chosen nondeterministically.

```
function INTEGER(start) returns an integer
    return choose(start, INTEGER(start+ 1))
```

It is important to understand what the code fragment `PRINT(INTEGER (0)); fail` will do. First, it will print some integer, although we cannot say which one without knowing what control strategy is used by **choose**. Then **fail** will return control to one of the choice points in one of the recursive calls to `INTEGER`. Eventually, a different integer will be chosen and printed, and then **fail** will be executed again. The result is an infinite loop, with a single integer printed each time.

We sometimes use the term **pick** to mean a choice that is not a backtracking point. For example, an algorithm to sum the elements of a set is to initialize the total to 0, pick an element from the set and add it to the total, and continue until there are no more elements to pick. There is no need to backtrack, because any order of picking will get the job done.

Static variables

We use the keyword **static** to say that a variable is given an initial value the first time a function is called and retains that value (or the value given to it by a subsequent assignment statement) on all subsequent calls to the function. Thus, static variables are like global variables in that they outlive a single call to their function, but they are only accessible within the function. The agent programs in the book use static variables for "memory." Programs with static variables can be implemented as "objects" in object-oriented languages such as C++ and Smalltalk. In functional languages, they can be implemented easily by executing lambda-expressions within an environment in which the required variables are defined.

Functions as values

Functions and procedures have capitalized names and variables have lower case italic names. So most of the time, a function call looks like `SOME-FUNCTION(variable)`. However, we allow the value of a variable to be a function; for example, if the value of *variable* is the square root function, then `variable(9)` returns 3.

B.3 THE CODE REPOSITORY

The pseudo-code in the book is meant to be easy to read and understand, but it is not easy to run on a computer. To fix this problem, we have provided a repository of working code. Most of the algorithms in the book are implemented, as well as a set of basic tools not covered in the book. Currently, the algorithms are all written in Lisp, although we may add other languages in the future. If you are reading this book as part of a course, your instructor will probably retrieve the code for you. If not, send an electronic mail message to

aima-request@cs.berkeley.edu

with the word "help" in the subject line or in the body. You will receive a return message with instructions on how to get the code. (We don't print the instructions here because they are subject to frequent change.) You can also order the code on a floppy disk by writing to Prentice-Hall Inc., Englewood Cliffs, NJ, 07632.

B.4 COMMENTS

If you have any comments on the book, any typos you have noticed, or any suggestions on how it can be improved, we would like to hear from you. Please send a message to

aima-bug@cs.berkeley.edu

or write to us in care of Prentice-Hall.

Bibliography

- Aarup, M., Arentoft, M. M., Parrod, Y., Stader, J., and Stokes, I. (1994). OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Fox, M. and Zweben, M., editors, *Knowledge Based Scheduling*. Morgan Kaufmann, San Mateo, California.
- Abu-Mostafa, Y. S. and Psaltis, D. (1987). Optical neural computers. *Scientific American*, 256:88–95.
- Acharya, A., Tambe, M., and Gupta, A. (1992). Implementation of production systems on message-passing computers. *IEEE Transactions on Parallel and Distributed Systems*, 3(4):477–487.
- Adelson-Velsky, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovsky, A. A., and Uskov, A. V. (1970). Programming a computer to play chess. *Russian Mathematical Surveys*, 25:221–262.
- Adelson-Velsky, G. M., Arlazarov, V. L., and Donskoy, M. V. (1975). Some methods of controlling the tree search in chess programs. *Artificial Intelligence*, 6(4):361–371.
- Agmon, S. (1954). The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6(3):382–392.
- Agre, P. E. and Chapman, D. (1987). Pengi: an implementation of a theory of activity. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 268–272, Milan, Italy. Morgan Kaufmann.
- Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts.
- Ait-Kaci, H. (1991). *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press, Cambridge, Massachusetts.
- Ait-Kaci, H. and Nasr, R. (1986). LOGIN: a logic programming language with built-in inheritance. *Journal of Logic Programming*, 3(3): 185–215.
- Ait-Kaci, H. and Podelski, A. (1993). Towards a meaning of LIFE. *Journal of Logic Programming*, 16(3–4): 195–234.
- Allais, M. (1953). Le comportement de l'homme rationnel devant la risque: critique des postulats et axiomes de l'école Américaine. *Econometrica*, 21:503–546.
- Alien, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the Association for Computing Machinery*, 26(11):832–843.
- Alien, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154.
- Alien, J. F. (1991). Time and time again: the many ways to represent time. *International Journal of Intelligent Systems*, 6:341–355.
- Alien, J. F. (1995). *Natural Language Understanding*. Benjamin/Cummings, Redwood City, California.
- Alien, J. F., Hendler, J., and Tate, A., editors (1990). *Readings in Planning*. Morgan Kaufmann, San Mateo, California.
- Almuallim, H. and Dietterich, T. G. (1991). Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 547–552, Anaheim, California. AAAI Press.
- Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. *International Journal of Computer Vision*, 1:333–356.
- Aloimonos, Y. (1992). Special issue on purposive, qualitative, active vision. *CVGIP: Image Understanding*, 56(1).
- Alshawi, H., editor (1992). *The Core Language Engine*. MIT Press, Cambridge, Massachusetts.
- Alspector, J., Alien, R. B., Hu, V., and Satyanarayana, S. (1987). Stochastic learning networks and their electronic implementation. In Anderson, D. Z., editor, *Neural Information Processing Systems, Denver 1987*, pages 9–21, Denver, Colorado. American Institute of Physics.
- Alterman, R. (1988). Adaptive planning. *Cognitive Science*, 12:393–422.
- Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie, D., editor, *Machine Intelligence 3*, volume 3, pages 131–171. Elsevier/North-Holland, Amsterdam, London, New York.

- Ambros-Ingeron, J. and Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 735-740, St. Paul, Minnesota. Morgan Kaufmann.
- Amit, D., Gutfreund, H., and Sompolinsky, H. (1985). Spin-glass models of neural networks. *Physical Review A* 32:1007-1018.
- Ammon, K. (1993). An automatic proof of Gödel's incompleteness theorem. *Artificial Intelligence*, 61(2):291-306.
- Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. (1989). HUGIN—a shell for building Bayesian beliefuniverses for expertsystems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 2, pages 1080-1085, Detroit, Michigan. Morgan Kaufmann.
- Anderson, A. R., editor (1964). *Minds and Machines*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Anderson, J. (1980). *Cognitive Psychology and its Implications*. W. H. Freeman, New York.
- Anderson, J. A. and Rosenfeld, E., editors (1988). *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, Massachusetts.
- Anderson, J. R. (1983). *The Architecture of Cognition*. Harvard University Press, Cambridge, Massachusetts.
- Armstrong, D. M. (1968). *A Materialist Theory of the Mind*. Routledge and Kegan Paul, London.
- Arnauld, A. (1662). *La logique, ou l'art de penser*. Chez Charles Savreux, au pied de la Tour de Nostre Dame, Paris. Usually referred to as the *Port-Royal Logic*; translated into English as Arnauld (1964).
- Arnauld, A. (1964). *The Art of Thinking*. Bobbs-Merrill, Indianapolis, Indiana. Translation of Arnauld (1662), usually referred to as the *Port-Royal Logic*.
- Ashby, W. R. (1952). *Design for a Brain*. Wiley, New York.
- Asimov, I. (1942). Runaround. *Astounding Science Fiction*.
- Asimov, I. (1950). *I, Robot*. Doubleday, Garden City, New York.
- Astrom, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.*, 10:174-205.
- Austin, J. L. (1962). *How To Do Things with Words*. Harvard University Press, Cambridge, Massachusetts.
- Bacchus, F. (1990). *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, Cambridge, Massachusetts.
- Bacchus, F., Grove, A., Halpern, J. Y., and Koller, D. (1992). From statistics to beliefs. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 602-608, San Jose, California. AAAI Press.
- Bach, E. (1986). The algebra of events. *Linguistics and Philosophy*, 9:5-16.
- Bachmann, P. G. H. (1894). *Die analytisch-Zahlentheorie*. B. G. Teubner, Leipzig.
- Bain, M. and Muggleton, S. H. (1991). Non-monotonic learning. In Hayes, J. E., Michie, D., and Tyugu, E., editors, *Machine Intelligence 12*, pages 105-119. Oxford University Press, Oxford.
- Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE*, 76(8):996-1005.
- Bajcsy, R. and Lieberman, L. (1976). Texture gradient as a depth cue. *Computer Graphics and Image Processing*, 5(1):52-67.
- Baker, C. L. (1989). *English Syntax*. MIT Press, Cambridge, Massachusetts.
- Baker, J. (1975). The Dragon system—an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23.
- Ballard, B. W. (1983). The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3):327-350.
- Bar-Hillel, Y. (1954). Indexical expressions. *Mind*, 63:359-379.
- Bar-Hillel, Y. (1960). The present status of automatic translation of languages. In Alt, F. L., editor, *Advances in Computers*. Academic Press, New York.
- Bar-Shalom, Y. and Fortmann, T. E. (1988). *Tracking and Data Association*. Academic Press, New York.

- Barr, A., Cohen, P. R., and Feigenbaum, E. A., editors (1989). *The Handbook of Artificial Intelligence*, volume 4. Addison-Wesley, Reading, Massachusetts.
- Barr, A. and Feigenbaum, E. A., editors (1981). *The Handbook of Artificial Intelligence*, volume 1. HeurisTech Press and William Kaufmann, Stanford, California and Los Altos, California. First of four volumes; other volumes published separately (Barr and Feigenbaum, 1982; Cohen and Feigenbaum, 1982; Barr et al., 1989).
- Barr, A. and Feigenbaum, E. A., editors (1982). *The Handbook of Artificial Intelligence*, volume 2. HeurisTech Press and William Kaufmann, Stanford, California and Los Altos, California.
- Barrett, A., Golden, K., Penberthy, J. S., and Weld, D. S. (1993). UCPOP user's manual (version 2.0). Technical Report 93-09-06, Department of Computer Science and Engineering, University of Washington.
- Barrett, R., Ramsay, A., and Sloman, A. (1985). *POP-11: A Practical Language for Artificial Intelligence*. Ellis Horwood, Chichester, England.
- Barstow, D. R. (1979). *Knowledge-Based Program Construction*. Elsevier/North-Holland, Amsterdam, London, New York.
- Barto, A. G., Bradtko, S. J., and Singh, S. P. (1991). Real-time learning and control using asynchronous dynamic programming. Technical Report TR-91-57, University of Massachusetts Computer Science Department, Amherst, Massachusetts.
- Barto, A. G., Sutton, R. S., and Brouwer, P. S. (1981). Associative search network: a reinforcement learning associative memory. *Biological Cybernetics*, 40(3):201–211.
- Barwise, J. (1993). Everyday reasoning and logical inference. *Behavioral and Brain Sciences*, 16(2):337–338.
- Barwise, J. and Etchemendy, J. (1993). *The Language of First-Order Logic: Including the Macintosh Program Tarski's World 4.0*. Center for the Study of Language and Information (CSLI), Stanford, California, third revised and expanded edition.
- Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 41.
- Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53:370–418.
- Beal, D. F. (1980). An analysis of minimax. In Clarke, M. R. B., editor, *Advances in Computer Chess 2*, pages 103–109. Edinburgh University Press, Edinburgh, Scotland.
- Beck, H. W., Gala, S. K., and Navathe, S. B. (1989). Classification as a query processing technique in the CANDIDE semantic data model. In *Proceedings Fifth International Conference on Data Engineering*, pages 572–581, Los Angeles, California. IEEE Computer Society Press.
- Belhumeur, P. N. (1993). A binocular stereo algorithm for reconstructing sloping, creased, and broken surfaces in the presence of half-occlusion. In *Proceedings of the 4th International Conference on Computer Vision*, Berlin. IEEE Computer Society Press.
- Bell, C. and Tate, A. (1985). Using temporal constraints to restrict search in a planner. In *Proceedings of the Third Alvey IKBS SIG Workshop*, Sunningdale, Oxfordshire.
- Bell, J. L. and Machover, M. (1977). *A Course in Mathematical Logic*. Elsevier/North-Holland, Amsterdam, London, New York.
- Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.
- Bellman, R. E. (1978). *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, San Francisco.
- Bellman, R. E. and Dreyfus, S. E. (1962). *Applied Dynamic Programming*. Princeton University Press, Princeton, New Jersey.
- Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). *Winning Ways, For Your Mathematical Plays*. Academic Press, New York.
- Berliner, H. J. (1977). BKG—A program that plays backgammon. Technical report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.
- Berliner, H. J. (1979). The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, 12(1):23–40.
- Berliner, H. J. (1980a). Backgammon computerprogram beats world champion. *Artificial Intelligence*, 14:205–220.

- Berliner, H. J. (1980b). Computer backgammon. *Scientific American*, 249(6):64-72.
- Berliner, H. J. (1989). Hitech chess: From master to senior master with no hardware change. In *MIV-89: Proceedings of the International Workshop on Industrial Applications of Machine Intelligence and Vision (Seiken Symposium)*, pages 12-21.
- Berliner, H. J. and Ebeling, C. (1989). Pattern knowledge and search: The SUPREM architecture. *Artificial Intelligence*, 38(2): 161-198.
- Berliner, H. J. and Goetsch, G. (1984). A quantitative study of search methods and the effect of constraint satisfaction. Technical Report CMU-CS-84-187, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.
- Bernoulli, D. (1738). Specimen theoriae novae de mensura sortis. *Proceedings of the St. Petersburg Imperial Academy of Sciences*, 5. Translated into English as Bernoulli (1954).
- Bernoulli, D. (1954). Exposition of a new theory of the measurement of risk. *Econometrica*, 22:123-136. Translation of Bernoulli (1738) by Louise Sommer.
- Bernstein, A. and Roberts, M. (1958). Computer vs. chess player. *Scientific American*, 198(6):96-105.
- Bernstein, A., Roberts, M., Arbuckle, T., and Belksky, M. S. (1958). A chess playing program for the IBM 704. In *Proceedings of the 1958 Western Joint Computer Conference*, pages 157-159, Los Angeles.
- Berry, D. A. and Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London.
- Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Beth, E. W. (1955). Semantic entailment and formal derivability. *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde, N.R.*, 18(13):309-342.
- Bibel, W. (1981). On matrices with connections. *Journal of the Association for Computing Machinery*, 28(4):633-645.
- Bibel, W. (1986). A deductive solution for plan generation. *New Generation Computing*, 4(2): 115-132.
- Birnbaum, L. and Selfridge, M. (1981). Conceptual analysis of natural language. In Schank, R. and Riesbeck, C., editors, *Inside Computer Understanding*. Lawrence Erlbaum.
- Biro, J. I. and Shahan, R. W., editors (1982). *Mind, Brain and Function: Essays in the Philosophy of Mind*. University of Oklahoma Press, Norman, Oklahoma.
- Birtwistle, G., Dahl, O.-J., Myrhaug, B., and Nygaard, K. (1973). *Simula Begin*. Studentlitteratur (Lund) and Auerbach, New York.
- Bitner, J. R. and Reingold, E. M. (1975). Backtrack programming techniques. *Communications of the Association for Computing Machinery*, 18(11):651-656.
- Black, E., Jelinek, F., Lafferty, J., Magerman, D., Mercer, R., and Roukos, S. (1992). Towards history-based grammars: using richer models for probabilistic parsing. In Marcus, M., editor, *Fifth DARPA Workshop on Speech and Natural Language*, Arden Conference Center, Harriman, New York.
- Block, N., editor (1980). *Readings in Philosophy of Psychology*, volume 1. Harvard University Press, Cambridge, Massachusetts.
- Bloom, P. (1994). *Language Acquisition: Core Readings*. MIT Press.
- Blum, A. L. and Rivest, R. L. (1992). Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117-127.
- Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929-965.
- Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1990). Occam's razor. In Shavlik, J. W. and Dietterich, T. G., editors, *Readings in Machine Learning*, pages 201-204. Morgan Kaufmann.
- Board, R. and Pitt, L. (1992). On the necessity of Occam algorithms. *Theoretical Computer Science*, 100(1):157-184.
- Bobrbw, D. G. (1967). Natural language input for a computer problem solving system. In Minsky, M. L., editor, *Semantic Information Processing*, pages 133-215. MIT Press, Cambridge, Massachusetts.

- Bobrow, D. G. and Raphael, B. (1974). New programming languages for artificial intelligence research. *Computing Surveys*, 6(3):153-174.
- Boden, M. A. (1977). *Artificial Intelligence and Natural Man*. Basic Books, New York.
- Boden, M. A., editor (1990). *The Philosophy of Artificial Intelligence*. Oxford University Press, Oxford.
- Boole, G. (1847). *The Mathematical Analysis of Logic: Being an Essay towards a Calculus of Deductive Reasoning*. Macmillan, Barclay, and Macmillan, Cambridge.
- Boolos, G. S. (1990). On "seeing" the truth of the Gödel sentence. *Behavioral and Brain Sciences*, 13(4):655-656. Peer commentary on Penrose (1990).
- Boolos, G. S. and Jeffrey, R. C. (1989). *Computability and Logic*. Cambridge University Press, Cambridge, third edition.
- Borgida, A., Brachman, R. J., McGuinness, D. L., and Alperin Resnick, L. (1989). CLASSIC: a structural data model for objects. *SIGMOD Record*, 18(2):58-67.
- Boyer, R. S. (1971). *Locking: A Restriction of Resolution*. PhD thesis, University of Texas, Austin, Texas.
- Boyer, R. S. and Moore, J. S. (1972). The sharing of structure in theorem-proving programs. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 7*, pages 101–116. Edinburgh University Press, Edinburgh, Scotland.
- Boyer, R. S. and Moore, J. S. (1979). *A Computational Logic*. Academic Press, New York.
- Boyer, R. S. and Moore, J. S. (1984). Proofchecking the RSA public key encryption algorithm. *American Mathematical Monthly*, 91(3):181–189.
- Brachman, R. J. (1979). On the epistemological status of semantic networks. In Findler, N. V., editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 3-50. Academic Press, New York.
- Brachman, R. J., Fikes, R. E., and Levesque, H. J. (1983). Krypton: A functional approach to knowledge representation. *Computer*, 16(10):67–73.
- Brachman, R. J. and Levesque, H. J., editors (1985). *Readings in Knowledge Representation*. Morgan Kaufmann, San Mateo, California.
- Bransford, J. and Johnson, M. K. (1973). Consideration of some problems in comprehension. In Chase, W. G., editor, *Visual Information Processing*. Academic Press, New York.
- Bratko, I. (1986). *Prolog Programming for Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, first edition.
- Bratko, I. (1990). *Prolog Programming for Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, second edition.
- Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, Massachusetts.
- Bratman, M. E. (1992). Planning and the stability of intention. *Minds and Machines*, 2(1):1-16.
- Brelaz, D. (1979). New methods to color the vertices of a graph. *Communications of the Association for Computing Machinery*, 22(4):251–256.
- Bresnan, J. (1982). *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts.
- Briggs, R. (1985). Knowledge representation in Sanskrit and artificial intelligence. *AI Magazine*, 6(1):32–39.
- Brooks, R. A. (1981). Symbolic reasoning among 3-D models and 2-D images. *Artificial Intelligence*, 17:285–348.
- Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23.
- Brooks, R. A. (1989). Engineering approach to building complete, intelligent beings. *Proceedings of the SPIE—The International Society for Optical Engineering*, 1002:618–625.
- Brudno, A. L. (1963). Bounds and valuations for shortening the scanning of variations. *Problems of Cybernetics*, 10:225–241.
- Bryson, A. E. and Ho, Y.-C. (1969). *Applied Optimal Control*. Blaisdell, New York.
- Buchanan, B. G. and Mitchell, T. M. (1978). Model-directed learning of production rules. In Waterman, D. A. and Hayes-Roth, E., editors, *Pattern-Directed Inference Systems*, pages 297–312. Academic Press, New York.

- Buchanan, B. G., Mitchell, T. M., Smith, R. G., and Johnson, C. R. (1978). Models of learning systems. In *Encyclopedia of Computer Science and Technology*, volume 11. Dekker.
- Buchanan, B. G. and Shortliffe, E. H., editors (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Massachusetts.
- Buchanan, B. G., Sutherland, G. L., and Feigenbaum, E. A. (1969). Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry. In Meltzer, B., Michie, D., and Swann, M., editors, *Machine Intelligence 4*, pages 209-254. Edinburgh University Press, Edinburgh, Scotland.
- Buchler, J., editor (1955). *Philosophical Writings of Peirce*. Dover, New York.
- Bundy, A. (1983). *The Computer Modelling of Mathematical Reasoning*. Academic Press, New York.
- Bunt, H. C. (1985). The formal representation of (quasi-) continuous concepts. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, chapter 2, pages 37-70. Ablex, Norwood, New Jersey.
- Burstall, R. M. (1974). Program proving as hand simulation with a little induction. In *Information Processing '74*, pages 308-312. Elsevier/North-Holland, Amsterdam, London, New York.
- Burstall, R. M. and Darlington, J. (1977). A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery*, 24(1):44-67.
- Bylander, T. (1992). Complexity results for serial decomposability. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 729-734, San Jose, California. AAAI Press.
- Caianello, E. R. (1961). Outline of a theory of thought and thinking machines. *Journal of Theoretical Biology*, 1:204-235.
- Campbell, P. K., Johnes, K. E., Huber, R. J., Horch, K. W., and Normann, R. A. (1991). A silicon-based, 3-dimensional neural interface: manufacturing processes for an intracortical electrode array. *IEEE Transactions on Biomedical Engineering*, 38(8):758-768.
- Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 8:679-698.
- Canny, J. and Reif, J. (1987). New lower bound techniques for robot motion planning problems. In *IEEE FOCS*, pages 39-48.
- Canny, J. F. (1988). *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, Massachusetts.
- Carbonell, J. R. and Collins, A. M. (1973). Natural semantics in artificial intelligence. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, Stanford, California. IJCAII.
- Carnap, R. (1948). On the application of inductive logic. *Philosophy and Phenomenological Research*, 8:133-148.
- Carnap, R. (1950). *Logical Foundations of Probability*. University of Chicago Press, Chicago, Illinois.
- Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1023-1028, Seattle, Washington. AAAI Press.
- Chakrabarti, P. P., Ghose, S., Acharya, A., and de Sarkar, S. C. (1989). Heuristic search in restricted memory. *Artificial Intelligence*, 41(2):197-122.
- Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York.
- Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333-377.
- Chapuis, A. and Droz, E. (1958). *Automata: A Historical and Technological Study*. Editions du Griffon, Neufchatel, Switzerland.
- Charniak, E. (1972). *Toward a Model of Children's Story Comprehension*. PhD thesis, Massachusetts Institute of Technology.
- Charniak, E. (1993). *Statistical Language Learning*. MIT Press.
- Charniak, E. and Goldman, R. P. (1992). A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53-79.

- Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts.
- Charniak, E., Riesbeck, C., McDermott, D., and Meehan, J. (1987). *Artificial Intelligence Programming*. Lawrence Erlbaum Associates, Potomac, Maryland, second edition.
- Cheeseman, P. (1985). In defense of probability. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 1002–1009, Los Angeles, California. Morgan Kaufmann.
- Cheeseman, P. (1988). An inquiry into computer understanding. *Computational Intelligence*, 4(1):58–66.
- Cheeseman, P., Self, M., Kelly, J., and Stutz, J. (1988). Bayesian classification. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, volume 2, pages 607–611, St. Paul, Minnesota. Morgan Kaufmann.
- Chellas, B. F. (1980). *Modal Logic: An Introduction*. Cambridge University Press, Cambridge.
- Cherniak, C. (1986). *Minimal Rationality*. MIT Press, Cambridge, Massachusetts.
- Chierchia, G. and McConnell-Ginet, S. (1990). *Meaning and Grammar*. MIT Press.
- Chitrao, M. and Grishman, R. (1990). Statistical parsing of messages. In *Proceedings of DARPA Speech and Natural Language Processing*. Morgan Kaufman: New York.
- Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.
- Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague and Paris.
- Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge, Massachusetts.
- Chomsky, N. (1980). Rules and representations. *The Behavioral and Brain Sciences*, 3:1–61.
- Chung, K. L. (1979). *Elementary Probability Theory with Stochastic Processes*. Springer-Verlag, Berlin, third edition.
- Church, A. (1936). A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1:40–41 and 101–102.
- Church, A. (1941). *The Calculi of Lambda-Conversion*. Princeton University Press, Princeton, New Jersey.
- Church, K. (1988). A stochastic parts program and noun phrase parser for unrestricted texts. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, Texas.
- Church, K. and Patil, R. (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics*, 8(3–4):139–149.
- Churchland, P. M. (1979). *Scientific Realism and the Plasticity of Mind*. Cambridge University Press, Cambridge.
- Churchland, P. M. and Churchland, P. S. (1982). Functionalism, qualia, and intentionality. In Biro, J. I. and Shaham, R. W., editors, *Mind, Brain and Function: Essays in the Philosophy of Mind*, pages 121–145. University of Oklahoma Press, Norman, Oklahoma.
- Churchland, P. S. (1986). *Neurophilosophy: Toward a Unified Science of the Mind-Brain*. MIT Press, Cambridge, Massachusetts.
- Clark, K. L. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 293–322. Plenum, New York.
- Clark, K. L. and Gregory, S. (1986). PARLOG: parallel programming in logic. *ACM Transactions on Programming Languages*, 8:1–49.
- Clark, R. (1992). The selection of syntactic knowledge. *Language Acquisition*, 2(2):83–149.
- Clarke, M. R. B., editor (1977). *Advances in Computer Chess I*. Edinburgh University Press, Edinburgh, Scotland.
- Clocks, W. F. and Mellish, C. S. (1987). *Programming in Prolog*. Springer-Verlag, Berlin, third revised and extended edition.
- Clowes, M. B. (1971). On seeing things. *Artificial Intelligence*, 2(1):79–116.
- Cobham, A. (1964). The intrinsic computational difficulty of functions. In Bar-Hillel, Y., editor, *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30. Elsevier/North-Holland.
- Cohen, J. (1966). *Human Robots in Myth and Science*. Alien and Unwin, London.

- Cohen, J. (1988). A view of the origins and development of PROLOG. *Communications of the Association for Computing Machinery*, 31:26–36.
- Cohen, P., Morgan, J., and Pollack, M. (1990). *Intentions in Communication*. MIT Press.
- Cohen, P. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3):177–212.
- Cohen, P. R. and Feigenbaum, E. A., editors (1982). *The Handbook of Artificial Intelligence*, volume 3: HeurisTech Press and William Kaufmann, Stanford, California and Los Altos, California.
- Colmerauer, A. (1975). Les grammaires de métamorphose. Technical report, Groupe d'Intelligence Artificielle, Université de Marseille-Luminy. Translated into English as Colmerauer (1978).
- Colmerauer, A. (1978). Metamorphosis grammars. In Bolc, L., editor, *Natural Language Communication with Computers*. Springer-Verlag, Berlin. English translation of Colmerauer (1975).
- Colmerauer, A. (1985). Prolog in 10 figures. *Communications of the Association for Computing Machinery*, 28(12):1296–1310.
- Colmerauer, A. (1990). Prolog III as it actually is. In Warren, D. H. D. and Szeredi, P., editors, *Logic Programming: Proceedings of the Seventh International Conference*, page 766, Jerusalem. MIT Press.
- Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P. (1973). Un système de communication homme-machine en Français. Rapport, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II.
- Colomb, R. M. (1991). Enhancing unification in PROLOG through clause indexing. *Journal of Logic Programming*, 10(1):23–44.
- Condon, E. U., Tawney, G. L., and Derr, W. A. (1940). Machine to play game of Nim. U.S. Patent 2,215,544, United States Patent Office, Washington, D.C.
- Condon, J. H. and Thompson, K. (1982). Belle chess hardware. In Clarke, M. R. B., editor, *Advances in Computer Chess 3*, pages 45–54. Pergamon, New York.
- Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, New York.
- Cooper, G. and Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347.
- Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405.
- Copeland, J. (1993). *Artificial Intelligence: A Philosophical Introduction*. Blackwell, Oxford.
- Cormen, T. H., Leiserson, C. E., and Rivest, R. R. (1990). *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts.
- Covington, M. A. (1994). *Natural Language Processing for Prolog Programmers*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Cowan, J. D. and Sharp, D. H. (1988a). Neural nets. *Quarterly Reviews of Biophysics*, 21:365–427.
- Cowan, J. D. and Sharp, D. H. (1988b). Neural nets and artificial intelligence. *Daedalus*, 117:85–121.
- Cox, R. T. (1946). Probability, frequency, and reasonable expectation. *American Journal of Physics*, 14(1):1–13.
- Cragg, B. G. and Temperley, H. N. V. (1954). The organization of neurones: A cooperative analogy. *EEG and Clinical Neurophysiology*, 6:85–92.
- Cragg, B. G. and Temperley, H. N. V. (1955). Memory: The analogy with ferromagnetic hysteresis. *Brain*, 78(II):304–316.
- Craik, K. J. W. (1943). *The Nature of Explanation*. Cambridge University Press, Cambridge.
- Crevier, D. (1993). *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, New York.
- Crockett, L. (1994). *The Turing Test and the Frame Problem: AI's Mistaken Understanding of Intelligence*. Ablex, Norwood, New Jersey.
- Cullingford, R. E. (1981). Integrating knowledge sources for computer ‘understanding’ tasks. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11.
- Currie, K. W. and Tate, A. (1991). O-Plan: the Open Planning Architecture. *Artificial Intelligence*, 52(1):49–86.

- Curry, H. B. and Feys, R. (1958). *Combinatory Logic*, volume 1. Elsevier/North-Holland, Amsterdam, London, New York.
- Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, Massachusetts.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Controls, Signals, and Systems*, 2:303–314.
- Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1): 141–153.
- Dahl, O.-J., Myrhaug, B., and Nygaard, K. (1970). (Simula 67) common base language. Technical Report N. S-22, Norsk Regnesentral (Norwegian Computing Center), Oslo.
- Dantzig, G. B. (1960). On the significance of solving linear programming problems with some integer variables. *Econometrica*, 28:30–44.
- Darwiche, A. Y. and Ginsberg, M. L. (1992). A symbolic generalization of probability theory. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 622–627, San Jose, California. AAAI Press.
- Davidson, D. (1980). *Essays on Actions and Events*. Oxford University Press, Oxford.
- Davies, T. (1985). Analogy. Informal Note IN-CSLI-85-4, Center for the Study of Language and Information (CSLI), Stanford, California.
- Davies, T. R. and Russell, S. J. (1987). A logical approach to reasoning by analogy. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, volume 1, pages 264–270, Milan, Italy. Morgan Kaufmann.
- Davis, E. (1986). *Representing and Acquiring Geographic Knowledge*. Pitman and Morgan Kaufmann, London and San Mateo, California.
- Davis, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann, San Mateo, California.
- Davis, M. (1957). A computer program for Presburger's algorithm. In Robinson, A., editor, *Proving Theorems, (as Done by Man, Logician, or Machine)*, pages 215–233, Cornell University, Ithaca, New York. Communications Research Division, Institute for Defense Analysis. Summaries of Talks Presented at the 1957 Summer Institute for Symbolic Logic. Second edition; publication date is 1960.
- Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7(3):201–215.
- Davis, R. (1980). Meta-rules: reasoning about control. *Artificial Intelligence*, 15(3):179–222.
- Davis, R. and Lenat, D. B. (1982). *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill, New York.
- Dayan, P. (1992). The convergence of TDA for general A. *Machine Learning*, 8(3–4):341–362.
- de Dombal, F. T., Leaper, D. J., Horrocks, J. C., and Staniland, J. R. (1974). Human and computer-aided diagnosis of abdominal pain: Further report with emphasis on performance of clinicians. *British Medical Journal*, 1:376–380.
- de Dombal, F. T., Staniland, J. R., and Clamp, S. E. (1981). Geographical variation in disease presentation. *Medical Decision Making*, 1:59–69.
- de Finetti, B. (1937a). Foresight: Its logical laws, its subjective sources. In Kyburg, H. E. and Smokier, H. E., editors, *Studies in Subjective Probability*, pages 55–118. Krieger, New York.
- de Finetti, B. (1937b). Le prévision: ses lois logiques, ses sources subjectives. *Ann. Inst. Poincaré*, 7:1–68. Translated into English as DeFinetti (1937a).
- de Groot, A. D. (1946). *Het Denken van den Schaker*. Elsevier/North-Holland, Amsterdam, London, New York. Translated as DeGroot (1978).
- de Groot, A. D. (1978). *Thought and Choice in Chess*. Mouton, The Hague and Paris, second edition.
- de Kleer, J. (1975). Qualitative and quantitative knowledge in classical mechanics. Technical Report AI-TR-352, MIT Artificial Intelligence Laboratory.
- de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, 28(2): 127–162.
- de Kleer, J. (1986a). Extending the ATMS. *Artificial Intelligence*, 28(2): 163–196.
- de Kleer, J. (1986b). Problem solving with the ATMS. *Artificial Intelligence*, 28(2): 197–224.

- de Kleer, J. and Brown, J. S. (1985). A qualitative physics based on confluences. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, chapter 4, pages 109-183. Ablex, Norwood, New Jersey.
- de Kleer, J., Doyle, J., Steele, G. L., and Sussman, G. J. (1977). AMORD: explicit control of reasoning. *SIGPLAN Notices*, 12(8):116-125.
- De Morgan, A. (1864). On the syllogism IV and on the logic of relations. *Cambridge Philosophical Transactions*, x:331-358.
- De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press, New York.
- Dean, T. and Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 49-54, St. Paul, Minnesota. Morgan Kaufmann.
- Dean, T., Firby, J., and Miller, D. (1990). Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 6(1).
- Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. (1993). Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 574-579, Washington, D.C. AAAIPress.
- Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142-150.
- Dean, T. L. and Wellman, M. P. (1991). *Planning and Control*. Morgan Kaufmann, San Mateo, California.
- Debreu, G. (1960). Topological methods in cardinal utility theory. In Arrow, K. J., Karlin, S., and Suppes, P., editors, *Mathematical Methods in the Social Sciences, 1959*. Stanford University Press, Stanford, California.
- Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery*, 32(3):505-536.
- DeGroot, M. H. (1970). *Optimal Statistical Decisions*. McGraw-Hill, New York.
- DeGroot, M. H. (1989). *Probability and Statistics*. Addison-Wesley, Reading, Massachusetts, second edition. Reprinted with corrections.
- DeJong, G. (1981). Generalizations based on explanations. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 67-69, Vancouver, British Columbia. Morgan Kaufmann.
- DeJong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1:145-176.
- Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39 (Series B): 1-38.
- Dempster, A. P. (1968). A generalization of Bayesian inference. *Journal of the Royal Statistical Society*, 30 (Series B):205-247.
- Dennett, D. C. (1969). *Content and Consciousness*. Routledge and Kegan Paul, London.
- Dennett, D. C. (1971). Intentional systems. *The Journal of Philosophy*, 68(4):87-106.
- Dennett, D. C. (1978a). *Brainstorms: Philosophical Essays on Mind and Psychology*. MIT Press, Cambridge, Massachusetts, first edition.
- Dennett, D. C. (1978b). Why you can't make a computer that feels pain. *Synthese*, 38(3).
- Dennett, D. C. (1984). Cognitive wheels: the frame problem of AI. In Hookway, C., editor, *Minds, Machines, and Evolution: Philosophical Studies*, pages 129-151. Cambridge University Press, Cambridge.
- Dennett, D. C. (1986). The moral first aid manual. Tanner lectures on human values, University of Michigan.
- Deo, N. and Pang, C. (1982). Shortest path algorithms: Taxonomy and annotation. Technical Report CS-80-057, Computer Science Department, Washington State University.
- Descotte, Y. and Latombe, J. C. (1985). Making compromises among antagonist constraints in a planner. *Artificial Intelligence*, 27:183-217.
- Devanbu, P., Brachman, R. J., Selfridge, P. G., and Ballard, B. W. (1991). LaSSIE: a knowledge-based software information system. *Communications of the Association for Computing Machinery*, 34(5):34-49.

- Dickmanns, E. D. and Zapp, A. (1987). Autonomous high speed road vehicle guidance by computer vision. In Isermann, R., editor, *Automatic Control—World Congress, 1987: Selected Papers from the 10th Triennial World Congress of the International Federation of Automatic Control*, pages 221–226, Munich, Germany. Pergamon.
- Dietterich, T. G. (1990). Machine learning. *Annual Review of Computer Science*, 4.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.
- Dinebas, M. and LePape, J.-P. (1984). Metacontrol of logic programs in METALOG. In *Proceedings of the International Conference on Fifth-Generation Computer Systems*, Tokyo. Elsevier/North-Holland.
- Dingwell, W. O. (1988). The evolution of human communicative behavior. In Newmeyer, F. J., editor, *Linguistics: The Cambridge Survey, Vol. III*, pages 274–313. Cambridge University Press.
- Doran, J. and Michie, D. (1966). Experiments with the graph traverser program. *Proceedings of the Royal Society of London*, 294, Series A:235–259.
- Dowty, D., Wall, R., and Peters, S. (1991). *Introduction to Montague Semantics*. D. Reidel, Dordrecht, The Netherlands.
- Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, 12(3):231–272. Reprinted in Webber and Nilsson (1981).
- Doyle, J. (1980). A model for deliberation, action, and introspection. Technical Report AI-TR-581, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts. Republished PhD dissertation.
- Doyle, J. (1983). What is rational psychology? Toward a modern mental philosophy. *AI Magazine*, 4(3):50–53.
- Doyle, J. and Patil, R. S. (1991). Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48(3):261–297.
- Drabble, B. (1990). Mission scheduling for space-craft: Diaries of T-SCHED. In *Expert Planning Systems*, pages 76–81. Institute of Electrical Engineers.
- Draper, D., Hanks, S., and Weld, D. (1994). Probabilistic planning with information gathering and contingent execution. In *Proceedings 2nd AIPS*, San Mateo, California. Morgan Kaufmann.
- Dreyfus, H. L. (1972). *What Computers Can't Do: A Critique of Artificial Reason*. Harper and Row, New York, first edition.
- Dreyfus, H. L. (1979). *What Computers Can't Do: The Limits of Artificial Intelligence*. Harper and Row, New York, revised edition.
- Dreyfus, H. L. (1992). *What Computers Still Can't Do: A Critique of Artificial Reason*. MIT Press, Cambridge, Massachusetts.
- Dreyfus, H. L. and Dreyfus, S. E. (1986). *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. Blackwell, Oxford. With Tom Athanasiou.
- Dreyfus, S. E. (1969). An appraisal of some shortest-paths algorithms. *Operations Research*, 17:395–412.
- Dubois, D. and Prade, H. (1994). A survey of belief revision and updating rules in various uncertainty models. *International Journal of Intelligent Systems*, 9(1):61–100.
- Duda, R., Gaschnig, J., and Hart, P. (1979). Model design in the Prospector consultant system for mineral exploration. In Michie, D., editor, *Expert Systems in the Microelectronic Age*, pages 153–167. Edinburgh University Press, Edinburgh, Scotland.
- Dyer, M. (1983). *In-Depth Understanding*. MIT Press, Cambridge, Massachusetts.
- Dzeroski, S., Muggleton, S., and Russell, S. I. (1992). PAC-learnability of determinate logic programs. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92)*, Pittsburgh, Pennsylvania. ACM Press.
- Earley, I. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102.
- Ebeling, C. (1987). *All the Right Moves*. MIT Press, Cambridge, Massachusetts.
- Edmonds, J. (1962). Covers and packings in a family of sets. *Bulletin of the American Mathematical Society*, 68:494–499.
- Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467.

- Edwards, P., editor (1967). *The Encyclopedia of Philosophy*. Macmillan, London.
- Elkan, C. (1992). Reasoning about action in first-order logic. In *Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence*, Vancouver, British Columbia.
- Elkan, C. (1993). The paradoxical success of fuzzy logic. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 698–703, Washington, D.C. AAAI Press.
- Empson, W. (1953). *Seven Types of Ambiguity*. New Directions.
- Enderton, H. B. (1972). *A Mathematical Introduction to Logic*. Academic Press, New York.
- Engelberger, J. F. (1980). *Robotics in Practice*. Amacom, New York.
- Engelberger, J. F. (1989). *Robotics in Service*. MIT Press, Cambridge, Massachusetts.
- Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R. (1980). The HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253. Reprinted in Webber and Nilsson (1981).
- Ernst, H. A. (1961). *MH-1, a Computer-Operated Mechanical Hand*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.
- Erol, K., Hendler, J., and Nau, D. S. (1994). HTN planning: complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Washington. AAAI Press.
- Etzioni, O. (1989). Tractable decision-analytic control. In *Proc. of 1st International Conference on Knowledge Representation and Reasoning*, pages 114–125, Toronto, Ontario.
- Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., and Williamson, M. (1992). An approach to planning with incomplete information. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*.
- Evans, T. G. (1968). A program for the solution of a class of geometric-analogy intelligence-test questions. In Minsky, M. L., editor, *Semantic Information Processing*, pages 271–353. MIT Press, Cambridge, Massachusetts.
- Fahlman, S. E. (1974). A planning system for robot construction tasks. *Artificial Intelligence*, 5(1):1–49.
- Fahlman, S. E. (1979). *NETL: A System for Representing and Using Real-World Knowledge*. MIT Press, Cambridge, Massachusetts.
- Farhat, N. H., Psaltis, D., Prata, A., and Paek, E. (1985). Optical implementation of the Hopfield model. *Applied Optics*, 24:1469–1475. Reprinted in Anderson and Rosenfeld (1988).
- Faugeras, O. (1993). *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, Massachusetts.
- Feigenbaum, E. and Shrobe, H. (1993). The Japanese national fifth generation project: introduction, survey, and evaluation. *Future Generation Computer Systems*, 9(2):105–117.
- Feigenbaum, E. A. (1961). The simulation of verbal learning behavior. *Proceedings of the Western Joint Computer Conference*, 19:121–131. Reprinted in (Feigenbaum and Feldman, 1963, pp. 297–309).
- Feigenbaum, E. A., Buchanan, B. G., and Ledberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 6*, pages 165–190. Edinburgh University Press, Edinburgh, Scotland.
- Feigenbaum, E. A. and Feldman, J., editors (1963). *Computers and Thought*. McGraw-Hill, New York.
- Feldman, J. A. and Sproull, R. F. (1977). Decision theory and artificial intelligence II: The hungry monkey. Technical report, Computer Science Department, University of Rochester.
- Feldman, J. A. and Yakimovsky, Y. (1974). Decision theory and artificial intelligence I: Semantics-based region analyzer. *Artificial Intelligence*, 5(4):349–371.
- Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288.
- Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208.
- Fikes, R. E. and Nilsson, N. J. (1993). STRIPS, a retrospective. *Artificial Intelligence*, 59(1–2):227–232.

- Findlay, J. N. (1941). Time: A treatment of some puzzles. *Australasian Journal of Psychology and Philosophy*, 19(3):216–235.
- Fischer, M. J. and Ladner, R. E. (1977). Propositional modal logic of programs. In *Proceedings of the 9th ACM Symposium on the Theory of Computing*, pages 286–294.
- Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London*, Series A 222:309–368.
- Floyd, R. W. (1962a). Algorithm 96: Ancestor. *Communications of the Association for Computing Machinery*, 5:344–345.
- Floyd, R. W. (1962b). Algorithm 97: Shortest path. *Communications of the Association for Computing Machinery*, 5:345.
- Fodor, J. A. (1980). Searle on what only brains can do. *Behavioral and Brain Sciences*, 3:431–432. Peer commentary on Searle (1980).
- Fodor, J. A. (1983). *The Modularity of Mind: An Essay on Faculty Psychology*. MIT Press, Cambridge, Massachusetts.
- Forbus, K. D. (1985). The role of qualitative dynamics in naive physics. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, chapter 5, pages 185–226. Ablex, Norwood, New Jersey.
- Forbus, K. D. and de Kleer, J. (1993). *Building Problem Solvers*. MIT Press, Cambridge, Massachusetts.
- Forsyth, D. and Zisserman, A. (1991). Reflections on shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 13(7):671–679.
- Fox, M. S. (1990). Constraint-guided scheduling: a short history of research at CMU. *Computers in Industry*, 14(1-3):79–88.
- Fox, M. S., Alien, B., and Strohm, G. (1981). Job shop scheduling: an investigation in constraint-based reasoning. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, Vancouver, British Columbia. Morgan Kaufmann.
- Fox, M. S. and Smith, S. F. (1984). Isis: a knowledge-based system for factory scheduling. *Expert Systems*, 1(1):25–49.
- Frean, M. (1990). The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209.
- Frege, G. (1879). *Begriffsschrifteine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, Berlin. Reprinted in English translation in van Heijenoort (1967).
- Friedberg, R., Dunham, B., and North, T. (1959). A learning machine: Part II. *IBM Journal of Research and Development*, 3(3):282–287.
- Friedberg, R. M. (1958). A learning machine: Part I. *IBM Journal*, 2:2–13.
- Fu, K.-S. and Booth, T. L. (1986a). Grammatical inference: Introduction and survey—part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3):343–359. Reprinted from *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-5, No. 1, January 1975.
- Fu, K.-S. and Booth, T. L. (1986b). Grammatical inference: Introduction and survey—part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3):360–375. Reprinted from *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-5, No. 4, January 1975.
- Fuchs, J. J., Gasquet, A., Olalainy, B., and Currie, K. W. (1990). PlanERS-1: An expert planning system for generating spacecraft mission plans. In *First International Conference on Expert Planning Systems*, pages 70–75, Brighton, United Kingdom. Institute of Electrical Engineers.
- Fung, R. and Chang, K. C. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, Windsor, Ontario. Morgan Kaufmann.
- Furukawa, K. (1992). Summary of basic research activities of the FGCS project. In *Fifth Generation Computer Systems 1992*, volume 1, pages 20–32, Tokyo. IOS Press.
- Furuta, K., Ochiai, T., and Ono, N. (1984). Attitude control of a triple inverted pendulum. *International Journal of Control*, 39(6):1351–1365.
- Gabbay, D. M. (1991). Abduction in labelled deductive systems: A conceptual abstract. In Kruse, R. and Siegel, P., editors, *Symbolic and Quantitative Approaches to Uncertainty: Proceedings of European Conference ECSQAU*, pages 3–11. Springer-Verlag.

- Gallaire, H. and Minker, J., editors (1978). *Logic and Databases*. Plenum, New York.
- Gallier, J. H. (1986). *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper and Row, New York.
- Gamba, A., Gamberini, L., Palmieri, G., and Sanna, R. (1961). Further experiments with PAPA. *Nuovo Cimento Supplemento*, 20(2):221-231.
- Garding, J. (1992). Shape from texture for smooth curved surfaces in perspective projection. *Journal of Mathematical Imaging and Vision*, 2(4):327-350.
- Gardner, M. (1968). *Logic Machines, Diagrams and Boolean Algebra*. Dover, New York.
- Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability*. W. H. Freeman, New York.
- Garside, R., Leech, F., and Sampson, G., editors (1987). *The Computational Analysis of English*. Longman.
- Gaschnig, J. (1979). Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Computer Science Department, Carnegie-Mellon University.
- Gauss, K. F. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Sumtibus F. Perthes et I. H. Besser, Hamburg.
- Gazdar, G. (1989). COMIT \Rightarrow^* PATR. In Wilks, Y., editor, *Theoretical Issues in Natural Language Processing*, Potomac, Maryland. Lawrence Erlbaum Associates.
- Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar*. Blackwell, Oxford.
- Geffner, H. (1992). *Default Reasoning: Causal and Conditional Theories*. MIT Press, Cambridge, Massachusetts.
- Gelb, A. (1974). *Applied Optimal Estimation*. MIT Press, Cambridge, Massachusetts.
- Gelernter, H. (1959). Realization of a geometry-theorem proving machine. In *Proceedings of an International Conference on Information Processing*, pages 273-282, Paris. UNESCO House.
- Gelernter, H., Hansen, J. R., and Gerberich, C. L. (1960). A FORTRAN-compiled list processing language. *Journal of the Association for Computing Machinery*, 7(2):87-101.
- Gelfond, M. and Lifschitz, V. (1988). Compiling circumscriptive theories into logic programs. In Reinfrank, M., de Kleer, J., Ginsberg, M. L., and Sandewall, E., editors, *Non-Monotonic Reasoning: 2nd International Workshop Proceedings*, pages 74-99, Grassau, Germany. Springer-Verlag.
- Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365-385.
- Genesereth, M. R. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24(1-3):411-436.
- Genesereth, M. R. and Ketchpel, S. P. (1994). Software agents. *Communications of the Association for Computing Machinery*, 37(7).
- Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.
- Genesereth, M. R. and Smith, D. (1981). Meta-level architecture. Memo HPP-81-6, Computer Science Department, Stanford University, Stanford, California.
- Gentner, D. (1983). Structure mapping: A theoretical framework for analogy. *Cognitive Science*, 7:155-170.
- Gentzen, G. (1934). Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176-210, 405-431.
- Georgeff, M. P. and Lansky, A. L., editors (1986). *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Timberline, Oregon. Morgan Kaufmann.
- Gibson, J. J. (1950). *The Perception of the Visual World*. Houghton Mifflin, Boston, Massachusetts.
- Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston, Massachusetts.
- Gibson, J. J., Olum, P., and Rosenblatt, F. (1955). Parallax and perspective during aircraft landings. *American Journal of Psychology*, 68:372-385.
- Gilmore, P. C. (1960). A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development*, 4:28-35.
- Ginsberg, M. (1993). *Essentials of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.

- Ginsberg, M. L., editor (1987). *Readings in Non-monotonic Reasoning*. Morgan Kaufmann, San Mateo, California.
- Ginsberg, M. L. (1989). Universal planning: An (almost) universally bad idea. *AIMagazine*, 10(4):40-44.
- Giralt, G., Alami, R., Chatila, R., and Freedman, P. (1991). Remote operated autonomous robots. In *Intelligent Robotics: Proceedings of the International Symposium*, volume 1571, pages 416-427, Bangalore, India. International Society for Optical Engineering (SPIE).
- Glanc, A. (1978). On the etymology of the word "robot". *SIGART Newsletter*, 67:12.
- Glover, F. (1989). Tabu search: 1. *ORSA Journal on Computing*, 1(3):190-206.
- Godel, K. (1930). *Über die Vollständigkeit des Logikkalküls*. PhD thesis, University of Vienna.
- Godel, K. (1931). Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I. *Monatsheft für Mathematik und Physik*, 38:173-198.
- Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10:447-474.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, Massachusetts.
- Goldman, R. P. and Charniak, E. (1992). Probabilistic text understanding. *Statistics and Computing*, 2(2):105-114.
- Goldszmidt, M., Morris, P., and Pearl, J. (1990). A maximum entropy approach to nonmonotonic reasoning. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, volume 2, pages 646-652, Boston, Massachusetts. MIT Press.
- Good, I. J. (1950). Contribution to the discussion of Eliot Slater's "Statistics for the chess computer and the factor of mobility". In *Symposium on Information Theory*, page 199, London. Ministry of Supply.
- Good, I. J. (1961). A causal calculus. *British Journal of the Philosophy of Science*, 11:305-318. Reprinted in Good (1983).
- Good, I. J. (1983). *Good Thinking: The Foundations of Probability and Its Applications*. University of Minnesota Press, Minneapolis, Minnesota.
- Goodman, N. (1954). *Fact, Fiction and Forecast*. University of London Press, London, first edition.
- Goodman, N. (1977). *The Structure of Appearance*. D. Reidel, Dordrecht, The Netherlands, third edition.
- Gorry, G. A. (1968). Strategies for computer-aided diagnosis. *Mathematical Biosciences*, 2(3-4):293-318.
- Gorry, G. A., Kassirer, J. P., Essig, A., and Schwartz, W. B. (1973). Decision analysis as the basis for computer-aided management of acute renal failure. *American Journal of Medicine*, 55:473-484.
- Gould, S. J. (1994). This view of life. *Natural History*, 8:10-17.
- Graham, S. L., Harrison, M. A., and Ruzzo, W. L. (1980). An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415-462.
- Grayson, C. I. (1960). Decisions under uncertainty: Drilling decisions by oil and gas operators. Technical report, Division of Research, Harvard Business School, Boston.
- Green, C. (1969a). Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 219-239, Washington, D.C. IJCAII.
- Green, C. (1969b). Theorem-proving by resolution as a basis for question-answering systems. In Meltzer, B., Michie, D., and Swann, M., editors, *Machine Intelligence 4*, pages 183-205. Edinburgh University Press, Edinburgh, Scotland.
- Greenblatt, R. D., Eastlake, D. E., and Crocker, S. D. (1967). The Greenblatt chess program. In *Proceedings of the Fall Joint Computer Conference*, pages 801-810.
- Greiner, R. (1989). Towards a formal analysis of EBL. In *Proceedings of the Sixth International Machine Learning Workshop*, Ithaca, NY. Morgan Kaufmann.
- Grice, H. P. (1957). Meaning. *Philosophical Review*, 66:377-388.
- Grimes, J. (1975). *The Thread of Discourse*. Moulton.

- Grosz, B., Appelt, D., Martin, P., and Pereira, F. (1987). Team: An experiment in the design of transportable natural-language interfaces. *Artificial Intelligence*, 32(2): 173-244.
- Grosz, B. J. and Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175-204.
- Grosz, B. J., Sparck Jones, K., and Webber, B. L., editors (1986). *Readings in Natural Language Processing*. Morgan Kaufmann, San Mateo, California.
- Gu, J. (1989). *Parallel Algorithms and Architectures for Very Fast AI Search*. PhD thesis, University of Utah.
- Guard, J., Oglesby, F., Bennett, J., and Settle, L. (1969). Semi-automated mathematics. *Journal of the Association for Computing Machinery*, 16:49-62.
- Haas, A. (1986). A syntactic theory of belief and action. *Artificial Intelligence*, 28(3):245-292.
- Hacking, I. (1975). *The Emergence of Probability*. Cambridge University Press, Cambridge.
- Hald, A. (1990). *A History of Probability and Statistics and Their Applications Before 1750*. Wiley, New York.
- Halpern, J. Y. (1987). Using reasoning about knowledge to analyze distributed systems. In Traub, J. R., Grosz, B. J., Lampson, B. W., and Nilsson, N. J., editors, *Annual review of computer science*, volume 2, pages 37-68. Annual Reviews, Palo Alto.
- Hamming, R. W. (1991). *The Art of Probability for Scientists and Engineers*. Addison-Wesley, Reading, Massachusetts.
- Hammond, K. (1989). *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, New York.
- Hanks, S., Russell, S., and Wellman, M., editors (1994). *Proceedings of the AAAI Spring Symposium on Decision-Theoretic Planning*, Stanford, California.
- Hanski, I. and Cambefort, Y., editors (1991). *Dung Beetle Ecology*. Princeton University Press, Princeton, New Jersey.
- Hansson, O. and Mayer, A. (1989). Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Ontario. Morgan Kaufmann.
- Haralick, R. and Elliot, G. (1980). Increasing tree search efficiency for constraint-satisfaction problems. *Artificial Intelligence*, 14(3):263-313.
- Harel, D. (1984). Dynamic logic. In Gabbay, D. and Guenther, R., editors, *Handbook of Philosophical Logic*, volume 2, pages 497-604. D. Reidel, Dordrecht, The Netherlands.
- Harkness, K. and Battell, J. S. (1947). This made chess history. *Chess Review*.
- Harman, G. H. (1983). *Change in View: Principles of Reasoning*. MIT Press, Cambridge, Massachusetts.
- Harp, S. A., Samad, T., and Guha, A. (1990). Designing application-specific neural networks using the genetic algorithm. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems II*, pages 447-454. Morgan Kaufmann, San Mateo, California.
- Harris, L. R. (1984). Experience with INTELLECT: Artificial intelligence technology transfer. *AI Magazine*, 5(2, Summer):43-55.
- Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100-107.
- Hart, P. E., Nilsson, N. J., and Raphael, B. (1972). Correction to "A formal basis for the heuristic determination of minimum cost paths". *SIGART Newsletter*, 37:28-29.
- Hart, T. P. and Edwards, D. J. (1961). The tree prune (TP) algorithm. Artificial Intelligence Project Memo 30, Massachusetts Institute of Technology, Cambridge, Massachusetts.
- Haugeland, J., editor (1981). *Mind Design*. MIT Press, Cambridge, Massachusetts.
- Haugeland, J., editor (1985). *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, Massachusetts.
- Haussler, D. (1989). Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1):7-40.
- Hawkins, J. (1961). Self-organizing systems: A review and commentary. *Proceedings of the IRE*, 49(1):31-48.

- Hayes, J. E. and Levy, D. N. L. (1976). *The World Computer Chess Championship: Stockholm 1974*. Edinburgh University Press, Edinburgh, Scotland.
- Hayes, P. J. (1973). Computation and deduction. In *Proceedings of the Second Symposium on Mathematical Foundations of Computer Science*, Czechoslovakia. Czechoslovakian Academy of Science.
- Hayes, P. J. (1978). The naive physics manifesto. In Michie, D., editor, *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh, Scotland.
- Hayes, P. J. (1979). The logic of frames. In Metzing, D., editor, *Frame Conceptions and Text Understanding*, pages 46–61. de Gruyter, Berlin.
- Hayes, P. J. (1985a). Naive physics I: ontology for liquids. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, chapter 3, pages 71–107. Ablex, Norwood, New Jersey.
- Hayes, P. J. (1985b). The second naive physics manifesto. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, chapter 1, pages 1–36. Ablex, Norwood, New Jersey.
- Hazan, M. (1973). *The Classic Italian Cookbook*. Ballantine.
- Hebb, D. O. (1949). *The Organization of Behavior*. Wiley, New York.
- Heckerman, D. (1986). Probabilistic interpretation for MYCIN's certainty factors. In Kanal, L. N. and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence*, pages 167–196. Elsevier/North-Holland, Amsterdam, London, New York.
- Heckerman, D. (1991). *Probabilistic Similarity Networks*. MIT Press, Cambridge, Massachusetts.
- Heckerman, D., Geiger, D., and Chickering, M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, Redmond, Washington.
- Held, M. and Karp, R. M. (1970). The traveling salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162.
- Helman, D. H., editor (1988). *Analogical Reasoning: Perspectives of Artificial Intelligence, Cognitive Science, and Philosophy*. Kluwer, Dordrecht, The Netherlands.
- Hendrix, G. G. (1975). Expanding the utility of semantic networks through partitioning. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 115–121, Tbilisi, Georgia. IJCAII.
- Henrion, M. (1988). Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F. and Kanal, L. N., editors, *Uncertainty in Artificial Intelligence 2*, pages 149–163. Elsevier/North-Holland, Amsterdam, London, New York.
- Heppenheimer, T. A. (1985). Man makes man. In Minsky, M., editor, *Robotics*, pages 28–69. Doubleday, Garden City, New York.
- Herbrand, J. (1930). *Recherches sur la Théorie de la Démonstration*. PhD thesis, University of Paris.
- Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Reading, Massachusetts.
- Hewitt, C. (1969). PLANNER: a language for proving theorems in robots. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 295–301, Washington, D.C. IJCAII.
- Hintikka, J. (1962). *Knowledge and Belief*. Cornell University Press, Ithaca, New York.
- Hinton, G. E. and Anderson, J. A. (1981). *Parallel Models of Associative Memory*. Lawrence Erlbaum Associates, Potomac, Maryland.
- Hinton, G. E. and Sejnowski, T. (1983). Optimal perceptual inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 448–453, Washington, D.C. IEEE Computer Society Press.
- Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, chapter 7, pages 282–317. MIT Press, Cambridge, Massachusetts.
- Hirsh, H. (1987). Explanation-based generalization in a logic programming environment. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milan, Italy. Morgan Kaufmann.
- Hirst, G. (1987). *Semantic Interpretation Against Ambiguity*. Cambridge University Press.

- Hobbs, J. R. (1985). Ontological promiscuity. In *Proceedings, 23rd Annual Meeting of the Association for Computational Linguistics*, pages 61-69, Chicago, Illinois.
- Hobbs, J. R. (1986). Overview of the TACITUS project. *Computational Linguistics*, 12(3):220-222.
- Hobbs, J. R. (1990). *Literature and Cognition*. CSLI Press, Stanford, California.
- Hobbs, J. R., Blenko, T., Croft, B., Hager, G., Kautz, H. A., Kube, P., and Shoham, Y. (1985). Commonsense summer: Final report. Technical Report CSLI-85-35, Center for the Study of Language and Information (CSLI), Stanford, California.
- Hobbs, J. R., Croft, W., Davies, T., Edwards, D. D., and Laws, K. I. (1987). Commonsense metaphysics and lexical semantics. *Computational Linguistics*, 13(3-4):241-250.
- Hobbs, J. R. and Moore, R. C., editors (1985). *Formal Theories of the Commonsense World*. Ablex, Norwood, New Jersey.
- Hobbs, J. R., Stickel, M., Appelt, D., and Martin, P. (1990). Interpretation as abduction. Technical Note 499, SRI International, Menlo Park, California.
- Hobbs, J. R., Stickel, M. E., Appelt, D. E., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence*, 63(1-2):69-142.
- Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press.
- Hopfield, J. J. (1982). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences (USA)*, 79:2554-2558.
- Horn, A. (1951). On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16:14-21.
- Horn, B. K. P. (1970). Shape from shading: a method for obtaining the shape of a smooth opaque object from one view. Technical Report 232, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts.
- Horn, B. K. P. (1986). *Robot Vision*. MIT Press, Cambridge, Massachusetts.
- Horn, B. K. P. and Brooks, M. J. (1989). *Shape from Shading*. MIT Press, Cambridge, Massachusetts.
- Horvitz, E. J., Breese, J. S., and Henrion, M. (1988). Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2:247-302.
- Horvitz, E. J. and Heckerman, D. (1986). The inconsistent use of measures of certainty in artificial intelligence research. In Kanal, L. N. and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence*, pages 137-151. Elsevier/North-Holland, Amsterdam, London, New York.
- Horvitz, E. J., Heckerman, D. E., and Langlotz, C. P. (1986). A framework for comparing alternative formalisms for plausible reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, volume 1, pages 210-214, Philadelphia, Pennsylvania. Morgan Kaufmann.
- Horvitz, E. J., Suermondt, H. J., and Cooper, G. F. (1989). Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, pages 182-193, Windsor, Ontario. Morgan Kaufmann.
- Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts.
- Howard, R. A. (1966). Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, SSC-2:22-26.
- Howard, R. A. (1977). Risk preference. In Howard, R. A. and Matheson, J. E., editors, *Readings in Decision Analysis*, pages 429-465. Decision Analysis Group, SRI International, Menlo Park, California.
- Howard, R. A. (1989). Microrisks for medical decision analysis. *International Journal of Technology Assessment in Health Care*, 5:357-370.
- Howard, R. A. and Matheson, J. E. (1984). Influence diagrams. In Howard, R. A. and Matheson, J. E., editors, *Readings on the Principles and Applications of Decision Analysis*, pages 721-762. Strategic Decisions Group, Menlo Park, California. Article dates from 1981.
- Hsu, F.-H., Anantharaman, T. S., Campbell, M. S., and Nowatzky, A. (1990). A grandmaster chess machine. *Scientific American*, 263(4):44-50.
- Hsu, K., Brady, D., and Psaltis, D. (1988). Experimental demonstration of optical neural computers. In Anderson, D. Z., editor, *Neural Information*

- Processing Systems, Denver 1987*, pages 377-386, Denver, Colorado. American Institute of Physics.
- Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S., and Weber, J. (1994). Automatic symbolic traffic scene analysis using belief networks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 966-972, Seattle, Washington. AAAI Press.
- Hubel, D. H. (1988). *Eye, Brain, and Vision*. W. H. Freeman, New York.
- Huddleston, R. D. (1988). *English Grammar: An Outline*. Cambridge University Press, Cambridge.
- Huffman, D. A. (1971). Impossible objects as non-sense sentences. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 6*, pages 295-324. Edinburgh University Press, Edinburgh, Scotland.
- Hughes, G. E. and Cresswell, M. J. (1968). *An Introduction to Modal Logic*. Methuen, London.
- Hughes, G. E. and Cresswell, M. J. (1984). *A Companion to Modal Logic*. Methuen, London.
- Hume, D. (1978). *A Treatise of Human Nature*. Oxford University Press, Oxford, second edition. Edited by E. A. Selby-Bigge and P. H. Nidditch.
- Hunt, E. B., Marin, J., and Stone, P. T. (1966). *Experiments in Induction*. Academic Press, New York.
- Hunter, G. (1971). *Metalogic: An Introduction to the Metatheory of Standard First-Order Logic*. University of California Press, Berkeley and Los Angeles.
- Hunter, E. and States, D. J. (1992). Bayesian classification of protein structure. *IEEE Expert*, 7(4):67-75.
- Huttenlocher, D. P. and Ullman, S. (1990). Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5(2): 195-212.
- Huygens, C. (1657). *Ratiocinii in ludo aleae*. In van Schooten, R, editor, *Exercitionum Mathematicorum*. Elsevirii, Amsterdam.
- Hwang, C. H. and Schubert, E. K. (1993). EE: a formal, yet natural, comprehensive knowledge representation. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 676-682, Washington, D.C. AAAI Press.
- Hyatt, R. M., Gower, A. E., and Nelson, H. E. (1986). Cray Blitz. In Beal, D. F., editor, *Advances in Computer Chess 4*, pages 8-18. Pergamon, New York.
- Ingerman, P. Z. (1967). Panini-Backus form suggested. *Communications of the Association for Computing Machinery*, 10(3):137.
- Jackson, P. (1986). *Introduction to Expert Systems*. Addison-Wesley, Reading, Massachusetts.
- Jacobs, P. and Rau, E. (1990). Scisor: A system for extracting information from on-line news. *Communications of the ACM*, 33(11):88-97.
- Jaffar, J. and Eassey, J.-E. (1987). Constraint logic programming. In *Proceedings of the Fourteenth ACM Conference on Principles of Programming Languages*, Munich. Association for Computing Machinery.
- Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. C. (1992a). The CEP(R) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339-395.
- Jaffar, J., Stuckey, P. J., Michaylov, S., and Yap, R. H. C. (1992b). An abstract machine for CEP(R). *SIGPLAN Notices*, 27(7): 128-139.
- Jáskowski, S. (1934). On the rules of suppositions in formal logic. *Studia Logica*, 1.
- Jeffrey, R. C. (1983). *The Logic of Decision*. University of Chicago Press, Chicago, Illinois, second edition.
- Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532-556.
- Jelinek, F. (1990). Self-organizing language modeling for speech recognition. In Waibel, A. and Lee, K.-F., editors, *Readings in Speech Recognition*, pages 450-506. Morgan Kaufmann.
- Jensen, F. V., Lauritzen, S. L., and Olesen, K. G. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 5(4):269-282.
- Jerison, H. J. (1991). *Brain Size and the Evolution of Mind*. American Museum of Natural History, New York.
- Jochum, T., Pomerleau, D., and Thorpe, C. (1993). Maniac: A next generation neurally based autonomous road follower. In *Proceedings of the International Conference on Intelligent Autonomous Systems: IAS-3*.

- Johnson, W. W. and Story, W. E. (1879). Notes on the "15" puzzle. *American Journal of Mathematics*, 2:397-404.
- Johnson-Laird, P. N. (1988). *The Computer and the Mind: An Introduction to Cognitive Science*. Harvard University Press, Cambridge, Massachusetts.
- Johnston, M. D. and Adorf, H.-M. (1992). Scheduling with neural networks: the case of the Hubble space telescope. *Computers & Operations Research*, 19(3-4):209-240.
- Jones, N. D., Gomard, C. K., and Sestoft, P. (1993). *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Joshi, A. (1985). Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions. In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural Language Parsing*. Cambridge University Press.
- Joshi, A., Webber, B., and Sag, I. (1981). *Elements of Discourse Understanding*. Cambridge University Press.
- Judd, J. S. (1990). *Neural Network Design and the Complexity of Learning*. MIT Press, Cambridge, Massachusetts.
- Julesz, B. (1971). *Foundations of Cyclopean Perception*. University of Chicago Press, Chicago, Illinois.
- Kaelbling, L. P. (1990). Learning functions in k -DNF from reinforcement. In *Machine Learning: Proceedings of the Seventh International Conference*, pages 162-169, Austin, Texas. Morgan Kaufmann.
- Kaelbling, L. P. and Rosenschein, S. J. (1990). Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6(1-2):35-48.
- Kahneman, D., Slovic, P., and Tversky, A., editors (1982). *Judgment under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge.
- Kaindl, H. (1990). Tree searching algorithms. In Marsland, A. T. and Schaeffer, J., editors, *Computers, Chess, and Cognition*, pages 133-158. Springer-Verlag, Berlin.
- Kaindl, H. and Khorsand, A. (1994). Memory-bounded bidirectional search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1359-1364, Seattle, Washington. AAAI Press.
- Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, pages 35-46.
- Kambhampati, S. and Nau, D. S. (1993). On the nature and role of modal truth criteria in planning. Technical Report ISR-TR-93-30, University of Maryland, Institute for Systems Research.
- Kanal, L. N. and Kumar, V. (1988). *Search in Artificial Intelligence*. Springer-Verlag, Berlin.
- Kanal, L. N. and Lemmer, J. F., editors (1986). *Uncertainty in Artificial Intelligence*. Elsevier/North-Holland, Amsterdam, London, New York.
- Kandel, E. R., Schwartz, J. H., and Jessell, T. M., editors (1991). *Principles of Neural Science*. Elsevier/North-Holland, Amsterdam, London, New York, third edition.
- Kaplan, D. and Montague, R. (1960). A paradox regained. *Notre Dame Journal of Formal Logic*, 1(3):79-90. Reprinted in Thomason (1974).
- Karger, D. R., Koller, D., and Phillips, S. J. (1993). Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22(6):1199-1217.
- Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85-103. Plenum, New York.
- Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts.
- Kautz, H. A. and Selman, B. (1991). Hard problems for simple default logics. *Artificial Intelligence*, 49(1-3):243-279.
- Kay, M., Gawron, J. M., and Norvig, P. (1994). *Verbmobil: A Translation System for Face-To-Face Dialog*. CSLI Press, Stanford, California.
- Kearns, M. J. (1990). *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, Massachusetts.
- Keeney, R. L. (1974). Multiplicative utility functions. *Operations Research*, 22:22-34.

- Keeney, R. L. and Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York.
- Kemp, M., editor (1989). *Leonardo on Painting: An Anthology of Writings*. Yale University Press, New Haven, Connecticut.
- Kemp, M. (1990). *The Science of Art: Optical Themes in Western Art from Brunelleschi to Seurat*. Yale University Press, New Haven, Connecticut.
- Keynes, J. M. (1921). *A Treatise on Probability*. Macmillan, London.
- Kierulf, A., Chen, K., and Nievergelt, J. (1990). Smart Game Board and Go Explorer: A study in software and knowledge engineering. *Communications of the Association for Computing Machinery*, 33(2):152–167.
- Kietz, J.-U. and Dzeroski, S. (1994). Inductive logic programming and learnability. *SIGART Bulletin*, 5(1):22–32.
- Kim, J. H. (1983). *CONVINCE: A Conversational Inference Consolidation Engine*. PhD thesis, Department of Computer Science, University of California at Los Angeles.
- Kim, J. H. and Pearl, J. (1983). A computational model for combined causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 190–193, Karlsruhe, Germany. Morgan Kaufmann.
- Kim, J. H. and Pearl, J. (1987). CONVINCE: A conversational inference consolidation engine. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(2):120–132.
- King, R. D., Muggleton, S., Lewis, R. A., and Sternberg, M. J. E. (1992). Drug design by machine learning: the use of inductive logic programming to model the structure activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences of the United States of America*, 89(23):11322–11326.
- King, S., Motet, S., Thomé, J., and Arlabosse, F. (1993). A visual surveillance system for incident detection. In *AAAI 93 Workshop on AI in Intelligent Vehicle Highway Systems*, pages 30–36, Washington, D.C.
- Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.
- Kirkpatrick, S. and Selman, B. (1994). Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264(5163):1297–1301.
- Kirousis, L. M. and Papadimitriou, C. H. (1988). The complexity of recognizing polyhedral scenes. *Journal of Computer and System Sciences*, 37(1):14–38.
- Kister, J., Stein, P., Ulam, S., Walden, W., and Wells, M. (1957). Experiments in chess. *Journal of the Association for Computing Machinery*, 4:174–177.
- Kjaerulff, U. (1992). A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 121–129.
- Knight, K. (1989). Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21(1):93–121.
- Knoblock, C. A. (1990). Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, volume 2, pages 923–928, Boston, Massachusetts. MIT Press.
- Knuth, D. (1968). Semantics for context-free languages. *Mathematical Systems Theory* 2, pages 127–145.
- Knuth, D. E. (1973). *The Art of Computer Programming*, volume 2: Fundamental Algorithms. Addison-Wesley, Reading, Massachusetts, second edition.
- Knuth, D. E. and Bendix, P. B. (1970). Simple word problems in universal algebras. In Leech, J., editor, *Computational Problems in Abstract Algebra*, pages 263–267. Pergamon, New York.
- Knuth, D. E. and Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326.
- Koenderink, J. J. (1990). *Solid Shape*. MIT Press, Cambridge, Massachusetts.
- Koenderink, J. J. and van Doorn, A. J. (1975). Invariant properties of the motion parallax field due to the movement of rigid bodies relative to an observer. *Optica Acta*, 22(9):773–791.
- Kohn, W. (1991). Declarative control architecture. *Communications of the Association for Computing Machinery*, 34(8):65–79.

- Kohonen, T. (1989). *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, third edition.
- Koller, D., Weber, J., Huang, T., Malik, J., Ogasawara, G., Rao, B., and Russell, S. (1994). Towards robust automatic traffic scene analysis in real-time. In *Proceedings of the International Conference on Pattern Recognition*, Israel.
- Kolmogorov, A. N. (1941). Interpolation und extrapolation von stationären zufälligen folgen. *Bulletin of the Academy of Sciences of the USSR*, Ser. Math.5:3-14.
- Kolmogorov, A. N. (1950). *Foundations of the Theory of Probability*. Chelsea, New York. English translation of Kolmogorov (1950).
- Kolmogorov, A. N. (1963). On tables of random numbers. *Sankhya, the Indian Journal of Statistics*, Series A 25.
- Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Problems in Information Transmission*, 1(1):1-7.
- Kolodner, J. (1983). Reconstructive memory: A computer model. *Cognitive Science*, 7:281-328.
- Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, California.
- Konolige, K. (1982). A first order formalization of knowledge and action for a multi-agent planning system. In Hayes, J. E., Michie, D., and Pao, Y.-H., editors, *Machine Intelligence 10*. Ellis Horwood, Chichester, England.
- Koopmans, T. C. (1972). Representation of preference orderings over time. In McGuire, C. B. and Radner, R., editors, *Decision and Organization*. Elsevier/North-Holland, Amsterdam, London, New York.
- Korf, R. E. (1985a). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97-109.
- Korf, R. E. (1985b). Iterative-deepening A*: An optimal admissible tree search. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 1034-1036, Los Angeles, California. Morgan Kaufmann.
- Korf, R. E. (1988). Optimal path finding algorithms. In Kanal, L. N. and Kumar, V., editors, *Search in Artificial Intelligence*, chapter 7, pages 223-267. Springer-Verlag, Berlin.
- Korf, R. E. (1993). Linear-space best-first search. *Artificial Intelligence*, 62(1):41-78.
- Kotok, A. (1962). A chess playing program for the IBM 7090. AIProject Memo 41, MIT Computation Center, Cambridge, Massachusetts.
- Kowalski, R. (1974). Predicate logic as a programming language. In *Proceedings of the IFIP-74 Congress*, pages 569-574. Elsevier/North-Holland.
- Kowalski, R. (1979a). Algorithm = logic + control. *Communications of the Association for Computing Machinery*, 22:424-436.
- Kowalski, R. (1979b). *Logic for Problem Solving*. Elsevier/North-Holland, Amsterdam, London, New York.
- Kowalski, R. (1988). The early years of logic programming. *Communications of the Association for Computing Machinery*, 31:38-43.
- Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, 4(1):67-95.
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts.
- Kripke, S. A. (1963). Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83-94.
- Kruppa, E. (1913). Zur Ermittlung eines Objektes aus zwei Perspektiven mit innerer Orientierung. *Sitz.-Ber. Akad. Wiss., Wien, Math. Naturw., Kl. Abt. IIa*, 122:1939-1948.
- Kuehner, D. (1971). A note on the relation between resolution and Maslov's inverse method. DCL Memo 36, University of Edinburgh.
- Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377-439.
- Kumar, V. (1991). A general heuristic bottom-up procedure for searching AND/OR graphs. *Information Sciences*, 56(1-3):39-57.
- Kumar, V. and Kanal, L. N. (1983). A general branch and bound formulation for understanding and synthesizing and/or tree search procedures. *Artificial Intelligence*, 21:179-198.

- Kumar, V. and Kanal, L. N. (1988). The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In Kanal, L. N. and Kumar, V., editors, *Search in Artificial Intelligence*, chapter 1, pages 1-27. Springer-Verlag, Berlin.
- Kumar, V., Nau, D. S., and Kanal, L. N. (1988). A general branch-and-bound formulation for AND/OR graph and game tree search. In Kanal, L. N. and Kumar, V., editors, *Search in Artificial Intelligence*, chapter 3, pages 91-130. Springer-Verlag, Berlin.
- Kurzweil, R. (1990). *The Age of Intelligent Machines*. MIT Press, Cambridge, Massachusetts.
- Kyburg, H. E. (1977). Randomness and the right referenceclass. *The Journal of Philosophy*, 74(9):501-521.
- Kyburg, H. E. (1983). The reference class. *Philosophy of Science*, 50:374-397.
- La Mettrie, J. O. d. (1748). *L'homme machine*. E. Luzac, Leyde. Translated into English as La Mettrie (1912).
- La Mettrie, J. O. d. (1912). *Man a Machine*. Open Court, La Salle, Illinois. English translation of La Mettrie (1748).
- Ladkin, P. (1986a). Primitives and units for time specification. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, volume 1, pages 354-359, Philadelphia, Pennsylvania. Morgan Kaufmann.
- Ladkin, P. (1986b). Time representation: a taxonomy of interval relations. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, volume 1, pages 360-366, Philadelphia, Pennsylvania. Morgan Kaufmann.
- Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1): 1-64.
- Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in Soar: the anatomy of a general learning mechanism. *Machine Learning*, 1:11-46.
- Laird, J. E., Yager, E. S., Hucka, M., and Tuck, M. (1991). Robo-Soar: an integration of external interaction, planning, and learning using Soar. *Robotics and Autonomous Systems*, 8(1-2):113-129.
- Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*. University of Chicago Press, Chicago, Illinois.
- Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press, Chicago, Illinois.
- Langley, P., Simon, H. A., Bradshaw, G. L., and Zytkow, J. M. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes*. MIT Press, Cambridge, Massachusetts.
- Lassez, J.-L., Maher, M. J., and Marriott, K. (1988). Unification revisited. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*, pages 587-625. Morgan Kaufmann, San Mateo, California.
- Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer, Dordrecht, The Netherlands.
- Lauritzen, S. L. (1991). The EM algorithm for graphical association models with missing data. Technical Report TR-91-05, Department of Statistics, Aalborg University.
- Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, B* 50(2): 157-224.
- Lauritzen, S. L. and Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17:31-57.
- Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research*, 14(4):699-719.
- Le Cun, Y., Jackel, L. D., Boser, B., and Denker, J. S. (1989). Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41-46.
- Lee, K.-F. (1989). *Automatic Speech Recognition: The Development of the SPHINX System*. Kluwer, Dordrecht, The Netherlands.
- Lee, K.-F. and Mahajan, S. (1988). A pattern classification approach to evaluation function learning. *Artificial Intelligence*, 36(1): 1-26.
- Lefkovitz, D. (1960). A strategic pattern recognition program for the game Go. Technical Note 60-243, Wright Air Development Division, University

- of Pennsylvania, The Moore School of Electrical Engineering.
- Lenat, D. B. (1983). EURISKO: a program that learns new heuristics and domain concepts: the nature of heuristics, III: program design and results. *Artificial Intelligence*, 21(1-2):61-98.
- Lenat, D. B. and Brown, J. S. (1984). Why AM and EURISKO appear to work. *Artificial Intelligence*, 23(3):269-294.
- Lenat, D. B. and Feigenbaum, E. A. (1991). On the thresholds of knowledge. *Artificial Intelligence*, 47(1-3): 185-250.
- Lenat, D. B. and Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, Reading, Massachusetts.
- Leonard, H. S. and Goodman, N. (1940). The calculus of individuals and its uses. *Journal of Symbolic Logic*, 5(2):45-55.
- Leonard, J. J. and Durrant-Whyte, H. F. (1992). *Directed sonar sensing for mobile robot navigation*. Kluwer, Dordrecht, The Netherlands.
- Leśniewski, S. (1916). Podstawy ogólnej teorii mnogości. Moscow.
- Levesque, H. J. and Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3(2):78-93.
- Levy, D. N. L. (1983). *Computer Gamesmanship: The Complete Guide to Creating and Structuring Intelligent Games Programs*. Simon and Schuster, New York.
- Levy, D. N. L., editor (1988a). *Computer Chess Compendium*. Springer-Verlag, Berlin.
- Levy, D. N. L., editor (1988b). *Computer Games*. Springer-Verlag, Berlin. Two volumes.
- Lewis, D. K. (1966). An argument for the identity theory. *The Journal of Philosophy*, 63(1):17-25.
- Lewis, D. K. (1972). General semantics. In Davidson, D. and Harman, G., editors, *Semantics of Natural Language*, pages 169-218. D. Reidel, Dordrecht, The Netherlands.
- Lewis, D. K. (1980). Mad pain and Martian pain. In Block, N., editor, *Readings in Philosophy of Psychology*, volume 1, pages 216-222. Harvard University Press, Cambridge, Massachusetts.
- Li, M. and Vitanyi, P. M. B. (1993). *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin.
- Lifschitz, V. (1986). On the semantics of STRIPS. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 1-9, Timberline, Oregon. Morgan Kaufmann.
- Lifschitz, V. (1989). Between circumscription and autoepistemic logic. In Brachman, R. J. and Levesque, H. J., editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 235-244, Toronto, Ontario. Morgan Kaufmann.
- Lighthill, J. (1973). Artificial intelligence: A general survey. In Lighthill, J., Sutherland, N. S., Needham, R. M., Longuet-Higgins, H. C., and Michie, D., editors, *Artificial Intelligence: A Paper Symposium*. Science Research Council of Great Britain, London.
- Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell Systems Technical Journal*, 44(10):2245-2269.
- Linden, T. A. (1991). Representing software designs as partially developed plans. In Lowry, M. R. and McCartney, R. D., editors, *Automating Software Design*, pages 603-625. MIT Press, Cambridge, Massachusetts.
- Lindsay, R. K. (1963). Inferential memory as the basis of machines which understand natural language. In Feigenbaum, E. A. and Feldman, J., editors, *Computers and Thought*, pages 217-236. McGraw-Hill, New York.
- Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. McGraw-Hill, New York.
- Lloyd, J. W. (1987). *Foundations of Logic Programming*. Springer-Verlag, Berlin.
- Locke, W. N. and Booth, A. D. (1955). *Machine Translation of Languages: Fourteen Essays*. MIT Press, Cambridge, Massachusetts.
- Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133-135.

- Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1-4):47-66.
- Loveland, D. W. (1968). Mechanical theorem proving by model elimination. *Journal of the Association for Computing Machinery*, 15(2):236-251.
- Loveland, D. W. (1984). Automated theorem-proving: A quarter-century review. *Contemporary Mathematics*, 29:1-45.
- Lowe, D. G. (1987). Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31:355-395.
- Löwenheim, L. (1915). Über möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76:447-470. Reprinted in English translation in van Heijenoort (1967).
- Lowerre, B. T. and Reddy, R. (1980). The HARPY speech recognition system. In Lea, W. A., editor, *Trends in Speech Recognition*, chapter 15. Prentice-Hall, Englewood Cliffs, New Jersey.
- Lowry, M. R. and McCartney, R. D. (1991). *Automating Software Design*. MIT Press, Cambridge, Massachusetts.
- Loyd, S. (1959). *Mathematical Puzzles of Sam Loyd: Selected and Edited by Martin Gardner*. Dover, New York.
- Lozano-Pérez, T., Mason, M., and Taylor, R. (1984). Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3-24.
- Lucas, J. R. (1961). Minds, machines, and Gödel. *Philosophy*, 36.
- Luger, G. F. and Stubblefield, W. A. (1993). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Benjamin/Cummings, Redwood City, California, second edition.
- Mackworth, A. K. (1973). Interpreting pictures of polyhedral scenes. *Artificial Intelligence*, 4:121-137.
- Maes, P., Darrell, T., Blumberg, B., and Pentland, A. (1994). ALIVE: Artificial Life Interactive Video Environment. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, page 1506, Seattle, Washington. AAAI Press.
- Magerman, D. (1993). *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University.
- Mahanti, A. and Daniels, C. J. (1993). A SIMD approach to parallel heuristic search. *Artificial Intelligence*, 60(2):243-282.
- Malik, J. (1987). Interpreting line drawings of curved objects. *International Journal of Computer Vision*, 1(1):73-103.
- Malik, J. and Rosenholtz, R. (1994). Recovering surface curvature and orientation from texture distortion: a least squares algorithm and sensitivity analysis. In Eklundh, J.-O., editor, *Proceedings of the Third European Conf. on Computer Vision*, pages 353-364, Stockholm. Springer-Verlag. Published as Lecture Notes in Computer Science 800.
- Manin, Y. I. (1977). *A Course in Mathematical Logic*. Springer-Verlag, Berlin.
- Mann, W. C. and Thompson, S. A. (1983). Relational propositions in discourse. Technical Report RR-83-115, Information Sciences Institute.
- Manna, Z. and Waldinger, R. (1971). Toward automatic program synthesis. *Communications of the Association for Computing Machinery*, 14(3):151-165.
- Manna, Z. and Waldinger, R. (1985). *The Logical Basis for Computer Programming: Volume I: Deductive Reasoning*. Addison-Wesley, Reading, Massachusetts.
- Manna, Z. and Waldinger, R. (1986). Special relations in automated deduction. *Journal of the Association for Computing Machinery*, 33(1):1-59.
- Manna, Z. and Waldinger, R. (1992). Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering*, 18(8):674-704.
- Marchand, M., Golea, M., and Ruján, P. (1990). A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11:487-492.
- Markov, A. A. (1913). An example of statistical investigation in the text of "Eugene Onegin" illustrating coupling of "tests" in chains. *Proceedings of the Academy of Sciences of St. Petersburg*, 7.
- Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, New York.

- Marsland, A. T. and Schaeffer, J., editors (1990). *Computers, Chess, and Cognition*. Springer-Verlag, Berlin.
- Martelli, A. and Montanari, U. (1973). Additive AND/OR graphs. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pages 1–11, Stanford, California. IJCAII.
- Martelli, A. and Montanari, U. (1976). Unification in linear time and space: A structured presentation. Internal Report B 76-16, Istituto di Elaborazione della Informazione, Pisa, Italy.
- Martelli, A. and Montanari, U. (1978). Optimizing decision trees through heuristically guided search. *Communications of the Association for Computing Machinery*, 21:1025–1039.
- Martin, C. D. (1993). The myth of the awesome thinking machine. *Communications of the Association for Computing Machinery*, 36(4): 120–133.
- Martin, J. H. (1990). *A Computational Model of Metaphor Interpretation*. Academic Press.
- Maslov, S. Y. (1964). An inverse method for establishing deducibility in classical predicate calculus. *Doklady Akademii nauk SSSR*, 159:17–20.
- Maslov, S. Y. (1967). An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus. *Doklady Akademii nauk SSSR*, 172:22–25.
- Maslov, S. Y. (1971). Relationship between tactics of the inverse method and the resolution method. *Seminars in Mathematics, V. A. Steklov Mathematical Institute, Leningrad, Consultants Bureau, New York-London*, 16:69–73.
- Mason, M. T. (1993). Kicking the sensing habit. *AI Magazine*, 14(1):58–59.
- Mates, B. (1953). *Stoic Logic*. University of California Press, Berkeley and Los Angeles.
- Maxwell, J. and Kaplan, R. (1993). The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590.
- Mays, E., Apté, C., Griesmer, J., and Kastner, J. (1987). Organizing knowledge in a complex financial domain. *IEEE Expert*, 2(3):61–70.
- McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 634–639, Anaheim, California. AAAI Press.
- McAllester, D. A. (1980). An outlook on truth maintenance. AI Memo 551, MIT AI Laboratory, Cambridge, Massachusetts.
- McAllester, D. A. (1988). Conspiracy numbers for min-max search. *Artificial Intelligence*, 35(3):287–310.
- McAllester, D. A. (1989). *Ontic: A Knowledge Representation System for Mathematics*. MIT Press, Cambridge, Massachusetts.
- McAllester, D. A. and Givan, R. (1992). Natural language syntax and first-order inference. *Artificial Intelligence*, 56(1):1–20.
- McCarthy, J. (1958). Programs with common sense. In *Proceedings of the Symposium on Mechanisation of Thought Processes*, volume 1, pages 77–84, London. Her Majesty's Stationery Office.
- McCarthy, J. (1963). Situations, actions, and causal laws. Memo 2, Stanford University Artificial Intelligence Project, Stanford, California. Reprinted as part of McCarthy (1968).
- McCarthy, J. (1968). Programs with common sense. In Minsky, M. L., editor, *Semantic Information Processing*, pages 403–418. MIT Press, Cambridge, Massachusetts.
- McCarthy, J. (1977). Epistemological problems in artificial intelligence. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Cambridge, Massachusetts. IJCAII.
- McCarthy, J. (1978). History of LISP. In Wexelblat, R. L., editor, *History of Programming Languages: Proceedings of the ACM SIGPLAN Conference*, pages 173–197. Academic Press. Published in 1981; 1978 is date of conference.
- McCarthy, J. (1980). Circumscription: a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1–2):27–39.
- McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., Michie, D., and Swann, M., editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, Scotland.

- McCawley, J. D. (1993). *Everything That Linguists Have Always Wanted to Know About Logic But Were Ashamed to Ask*. University of Chicago Press, Chicago, Illinois, second edition.
- McCorduck, P. (1979). *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. W. H. Freeman, New York.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–137.
- McCune, W. W. (1992). Automated discovery of new axiomatizations of the left group and right group calculi. *Journal of Automated Reasoning*, 9(1):1–24.
- McDermott, D. (1976). Artificial intelligence meets natural stupidity. *SIGART Newsletter*, 57.
- McDermott, D. (1978a). Planning and acting. *Cognitive Science*, 2(2):71–109.
- McDermott, D. (1978b). Tarskian semantics, or, no notation without denotation! *Cognitive Science*, 2(3).
- McDermott, D. (1987). A critique of pure reason. *Computational Intelligence*, 3(3):151–237. Includes responses by a number of commentators and final rebuttal by the author.
- McDermott, D. (1991). Regression planning. *International Journal of Intelligent Systems*, 6:357–416.
- McDermott, D. and Doyle, J. (1980). Non-monotonic logic I. *Artificial Intelligence*, 13(1–2):41–72.
- McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19(I):39–88.
- Mead, C. (1989). *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, Massachusetts.
- Megiddo, N. and Wigderson, A. (1986). On play by means of computing machines. In Halpern, J. Y., editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference (TARK-86)*, pages 259–274, Monterey, California. IBM and AAAI, Morgan Kaufmann.
- Melčuk, I. A. and Polguere, A. (1988). A formal lexicon in the meaning-text theory (or how to do lexica with words). *Computational Linguistics*, 13(3–4):261–275.
- Mero, L. (1984). A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23(1): 13–27.
- Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091.
- Mézard, M. and Nadal, J.-P. (1989). Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics*, 22:2191–2204.
- Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1983). *Machine Learning: An Artificial Intelligence Approach*, volume 1. Morgan Kaufmann, San Mateo, California.
- Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1986). *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann, San Mateo, California.
- Michie, D. (1966). Game-playing and game-learning automata. In Fox, L., editor, *Advances in Programming and Non-Numerical Computation*, pages 183–200. Pergamon, New York.
- Michie, D. (1972). Machine intelligence at Edinburgh. *Management Informatics*, 2(1):7–12.
- Michie, D. (1982). The state of the art in machine learning. In *Introductory Readings in Expert Systems*, pages 209–229. Gordon and Breach, New York.
- Michie, D. (1986). Current developments in expert systems. In *Proc. 2nd Australian Conference on Applications of Expert Systems*, pages 163–182, Sydney, Australia.
- Michie, D. and Chambers, R. A. (1968). BOXES: An experiment in adaptive control. In Dale, E. and Michie, D., editors, *Machine Intelligence 2*, pages 125–133. Elsevier/North-Holland, Amsterdam, London, New York.
- Michie, D., Spiegelhalter, D. J., and Taylor, C. C., editors (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Chichester, England.
- Miles, F. A. (1969). *Excitable Cells*. William Heinemann Medical Books, London.
- Mill, J. S. (1843). *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence, and Methods of Scientific Investigation*. J. W. Parker, London.

- Mill, J. S. (1863). *Utilitarianism*. Parker, Son and Bourn, London.
- Miller, A. C., Merkhofer, M. M., Howard, R. A., Matheson, J. E., and Rice, T. R. (1976). Development of automated aids for decision analysis. Technical report, SRI International, Menlo Park, California.
- Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, Arlington, Virginia. Morgan Kaufmann.
- Milne, A. A. (1926). *Winnie-the-Pooh*. Methuen, London. With decorations by Ernest H. Shepard.
- Minker, J., editor (1988). *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, San Mateo, California.
- Minsky, M. L. (1954). *Neural Nets and the Brain-Model Problem*. PhD thesis, Princeton University.
- Minsky, M. L., editor (1968). *Semantic Information Processing*. MIT Press, Cambridge, Massachusetts.
- Minsky, M. L. (1975). A framework for representing knowledge. In Winston, P. M., editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York. Originally appeared as an MIT Artificial Intelligence Laboratory memo; the present version is abridged, but is the most widely cited. Another, later abridged version appeared in Haugeland (1981).
- Minsky, M. L. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Massachusetts, first edition.
- Minsky, M. L. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Massachusetts, expanded edition.
- Minton, S. (1984). Constraint-based generalization: Learning game-playing plans from single examples. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, pages 251–254, Austin, TX. Morgan Kaufmann.
- Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, St. Paul, Minnesota. Morgan Kaufmann.
- Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1–3):161–205.
- Mitchell, T., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80.
- Mitchell, T. M. (1977). Version spaces: a candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 305–310, Cambridge, Massachusetts. IJCAI.
- Mitchell, T. M. (1980). The need for biases in learning generalizations. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, New Brunswick, New Jersey.
- Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18(2):203–226.
- Mitchell, T. M. (1990). Becoming increasingly reactive (mobile robots). In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, volume 2, pages 1051–1058, Boston, Massachusetts. MIT Press.
- Mitchell, T. M., Utgoff, P. E., and Banerji, R. (1983). Learning by experimentation: acquiring and refining problem-solving heuristics. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 163–190. Morgan Kaufmann, San Mateo, California.
- Montague, R. (1970). English as a formal language. In *Linguaggi nella Società e nella Tecnica*, pages 189–224. Edizioni di Comunità, Milan. Reprinted in (Thomason, 1974, pp. 188–221).
- Montague, R. (1973). The proper treatment of quantification in ordinary English. In Hintikka, K. J. J., Moravcsik, J. M. E., and Suppes, P., editors, *Approaches to Natural Language*. D. Reidel, Dordrecht, The Netherlands.
- Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping—reinforcement learning with less data and less time. *Machine Learning*, 13:103–130.
- Moore, J. and Newell, A. (1973). How can Merlin understand? In Gregg, L., editor, *Knowledge and Cognition*. Lawrence Erlbaum Associates, Potomac, Maryland.

- Moore, R. C. (1980). Reasoning about knowledge and action. Artificial Intelligence Center Technical Note 191, SRI International, Menlo Park, California.
- Moore, R. C. (1985a). A formal theory of knowledge and action. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, pages 319-358. Ablex, Norwood, New Jersey.
- Moore, R. C. (1985b). Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75-94.
- Moore, R. C. (1993). Autoepistemic logic revisited. *Artificial Intelligence*, 59(1-2):27-30.
- Moravec, H. (1988). *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press, Cambridge, Massachusetts.
- Morgenstern, L. (1987). Knowledge preconditions for actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 867-874, Milan, Italy. Morgan Kaufmann.
- Morrison, P. and Morrison, E., editors (1961). *Charles Babbage and His Calculating Engines: Selected Writings by Charles Babbage and Others*. Dover, New York.
- Mostow, J. and Priditidis, A. E. (1989). Discovering admissible heuristics by abstracting and optimizing: a transformational approach. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 1, pages 701-707, Detroit, Michigan. Morgan Kaufmann.
- Motzkin, T. S. and Schoenberg, I. J. (1954). The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6(3):393-404.
- Mourelatos, A. P. D. (1978). Events, processes, and states. *Linguistics and Philosophy*, 2:415-434.
- Moussouris, J., Holloway, J., and Greenblatt, R. (1979). CHEOPS: A chess-oriented processing system. In Hayes, J. E., Michie, D., and Mikulich, L. I., editors, *Machine Intelligence 9*, pages 351-360. Ellis Horwood, Chichester, England.
- Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8:295-318.
- Muggleton, S. (1992). *Inductive Logic Programming*. Academic Press, New York.
- Muggleton, S. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *MLC-88*.
- Muggleton, S. and Cao, R. (1990). Efficient induction of logic programs. In *Proceedings of the Workshop on Algorithmic Learning Theory*, Tokyo.
- Muggleton, S., King, R. D., and Sternberg, M. J. E. (1992). Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647-657.
- Mundy, J. and Zisserman, A., editors (1992). *Geometric Invariance in Computer Vision*. MIT Press, Cambridge, Massachusetts.
- Nagel, T. (1974). What is it like to be a bat? *Philosophical Review*, 83:435-450.
- Nalwa, V. S. (1993). *A Guided Tour of Computer Vision*. Addison-Wesley, Reading, Massachusetts.
- Nau, D. S. (1980). Pathology on game trees: A summary of results. In *Proceedings of the First Annual National Conference on Artificial Intelligence (AAAI-80)*, pages 102-104, Stanford, California. AAAI.
- Nau, D. S. (1983). Pathology on game trees revisited, and an alternative to minimaxing. *Artificial Intelligence*, 21(1-2):221-244.
- Nau, D. S., Kumar, V., and Kanal, L. N. (1984). General branch and bound, and its relation to A* and AO*. *Artificial Intelligence*, 23:29-58.
- Naur, P. (1963). Revised report on the algorithmic language Algol 60. *Communications of the Association for Computing Machinery*, 6(1): 1-17.
- Neal, R. M. (1991). Connectionist learning of belief networks. *Artificial Intelligence*, 56:71-113.
- Neapolitan, R. E. (1990). *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. Wiley, New York.
- Netto, E. (1901). *Lehrbuch der Combinatorik*. B. G. Teubner, Leipzig.
- Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18(1):82-127.
- Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts.

- Newell, A. and Ernst, G. (1965). The search for generality. In Kalenich, W. A., editor, *Information Processing 1965: Proceedings of IFIP Congress 1965*, volume 1, pages 17-24. Spartan.
- Newell, A. and Shaw, J. C. (1957). Programming the logic theory machine. In *Proceedings of the 1957 Western Joint Computer Conference*, pages 230-240. IRE.
- Newell, A., Shaw, J. C., and Simon, H. A. (1957). Empirical explorations with the logic theory machine. *Proceedings of the Western Joint Computer Conference*, 15:218-239. Reprinted in Feigenbaum and Feldman (1963).
- Newell, A., Shaw, J. C., and Simon, H. A. (1958). Chess playing programs and the problem of complexity. *IBM Journal of Research and Development*, 4(2):320-335. Reprinted in Feigenbaum and Feldman (1963).
- Newell, A. and Simon, H. A. (1961). GPS, a program that simulates human thought. In Billing, H., editor, *Lernende Automaten*, pages 109-124. R. Oldenbourg, Munich, Germany. Reprinted in (Feigenbaum and Feldman, 1963, pp. 279-293).
- Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Neyman, A. (1985). Bounded complexity justifies cooperation in the finitely repeated prisoners' dilemma. *Economics Letters*, 19:227-229.
- Nicholson, A. E. and Brady, J. M. (1992). The data association problem when monitoring robot vehicles using dynamic belief networks. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, pages 689-693, Vienna, Austria. Wiley.
- Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill, New York.
- Nilsson, N. J. (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York.
- Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.
- Nilsson, N. J. (1984). Shakey the robot. Technical Note 323, SRI International, Menlo Park, California.
- Nilsson, N. J. (1990). *The Mathematical Foundations of Learning Machines*. Morgan Kaufmann, San Mateo, California. Introduction by Terrence I. Sejnowski and Halbert White.
- Nitta, K., Taki, K., and Ichiyoshi, N. (1992). Experimental parallel inference software. In *Fifth Generation Computer Systems 1992*, volume 1, pages 166-190, Tokyo. IOS Press.
- Norvig, P. (1988). Multiple simultaneous interpretations of ambiguous sentences. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society*.
- Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, San Mateo, California.
- Nowick, S. M., Dean, M. E., Dill, D. L., and Horowitz, M. (1993). The design of a high-performance cache controller: a case study in asynchronous synthesis. *Integration: The VLSI Journal*, 15(3):241-262.
- Nunberg, G. (1979). The non-uniqueness of semantic solutions: Polysemy. *Language and Philosophy*, 3(2):143-184.
- Nussbaum, M. C. (1978). *Aristotle's De Motu Animalium*. Princeton University Press, Princeton, New Jersey.
- O'Keefe, R. (1990). *The Craft of Prolog*. MIT Press, Cambridge, Massachusetts.
- Olawsky, D. and Gini, M. (1990). Deferred planning and sensor use. In Sycara, K. P., editor, *Proceedings, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, California. Defense Advanced Research Projects Agency (DARPA), Morgan Kaufmann.
- Olesen, K. G. (1993). Causal probabilistic networks with both discrete and continuous variables. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 15(3):275-279.
- Oliver, R. M. and Smith, J. Q., editors (1990). *Influence Diagrams, BeliefNets and Decision Analysis*. Wiley, New York.
- Olson, C. F. (1994). Time and space efficient pose clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 251-258, Seattle, Washington.
- Ortony, A., editor (1979). *Metaphor and Thought*. Cambridge University Press, Cambridge.

- Osherson, D. N., Stob, M., and Weinstein, S. (1986). *Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, Cambridge, Massachusetts.
- Paige, R. and Henglein, F. (1987). Mechanical translation of set theoretic problem specifications into efficient RAM code—a case study. *Journal of Symbolic Computation*, 4:207-232.
- Palay, A. J. (1985). *Searching with Probabilities*. Pitman, London.
- Palmieri, G. and Sanna, R. (1960). Automatic probabilistic programmer/analyzer for pattern recognition. *Methodos*, 12(48):331–357.
- Papadimitriou, C. H. and Yannakakis, M. (1994). On complexity as bounded rationality. In *Symposium on Theory of Computation (STOC-94)*.
- Parker, D. B. (1985). Learning logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.
- Partridge, D. (1991). *A New Guide to Artificial Intelligence*. Ablex, Norwood, New Jersey.
- Paterson, M. S. and Wegman, M. N. (1978). Linear unification. *Journal of Computer and System Sciences*, 16:158-167.
- Patrick, B. G., Almulla, M., and Newborn, M. M. (1992). An upper bound on the time complexity of iterative-deepening-A*. *Annals of Mathematics and Artificial Intelligence*, 5(2-4):265–278.
- Paul, R. P. (1981). *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, Massachusetts.
- Peano, G. (1889). *Arithmetices principia, nova methodo exposita*. Fratres Bocca, Turin.
- Pearl, J. (1982a). Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, pages 133–136, Pittsburgh, Pennsylvania. Morgan Kaufmann.
- Pearl, J. (1982b). The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the Association for Computing Machinery*, 25(8):559-564.
- Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, Massachusetts.
- Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241-288.
- Pearl, J. (1987). Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32:247-257.
- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California.
- Pednault, E. P. D. (1986). Formulating multiagent, dynamic-world problems in the classical planning framework. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 47-82, Timberline, Oregon. Morgan Kaufmann.
- Peirce, C. S. (1870). Description of a notation for the logic of relatives, resulting from an amplification of the conceptions of Boole's calculus of logic. *Memoirs of the American academy of arts and sciences*, 9:317-378.
- Peirce, C. S. (1883). A theory of probable inference. Note B. The logic of relatives. In *Studies in logic by members of the Johns Hopkins University*, pages 187–203, Boston.
- Peirce, C. S. (1902). Logic as semiotic: the theory of signs. Unpublished manuscript; reprinted in (Buchler, 1955, pp. 98–119).
- Penberthy, J. S. and Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, pages 103–114.
- Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 2:437-454.
- Penrose, R. (1990). The emperor's new mind: concerning computers, minds, and the laws of physics. *Behavioral and Brain Sciences*, 13(4):643-654.
- Peot, M. and Smith, D. (1992). Conditional nonlinear planning. In Hendler, J., editor, *Proceedings of the First International Conference on AI Planning Systems*, pages 189-197, College Park, Maryland. Morgan Kaufmann.
- Pereira, F. (1983). Logic for natural language analysis. Technical Note 275, SRI International.

- Pereira, F. C. N. and Shieber, S. M. (1987). *Prolog and Natural-Language Analysis*. Center for the Study of Language and Information (CSLI), Stanford, California.
- Pereira, F. C. N. and Warren, D. H. D. (1980). Definite clause grammars for language analysis: a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231-278.
- Peterson, C., Redfield, S., Keeler, J. D., and Hartman, E. (1990). An optoelectronic architecture for multilayer learning in a single photorefractive crystal. *Neural Computation*, 2:25-34.
- Pinker, S. (1989). *Learnability and Cognition*. MIT Press, Cambridge, MA.
- Place, U. T. (1956). Is consciousness a brain process? *British Journal of Psychology*, 47:44-50.
- Plotkin, G. D. (1971). *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University.
- Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46-57, Providence, Rhode Island. IEEE, IEEE Computer Society Press.
- Pohl, I. (1969). Bi-directional and heuristic search in path problems. Technical Report 104, SLAC (Stanford Linear Accelerator Center, Stanford, California).
- Pohl, I. (1970). First results on the effect of error in heuristic search. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 5*, pages 219-236. Elsevier/North-Holland, Amsterdam, London, New York.
- Pohl, I. (1971). Bi-directional search. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 6*, pages 127-140. Edinburgh University Press, Edinburgh, Scotland.
- Pohl, I. (1973). The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pages 20-23, Stanford, California. IJCAI.
- Pohl, I. (1977). Practical and theoretical considerations in heuristic search algorithms. In Elcock, E. W. and Michie, D., editors, *Machine Intelligence 8*, pages 55-72. Ellis Horwood, Chichester, England.
- Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, Illinois.
- Pólya, G. (1957). *How to Solve It: A New Aspect of Mathematical Method*. Doubleday, Garden City, New York, second edition.
- Pomerleau, D. A. (1993). *Neural Network Perception for Mobile Robot Guidance*. Kluwer, Dordrecht, The Netherlands.
- Popper, K. R. (1959). *The Logic of Scientific Discovery*. Basic Books, New York.
- Popper, K. R. (1962). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Basic Books, New York.
- Post, E. L. (1921). Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43:163-185.
- Pradhan, M., Provan, G. M., Middleton, B., and Henrion, M. (1994). Knowledge engineering for large belief networks. In *Proceedings of Uncertainty in Artificial Intelligence*, Seattle, Washington. Morgan Kaufmann.
- Pratt, V. R. (1976). Semantical considerations on Floyd-Hoare logic. In *Proceedings of the 17th IEEE Symposium on the Foundations of Computer Science*, pages 109-121.
- Prawitz, D. (1960). An improved proof procedure. *Theoria*, 26:102-139.
- Prawitz, D. (1965). *Natural Deduction: A Proof Theoretical Study*. Almqvist and Wiksell, Stockholm.
- Prieditis, A. E. (1993). Machine discovery of effective admissible heuristics. *Machine Learning*, 12(1-3):117-141.
- Prinz, D. G. (1952). Robot chess. *Research*, 5:261-266.
- Prior, A. N. (1967). *Past, Present, and Future*. Oxford University Press, Oxford.
- Pullum, G. K. (1991). *The Great Eskimo Vocabulary Hoax (and Other Irreverent Essays on the Study of Language)*. University of Chicago Press, Chicago, Illinois.

- Purdom, P. (1983). Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117-133.
- Putnam, H. (1960). Minds and machines. In Hook, S., editor, *Dimensions of Mind*, pages 138-164. Macmillan, London.
- Putnam, H. (1963). 'Degree of confirmation' and inductive logic. In Schilpp, P. A., editor, *The Philosophy of Rudolf Carnap*. Open Court, La Salle, Illinois.
- Putnam, H. (1967). The nature of mental states. In Capitan, W. H. and Merrill, D. D., editors, *Art, Mind, and Religion*, pages 37-48. University of Pittsburgh Press, Pittsburgh, Pennsylvania. Original title was "Psychological predicates"; title changed in later reprints at the request of the author.
- Pylyshyn, Z. W. (1974). Minds, machines and phenomenology: Some reflections on Dreyfus' "What Computers Can't Do". *International Journal of Cognitive Psychology*, 3(1):57-77.
- Pylyshyn, Z. W. (1984). *Computation and Cognition: Toward a Foundation for Cognitive Science*. MIT Press, Cambridge, Massachusetts.
- Quillian, M. R. (1961). A design for an understanding machine. Paper presented at a colloquium: Semantic Problems in Natural Language, King's College, Cambridge, England.
- Quillian, M. R. (1968). Semantic memory. In Minsky, M. L., editor, *Semantic Information Processing*, pages 216-270. MIT Press, Cambridge, Massachusetts.
- Quine, W. V. (1953). Two dogmas of empiricism. In *From a Logical Point of View*, pages 20-46. Harper and Row, New York.
- Quine, W. V. (1960). *Word and Object*. MIT Press, Cambridge, Massachusetts.
- Quine, W. V. (1982). *Methods of Logic*. Harvard University Press, Cambridge, Massachusetts, fourth edition.
- Quinlan, E. and O'Brien, S. (1992). Sublanguage: characteristics and selection guidelines for MT. In *AI and Cognitive Science '92: Proceedings of Annual Irish Conference on Artificial Intelligence and Cognitive Science '92*, pages 342-345, Limerick, Ireland. Springer-Verlag.
- Quinlan, J. R. (1979). Discovering rules from large collections of examples: a case study. In Michie, D., editor, *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh, Scotland.
- Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81-106.
- Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3):239-266.
- Quinlan, J. R. (1993). Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 236-243, Amherst, Massachusetts. Morgan Kaufmann.
- Quinlan, J. R. and Cameron-Jones, R. M. (1993). FOIL: a midterm report. In Brazdil, P. B., editor, *European Conference on Machine Learning Proceedings (ECML-93)*, pages 3-20, Vienna. Springer-Verlag.
- Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman, New York.
- Rabiner, L. R. (1990). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*. Reprinted in Waibel and Lee (1990).
- Rabiner, L. R. and Juang, B.-H. (1993). *Fundamentals of Speech Recognition*. Prentice-Hall.
- Raiert, M. H. (1986). *Legged Robots That Balance*. MIT Press, Cambridge, Massachusetts.
- Ramsey, F. P. (1931). Truth and probability. In Braithwaite, R. B., editor, *The Foundations of Mathematics and Other Logical Essays*. Harcourt Brace Jovanovich, New York.
- Raphael, B. (1968). SIR: Semantic information retrieval. In Minsky, M. L., editor, *Semantic Information Processing*, pages 33-134. MIT Press, Cambridge, Massachusetts.
- Raphael, B. (1976). *The Thinking Computer: Mind Inside Matter*. W. H. Freeman, New York.
- Ratner, D. and Warmuth, M. (1986). Finding a shortest solution for the $n \times n$ extension of the 15-puzzle is intractable. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, volume 1, pages 168-172, Philadelphia, Pennsylvania. Morgan Kaufmann.
- Reichardt, J. (1978). *Robots: Fact, Fiction, and Prediction*. Penguin Books, New York.

- Reichenbach, H. (1949). *The Theory of Probability: An Inquiry into the Logical and Mathematical Foundations of the Calculus of Probability*. University of California Press, Berkeley and Los Angeles, second edition.
- Reif, J. H. (1979). Complexity of the mover's problem and generalizations. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 421–427, San Juan, Puerto Rico. IEEE, IEEE Computer Society Press.
- Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13(1–2):81–132.
- Reiter, R. (1991). The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, New York.
- Reitman, W. and Wilcox, B. (1979). The structure and performance of the INTERIM.2 Go program. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence (IJCAI-79)*, pages 711–719, Tokyo. IJCAII.
- Remus, H. (1962). Simulation of a learning machine for playing Go. In *Proceedings IFIP Congress*, pages 428–432. Elsevier/North-Holland.
- Rényi, A. (1970). *Probability Theory*. Elsevier/North-Holland, Amsterdam, London, New York.
- Rescher, N. and Urquhart, A. (1971). *Temporal Logic*. Springer-Verlag, Berlin.
- Resnik, P. (1993). Semantic classes and syntactic ambiguity. ARPA Workshop on Human Language Technology. Princeton.
- Rich, E. and Knight, K. (1991). *Artificial Intelligence*. McGraw-Hill, New York, second edition.
- Rieger, C. (1976). An organization of knowledge for problem solving and language comprehension. *Artificial Intelligence*, 7:89–127.
- Ringle, M. (1979). *Philosophical Perspectives in Artificial Intelligence*. Humanities Press, Atlantic Highlands, New Jersey.
- Rissanen, J. (1984). Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, IT-30(4):629–636.
- Ritchie, G. D. and Hanna, F. K. (1984). AM: a case study in AI methodology. *Artificial Intelligence*, 23(3):249–268.
- Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, 2(3):229–246.
- Roach, J. W., Sundararajan, R., and Watson, L. T. (1990). Replacing unification by constraint satisfaction to improve logic program expressiveness. *Journal of Automated Reasoning*, 6(1):51–75.
- Roberts, D. D. (1973). *The Existential Graphs of Charles S. Peirce*. Mouton, The Hague and Paris.
- Roberts, L. G. (1963). Machine perception of three-dimensional solids. Technical Report 315, MIT Lincoln Laboratory. Ph.D. dissertation.
- Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41.
- Rock, I. (1984). *Perception*. W. H. Freeman, New York.
- Rorty, R. (1965). Mind-body identity, privacy, and categories. *Review of Metaphysics*, 19(1):24–54.
- Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automaton. Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Ithaca, New York.
- Rosenblatt, F. (1960). On the convergence of reinforcement procedures in simple perceptrons. Report VG-1196-G-4, Cornell Aeronautical Laboratory, Ithaca, New York.
- Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan, Chicago.
- Rosenschein, J. S. and Genesereth, M. R. (1987). Communication and cooperation among logic-based agents. In Friesen, O. and Golshani, F., editors, *Sixth Annual International Phoenix Conference on Computers and Communications: 1987 Conference Proceedings*, pages 594–600. IEEE Computer Society Press.
- Rosenschein, S. J. (1985). Formal theories of knowledge in AI and robotics. *New Generation Computing*, 3(4):345–357.
- Rosenthal, D. M., editor (1971). *Materialism and the Mind-Body Problem*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Ross, S. M. (1988). *A First Course in Probability*. Macmillan, London, third edition.

- Rothwell, C. A., Zisserman, A., Mundy, J. L., and Forsyth, D. A. (1993). Efficient model library access by projectively invariant indexing functions. In *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 109–114, Champaign, Illinois. IEEE Computer Society Press.
- Roussel, P. (1975). Prolog: manual de référence et d'utilisation. Technical report, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille.
- Rouveiro, C. and Puget, J.-F. (1989). A simple and general solution for inverting resolution. In *Proceedings of the European Working Session on Learning*, pages 201–210, Porto, Portugal. Pitman.
- Rowe, N. C. (1988). *Artificial intelligence through Prolog*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume 1, chapter 8, pages 318–362. MIT Press, Cambridge, Massachusetts.
- Rumelhart, D. E. and McClelland, J. L., editors (1986). *Parallel Distributed Processing*. MIT Press, Cambridge, Massachusetts. In two volumes.
- Ruspini, E. H., Lowrance, J. D., and Strat, T. M. (1992). Understanding evidential reasoning. *International Journal of Approximate Reasoning*, 6(3):401–424.
- Russell, J. G. B. (1990). Is screening for abdominal aortic aneurysm worthwhile? *Clinical Radiology*, 41:182–184.
- Russell, S. J. (1985). The compleat guide to MRS. Report STAN-CS-85-1080, Computer Science Department, Stanford University.
- Russell, S. J. (1986a). Preliminary steps toward the automation of induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, Pennsylvania. Morgan Kaufmann.
- Russell, S. J. (1986b). A quantitative analysis of analogy by similarity. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, Pennsylvania. Morgan Kaufmann.
- Russell, S. J. (1988). Tree-structured bias. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, volume 2, pages 641–645, St. Paul, Minnesota. Morgan Kaufmann.
- Russell, S. J. (1992). Efficient memory-bounded search methods. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, pages 1–5, Vienna, Austria. Wiley.
- Russell, S. J., Binder, J., and Koller, D. (1994). Adaptive probabilistic networks. Technical Report UCB/CSD-94-824, Computer Science Division, University of California at Berkeley.
- Russell, S. J. and Grosof, B. (1987). A declarative approach to bias in concept learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle, Washington. Morgan Kaufmann.
- Russell, S. J. and Subramanian, D. (1993). Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chambery, France. Morgan Kaufmann.
- Russell, S. J. and Wefald, E. H. (1989). On optimal game-tree search using rational meta-reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 334–340, Detroit, Michigan. Morgan Kaufmann.
- Russell, S. J. and Wefald, E. H. (1991). *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, Massachusetts.
- Ryder, J. L. (1971). Heuristic analysis of large trees as generated in the game of Go. Memo AIM-155, Stanford Artificial Intelligence Project, Computer Science Department, Stanford University, Stanford, California.
- Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135.
- Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 206–214, Tbilisi, Georgia. IJCAII.
- Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior*. Elsevier/North-Holland, Amsterdam, London, New York.

- Sacerdoti, E. D., Fikes, R. E., Reboh, R., Sagalowicz, D., Waldinger, R. J., and Wilber, B. M. (1976). QLISP—a language for the interactive development of complex systems. In *Proceedings of the AFIPS National Computer Conference*, pages 349–356.
- Sachs, J. S. (1967). Recognition memory for syntactic and semantic aspects of connected discourse. *Perception and Psychophysics*, 2:437–442.
- Sacks, E. and Joskowicz, L. (1993). Automated modeling and kinematic simulation of mechanisms. *ComputerAided Design*, 25(2): 106–118.
- Sager, N. (1981). *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Massachusetts.
- Salton, G. (1989). *Automatic Text Processing*. Addison-Wesley.
- Sammut, C., Hurst, S., Kedzier, D., and Michie, D. (1992). Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen. Morgan Kaufmann.
- Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210-229.
- Samuel, A. L. (1967). Some studies in machine learning using the game of checkers II—Recent progress. *IBM Journal of Research and Development*, 11(6):601–617.
- Samuelsson, C. and Rayner, M. (1991). Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 609–615, Sydney. Morgan Kaufmann.
- Saraswat, V. A. (1993). *Concurrent constraint programming*. MIT Press, Cambridge, Massachusetts.
- Savage, L. J. (1954). *The Foundations of Statistics*. Wiley, New York.
- Sayre, K. (1993). Three more flaws in the computational model. Paper presented at the APA (Central Division) Annual Conference, Chicago, Illinois.
- Schabes, Y., Abeille, A., and Joshi, A. K. (1988). Parsing strategies with 'lexicalized' grammars: application to tree adjoining grammars. In Vargha, D., editor, *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, volume 2, pages 578–583, Budapest, Hungary. John von Neumann Society for Computer Science.
- Schaeffer, J., Culberson, J., Treloar, N., and Knight, B. (1992). A world championship caliber checkers program. *Artificial Intelligence*, 53(2-3):273–289.
- Schalkoff, R. I. (1990). *Artificial Intelligence: An Engineering Approach*. McGraw-Hill, New York.
- Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum Associates, Potomac, Maryland.
- Schank, R. C. and Riesbeck, C. K. (1981). *Inside Computer Understanding: Five Programs Plus Miniatures*. Lawrence Erlbaum Associates, Potomac, Maryland.
- Scherl, R. B. and Levesque, H. J. (1993). The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 689–695, Washington, D.C. AAAI Press.
- Schmolze, J. G. and Lipkis, T. A. (1983). Classification in the KL-ONE representation system. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, Karlsruhe, Germany. Morgan Kaufmann.
- Schofield, P. D. A. (1967). Complete solution of the eight puzzle. In Dale, E. and Michie, D., editors, *Machine Intelligence 2*, pages 125–133. Elsevier/North-Holland, Amsterdam, London, New York.
- Schönfinkel, M. (1924). Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316. Translated into English and republished as "On the building blocks of mathematical logic" in (van Heijenoort, 1967, pp. 355–366).
- Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046, Milan, Italy. Morgan Kaufmann.
- Schoppers, M. J. (1989). In defense of reaction plans as caches. *AI Magazine*, 10(4):51–60.
- Schröder, E. (1877). *Der Operationskreis des Logikkalküls*. B. G. Teubner, Leipzig.
- Schwuttke, U. M. (1992). Artificial intelligence for real-time monitoring and control. In *Proceedings of the IEEE International Symposium on Industrial Electronics*, volume 1, pages 290–294, Xian, China.

- Scriven, M. (1953). The mechanical concept of mind. *Mind*, 62:230-240.
- Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge.
- Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, 3:417-457.
- Searle, J. R. (1984). *Minds, Brains and Science*. Harvard University Press, Cambridge, Massachusetts.
- Searle, J. R. (1992). *The Rediscovery of the Mind*. MIT Press, Cambridge, Massachusetts.
- Sejnowski, T. J. and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145-168.
- Selfridge, O. G. (1959). Pandemonium: A paradigm for learning. In Blake, D. V. and Uttley, A. M., editors, *Proceedings of the Symposium on Mechanization of Thought Processes*, pages 511-529, Teddington, United Kingdom. National Physical Laboratory, Her Majesty's Stationery Office.
- Selfridge, O. G. and Neisser, U. (1960). Pattern recognition by machine. *Scientific American*, 203:60-68. Reprinted in Feigenbaum and Feldman (1963).
- Sells, P. (1985). *Lectures on Contemporary Syntactic Theories: An Introduction to Government-Binding Theory, Generalized Phrase Structure Grammar, and Lexical-Functional Grammar*. Center for the Study of Language and Information (CSLI), Stanford, California.
- Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440-446, San Jose, California. AAAI Press.
- Selman, B. and Levesque, H. J. (1993). The complexity of path-based defeasible inheritance. *Artificial Intelligence*, 62(2):303-339.
- Shachter, R. D. (1986). Evaluating influence diagrams. *Operations Research*, 34:871-882.
- Shachter, R. D., D'Ambrosio, B., and Del Favero, B. A. (1990). Symbolic probabilistic inference in belief networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 126-131, Boston, Massachusetts. MIT Press.
- Shachter, R. D. and Peot, M. A. (1989). Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, Windsor, Ontario. Morgan Kaufmann.
- Shachter, R. S. and Kenley, C. R. (1989). Gaussian influence diagrams. *Management Science*, 35(5):527-550.
- Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey.
- Shafer, G. and Pearl, J., editors (1990). *Readings in Uncertain Reasoning*. Morgan Kaufmann, San Mateo, California.
- Shankar, N. (1986). *Proof-Checking Metamathematics*. PhD thesis, Computer Science Department, University of Texas at Austin.
- Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, 41(4):256-275.
- Shannon, C. E. and Weaver, W. (1949). *The Mathematical Theory of Communication*. University of Illinois Press, Urbana.
- Shapiro, E. (1981). An algorithm that infers theories from facts. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, Vancouver, British Columbia. Morgan Kaufmann.
- Shapiro, E. Y. (1983). A subset of Concurrent Prolog and its interpreter. ICOT Technical Report TR-003, Institute for New Generation Computing Technology, Tokyo.
- Shapiro, S. C. (1979). The SNePS semantic network processing system. In Findler, N. V., editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 179-203. Academic Press, New York.
- Shapiro, S. C. (1992). *Encyclopedia of Artificial Intelligence*. Wiley, New York, second edition. Two volumes.
- Sharpies, M., Hogg, D., Hutchinson, C., Torrance, S., and Young, D. (1989). *Computers and Thought: A Practical Introduction to Artificial Intelligence*. MIT Press, Cambridge, Massachusetts.

- Shavlik, J. and Dietterich, T., editors (1990). *Readings in Machine Learning*. Morgan Kaufmann, San Mateo, California.
- Shenoy, P. P. (1989). A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, 3(5):383–411.
- Shieber, S. M. (1986). *An Introduction to Unification-Based Approaches to Grammar*. Center for the Study of Language and Information (CSLI), Stanford, California.
- Shirayanagi, K. (1990). Knowledge representation and its refinement in Go programs. In Marsland, A. T. and Schaeffer, J., editors, *Computers, Chess, and Cognition*, pages 287–300. Springer-Verlag, Berlin.
- Shoham, Y. (1987). Temporal logics in AI: semantical and ontological considerations. *Artificial Intelligence*, 33(1):89–104.
- Shoham, Y. (1988). *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, Cambridge, Massachusetts.
- Shoham, Y. and McDermott, D. (1988). Problems in formal temporal reasoning. *Artificial Intelligence*, 36(1):49–61.
- Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*. Elsevier/North-Holland, Amsterdam, London, New York.
- Shwe, M. and Cooper, G. (1991). An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief network. *Computers and Biomedical Research*, 1991(5):453–475.
- Siekmann, J. and Wrightson, G., editors (1983). *Automation of Reasoning*. Springer-Verlag, Berlin. Two volumes.
- Sietsma, J. and Dow, R. J. F. (1988). Neural net pruning—why and how. In *IEEE International Conference on Neural Networks*, pages 325–333, San Diego. IEEE.
- Siklossy, L. and Dreussi, J. (1973). An efficient robot planner which generates its own procedures. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pages 423–430, Stanford, California. IJCAI.
- Simmons, R., Krotkov, E., Whittaker, W., and Albrecht, B. (1992). Progress towards robotic exploration of extreme terrain. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, 2(2):163–180.
- Simon, H. A. (1958). Rational choice and the structure of the environment. In *Models of Bounded Rationality*, volume 2. MIT Press, Cambridge, Massachusetts.
- Simon, H. A. (1963). Experiments with a heuristic compiler. *Journal of the Association for Computing Machinery*, 10:493–506.
- Simon, H. A. (1981). *The Sciences of the Artificial*. MIT Press, Cambridge, Massachusetts, second edition.
- Simon, H. A. and Newell, A. (1958). Heuristic problem solving: The next advance in operations research. *Operations Research*, 6:1–10. Based on a talk given in 1957.
- Simon, H. A. and Newell, A. (1961). Computer simulation of human thinking and problem solving. *Datamation*, pages 35–37.
- Siskind, J. M. (1994). Lexical acquisition in the presence of noise and homonymy. In *Proceedings of AAAI-94*.
- Skinner, B. F. (1953). *Science and Human Behavior*. Macmillan, London.
- Skolem, T. (1920). Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über die dichte Mengen. *Videnskapselskapets skrifter, I. Matematisk-naturvidenskabelig klasse*, 4.
- Skolem, T. (1928). Über die mathematische Logik. *Norsk matematisk tidsskrift*, 10:125–142.
- Slagle, J. R. (1963a). A heuristic program that solves symbolic integration problems in freshman calculus. *Journal of the Association for Computing Machinery*, 10(4). Reprinted in Feigenbaum and Feldman (1963).
- Slagle, J. R. (1963b). Game trees, $m \& n$ minimaxing, and the $m \& n$ alpha-beta procedure. Artificial Intelligence Group Report 3, University of California, Lawrence Radiation Laboratory, Livermore, California.
- Slagle, J. R. (1971). *Artificial Intelligence: The Heuristic Programming Approach*. McGraw-Hill, New York.

- Slagle, J. R. and Dixon, J. K. (1969). Experiments with some programs that search game trees. *Journal of the Association for Computing Machinery*, 16(2): 189-207.
- Slate, D. J. and Atkin, L. R. (1977). CHESS 4.5—The Northwestern University chess program. In Frey, P. W., editor, *Chess Skill in Man and Machine*, pages 82–118. Springer-Verlag, Berlin.
- Slater, E. (1950). Statistics for the chess computer and the factor of mobility. In *Symposium on Information Theory*, pages 150-152, London. Ministry of Supply.
- Sloman, A. (1978). *The Computer Revolution in Philosophy*. Harvester Press, Hassocks, Sussex.
- Sloman, A. (1985). POPLOG, a multi-purpose multi-language program development environment. In *Artificial Intelligence—Industrial and Commercial Applications. First International Conference*, pages 45-63, London. Queensdale.
- Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071-1088.
- Smith, D. E. (1989). Controlling backward inference. *Artificial Intelligence*, 39(2): 145-208.
- Smith, D. E., Genesereth, M. R., and Ginsberg, M. L. (1986). Controlling recursive inference. *Artificial Intelligence*, 30(3):343-389.
- Smith, D. R. (1990). KIDS: a semiautomatic program development system. *IEEE Transactions on Software Engineering*, 16(9):1024–1043.
- Soderland, S. and Weld, D. (1991). Evaluating non-linear planning. Technical Report TR-91-02-03, University of Washington Department of Computer Science and Engineering, Seattle, Washington.
- Solomonoff, R. J. (1964). A formal theory of inductive inference. *Information and Control*, 7:1–22, 224-254.
- Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University, Stanford, California.
- Spiegelhalter, D., Dawid, P., Lauritzen, S., and Cowell, R. (1993). Bayesian analysis in expert systems. *Statistical Science*, 8:219-282.
- Spiegelhalter, D. J. (1986). Probabilistic reasoning in predictive expert systems. In Kanal, L. N. and Lemmer, J. E., editors, *Uncertainty in Artificial Intelligence*, pages 47-67. Elsevier/North-Holland, Amsterdam, London, New York.
- Spirtes, P., Glymour, C., and Scheines, R. (1993). *Causation, prediction, and search*. Springer-Verlag, Berlin.
- Strivas, M. and Bickford, M. (1990). Formal verification of a pipelined microprocessor. *IEEE Software*, 7(5):52-64.
- Steele, G. (1990). *Common LISP: The Language*. Digital Press, Bedford, Massachusetts, second edition.
- Stefik, M. J. (1981a). Planning and meta-planning. *Artificial Intelligence*, 16:141–169.
- Stefik, M. J. (1981b). Planning with constraints. *Artificial Intelligence*, 16:111–140.
- Sterling, L. and Shapiro, E. (1986). *The Art of Prolog*. MIT Press, Cambridge, Massachusetts.
- Stevens, K. A. (1981). The information content of texture gradients. *Biological Cybernetics*, 42:95-105.
- Stickel, M. E. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1(4):333-355.
- Stickel, M. E. (1988). A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353-380.
- Stockman, G. (1979). A minimax algorithm better than alpha-beta? *Artificial Intelligence*, 12(2): 179-196.
- Stolcke, A. (1993). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Report TR-93-065, ICSI, Berkeley.
- Stone, H. S. and Stone, J. (1986). Efficient search techniques: an empirical study of the *n*-queens problem. Technical Report RC 12057, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.
- Stonebraker, M. (1992). The integration of rule systems and database systems. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):415–423.
- Strachey, C. S. (1952). Logical or non-mathematical programmes. In *Proceedings of the Association for Computing Machinery (ACM)*, pages 46–49, Ontario, Canada.

- Subramanian, D. (1993). Artificial intelligence and conceptual design. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 800-809, Chambery, France. Morgan Kaufmann.
- Subramanian, D. and Feldman, R. (1990). The utility of EBL in recursive domain theories. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, volume 2, pages 942-949, Boston, Massachusetts. MIT Press.
- Subramanian, D. and Wang, E. (1994). Constraint-based kinematic synthesis. In *Proceedings of the International Conference on Qualitative Reasoning*. AAAI Press.
- Sugihara, K. (1984). A necessary and sufficient condition for a picture to represent a polyhedral scene. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(5):578-586.
- Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. Elsevier/North-Holland, Amsterdam, London, New York.
- Sussman, G. J. and McDermott, D. V. (1972). From PLANNER to CONNIVER—a genetic approach. In *Proceedings of the 1972 AFIPS Joint Computer Conference*, pages 1171-1179.
- Sussman, G. J. and Winograd, T. (1970). MICRO-PLANNER Reference Manual. AI Memo 203, MIT AI Lab, Cambridge, Massachusetts.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9-44.
- Swade, D. D. (1993). Redeeming Charles Babbage's mechanical computer. *Scientific American*, 268(2):86-91.
- Tadepalli, P. (1993). Learning from queries and examples with tree-structured bias. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts. Morgan Kaufmann.
- Tait, P. G. (1880). Note on the theory of the "15 puzzle". *Proceedings of the Royal Society of Edinburgh*, 10:664-665.
- Taki, K. (1992). Parallel inference machine PIM. In *Fifth Generation Computer Systems 1992*, volume 1, pages 50-72, Tokyo. IOS Press.
- Tambe, M., Newell, A., and Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5:299-348.
- Tanimoto, S. (1990). *The Elements of Artificial Intelligence Using Common LISP*. Computer Science Press, Rockville, Maryland.
- Tarjan, R. E. (1983). *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM (Society for Industrial and Applied Mathematics), Philadelphia, Pennsylvania.
- Tarski, A. (1935). Die Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261-405.
- Tarski, A. (1956). *Logic, Semantics, Metamathematics: Papers from 1923 to 1938*. Oxford University Press, Oxford.
- Tash, J. K. and Russell, S. J. (1994). Control strategies for a stochastic planner. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Washington. AAAI Press.
- Tate, A. (1975a). Interacting goals and their use. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 215-218, Tbilisi, Georgia. IJCAII.
- Tate, A. (1975b). *Using Goal Structure to Direct Search in a Problem Solver*. PhD thesis, University of Edinburgh, Edinburgh, Scotland.
- Tate, A. (1977). Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 888-893, Cambridge, Massachusetts. IJCAII.
- Tate, A. and Whiter, A. M. (1984). Planning with multiple resource constraints and an application to a naval planning problem. In *Proceedings of the First Conference on AI Applications*, pages 410-416, Denver, Colorado.
- Tatman, J. A. and Shachter, R. D. (1990). Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):365-379.
- Taylor, R. J. (1989). Review of ernst (1961). In Khatib, O., Craig, J. J., and Lozano-Pérez, T., editors, *The Robotics Review J*, pages 121-127. MIT Press, Cambridge, Massachusetts.

- Tenenberg, J. (1988). Abstraction in planning. Technical Report TR250, University of Rochester, Rochester, New York.
- Tennant, H. R., Ross, K. M., Saenz, R. M., and Thompson, C. W. (1983). Menu-based natural language understanding. In *21st Annual Meeting of the Association for Computational Linguistics: Proceedings of the Conference*, pages 151–158, Cambridge, Massachusetts.
- Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4):257–277.
- Tesauro, G. and Sejnowski, T. J. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence*, 39(3):357–390.
- Thomason, R. H., editor (1974). *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, Connecticut.
- Thorne, J., Bratley, P., and Dewar, H. (1968). The syntactic analysis of English by machine. In Michie, D., editor, *Machine Intelligence 3*, pages 281–310. Elsevier/North-Holland, Amsterdam, London, New York.
- Todd, B. S., Stamper, R., and Macpherson, P. (1993). A probabilistic rule-based expert system. *International Journal of Bio-Medical Computing*, 33(2):129–148.
- Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9:137–154.
- Touretzky, D. S. (1986). *The Mathematics of Inheritance Systems*. Pitman and Morgan Kaufmann, London and San Mateo, California.
- Touretzky, D. S., editor (1989). *Advances in Neural Information Processing Systems I*. Morgan Kaufmann, San Mateo, California.
- Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 2nd series*, 42:230–265. Correction published in Vol. 43, pages 544–546.
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59:433–460.
- Turing, A. M., Strachey, C., Bates, M. A., and Bowden, B. V. (1953). Digital computers applied to games. In Bowden, B. V., editor, *Faster Than Thought*, pages 286–310. Pitman, London. Turing is believed to be sole author of the section of this paper that deals with chess.
- Tversky, A. and Kahneman, D. (1982). Causal schemata in judgements under uncertainty. In Kahneman, D., Slovic, P., and Tversky, A., editors, *Judgement Under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge.
- Ueda, K. (1985). Guarded Horn clauses. ICOT Technical Report TR-103, Institute for New Generation Computing Technology, Tokyo.
- Ullman, J. D. (1989). *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Rockville, Maryland.
- Ullman, S. (1979). *The Interpretation of Visual Motion*. MIT Press, Cambridge, Massachusetts.
- Valiant, L. (1984). A theory of the learnable. *Communications of the Association for Computing Machinery*, 27:1134–1142.
- van Benthem, J. (1983). *The Logic of Time*. D. Reidel, Dordrecht, The Netherlands.
- van Benthem, J. (1985). *A Manual of Intensional Logic*. Center for the Study of Language and Information (CSLI), Stanford, California.
- van Harmelen, F. and Bundy, A. (1988). Explanation-based generalisation = partial evaluation. *Artificial Intelligence*, 36(3):401–412.
- van Heijenoort, J., editor (1967). *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, Massachusetts.
- Van Roy, P. L. (1990). Can logic programming execute as fast as imperative programming? Report UCB/CSD 90/600, Computer Science Division, University of California, Berkeley. Ph.D. dissertation.
- VanLehn, K. (1978). Determining the scope of English quantifiers. Technical Report AI-TR-483, MIT AI Lab.
- Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16:264–280.

- Vasconcellos, M. and León, M. (1985). SPANAM and ENGSPLAN: machine translation at the Pan American Health Organization. *Computational Linguistics*, 11(2-3):122-136.
- Veloso, M. and Carbonell, J. (1993). Derivational analogy in PRODIGY: automating case acquisition, storage, and utilization. *Machine Learning*, 10:249-278.
- Vendler, Z. (1967). *Linguistics and Philosophy*. Cornell University Press, Ithaca, New York.
- Vendler, Z. (1968). *Adjectives and Nominalizations*. Mouton, The Hague and Paris.
- Vere, S. A. (1983). Planning in time: windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 5:246-267.
- von Mises, R. (1928). *Wahrscheinlichkeit, Statistik und Wahrheit*. J. Springer, Berlin. Translated into English as von Mises (1957).
- von Mises, R. (1957). *Probability, Statistics, and Truth*. Alien and Unwin, London.
- Von Neumann, J. (1958). *The Computer and the Brain*. Yale University Press, New Haven, Connecticut.
- Von Neumann, J. and Morgenstern, O. (1944). *Theory of games and economic behavior*. Princeton University Press, Princeton, New Jersey, first edition.
- von Winterfeldt, D. and Edwards, W. (1986). *Decision Analysis and Behavioral Research*. Cambridge University Press, Cambridge.
- Voorhees, E. M. (1993). Using WordNet to disambiguate word senses for text retrieval. In *Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 171-80, Pittsburgh, Pennsylvania. Association for Computing Machinery.
- Waibel, A. and Lee, K.-F. (1990). *Readings in Speech Recognition*. Morgan Kaufmann, San Mateo, California.
- Waldinger, R. (1975). Achieving several goals simultaneously. In Elcock, E. W. and Michie, D., editors, *Machine Intelligence 8*, pages 94-138. Ellis Horwood, Chichester, England.
- Waltz, D. (1975). Understanding line drawings of scenes with shadows. In Winston, P. H., editor, *The Psychology of Computer Vision*. McGraw-Hill, New York.
- Wand, M. (1980). Continuation-based program transformation strategies. *Journal of the ACM*, 27(1):174-180.
- Wang, H. (1960). Toward mechanical mathematics. *IBM Journal of Research and Development*, 4:2-22.
- Wanner, E. and Gleitman, L., editors (1982). *Language Acquisition: The State of the Art*. Cambridge University Press.
- Warren, D. H. D. (1974). WARPLAN: a system for generating plans. Department of Computational Logic Memo 76, University of Edinburgh, Edinburgh, Scotland.
- Warren, D. H. D. (1976). Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, pages 344-354.
- Warren, D. H. D. (1983). An abstract Prolog instruction set. Technical Note 309, SRI International, Menlo Park, California.
- Warren, D. H. D., Pereira, L. M., and Pereira, F. (1977). PROLOG: The language and its implementation compared with LISP. *SIGPLAN Notices*, 12(8):109-115.
- Wasserman, P. D. and Oetzel, R. M., editors (1990). *NeuralSource: The Bibliographic Guide to Artificial Neural Networks*. Van Nostrand Reinhold, New York.
- Watkins, C. J. (1989). *Models of Delayed Reinforcement Learning*. PhD thesis, Psychology Department, Cambridge University, Cambridge, United Kingdom.
- Watson, C. S. (1991). Speech-perception aids for hearing-impaired people: current status and needed research. *Journal of the Acoustical Society of America*, 90(2):637-685.
- Webber, B. L. (1983). So what can we talk about now. In Brady, M. and Berwick, R., editors, *Computational Models of Discourse*. MIT Press. Reprinted in Grosz et al. (1986).
- Webber, B. L. (1988). Tense as discourse anaphora. *Computational Linguistics*, 14(2):61-73.
- Webber, B. L. and Nilsson, N. J. (1981). *Readings in Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.

- Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann, San Mateo, California.
- Weizenbaum, J. (1965). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the Association for Computing Machinery*, 9(1):36–45.
- Weizenbaum, J. (1976). *Computer Power and Human Reason*. W. H. Freeman, New York.
- Weld, D. and Etzioni, O. (1994). The first law of robotics: A call to arms. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Washington. AAAI Press.
- Weld, D. S. (1994). An introduction to least commitment planning. *AI Magazine*. To appear.
- Weld, D. S. and de Kleer, J. (1990). *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, California.
- Wellman, M. and Doyle, J. (1992). Modular utility representation for decision-theoretic planning. In *Proceedings, First International Conference on AI Planning Systems*, College Park, Maryland. Morgan Kaufmann.
- Wellman, M. P. (1985). Reasoning about preference models. Technical Report MIT/LCS/TR-340, Laboratory for Computer Science, MIT, Cambridge, Massachusetts. M.S. thesis.
- Wellman, M. P. (1988). *Formulation of Trade-offs in Planning under Uncertainty*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.
- Wellman, M. P. (1990). Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44(3):257–303.
- Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, Massachusetts.
- Wheatstone, C. (1838). On some remarkable, and hitherto unresolved, phenomena of binocular vision. *Philosophical Transactions of the Royal Society of London*, 2:371–394.
- Whitehead, A. N. (1911). *An Introduction to Mathematics*. Williams and Northgate, London.
- Whitehead, A. N. and Russell, B. (1910). *Principia Mathematica*. Cambridge University Press, Cambridge.
- Whorf, B. (1956). *Language, Thought, and Reality*. MIT Press, Cambridge, Massachusetts.
- Widrow, B. (1962). Generalization and information storage in networks of adaline "neurons". In Yovits, M. C., Jacobi, G. T., and Goldstein, G. D., editors, *Self-Organizing Systems 1962*, pages 435–461, Chicago, Illinois. Spartan.
- Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pages 96–104, New York.
- Wiener, N. (1942). The extrapolation, interpolation, and smoothing of stationary time series. OSRD 370, Report to the Services 19, Research Project DIC-6037, MIT.
- Wiener, N. (1948). *Cybernetics*. Wiley, New York.
- Wilensky, R. (1983). *Planning and Understanding*. Addison-Wesley.
- Wilensky, R. (1990). Computability, consciousness, and algorithms. *Behavioral and Brain Sciences*, 13(4):690–691. Peer commentary on Penrose (1990).
- Wilkins, D. E. (1980). Using patterns and plans in chess. *Artificial Intelligence*, 14(2): 165–203.
- Wilkins, D. E. (1986). Hierarchical planning: definition and implementation. In *ECAI '86: Seventh European Conference on Artificial Intelligence*, volume 1, pages 466–478, Brighton, United Kingdom.
- Wilkins, D. E. (1988). *Practical Planning: Extending the AI Planning Paradigm*. Morgan Kaufmann, San Mateo, California.
- Wilkins, D. E. (1990). Can AI planners solve practical problems? *Computational Intelligence*, 6(4):232–246.
- Wilks, Y. (1975). An intelligent analyzer and understander of English. *Communications of the ACM*, 18(5):264–274. Reprinted in Grosz et al. (1986).
- Wilson, R. H. and Schweikard, A. (1992). Assembling polyhedra with single translations. In *IEEE Conference on Robotics and Automation*, pages 2392–2397.

- Winograd, S. and Cowan, J. D. (1963). *Reliable Computation in the Presence of Noise*. MIT Press, Cambridge, Massachusetts.
- Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, 3(1).Reprinted as a book by Academic Press.
- Winograd, T. and Flores, F. (1986). *Understanding Computers and Cognition*. Ablex, Norwood, New Jersey.
- Winston, P. H. (1970). Learning structural descriptions from examples. Technical Report MAC-TR-76, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.PhD dissertation.
- Winston, P. H. (1992). *Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, third edition.
- Wittgenstein, L. (1922). *Tractatus Logico-Philosophicus*. Routledge and Kegan Paul, London, second edition.Reprinted 1971, edited by D. F. Pears and B. F. McGuinness. This edition of the English translation also contains Wittgenstein's original German text on facing pages, as well as Bertrand Russell's introduction to the 1922 edition.
- Wittgenstein, L. (1953). *Philosophical Investigations*. Macmillan, London.
- Woehr, J. (1994). Lotfi visions, part 2. *Dr. Dobbs Journal*, 217.
- Wojciechowski, W. S. and Wojcik, A. S. (1983). Automated design of multiple-valued logic circuits by automated theorem proving techniques. *IEEE Transactions on Computers*, C-32(9):785-798.
- Wojcik, A. S. (1983). Formal design verification of digital systems. In *ACM IEEE 20th Design Automation Conference Proceedings*, pages 228-234, Miami Beach, Florida. IEEE.
- Woods, W. (1972). Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings*.Vol. 42.
- Woods, W. (1978). Semantics and quantification in natural language question answering. In *Advances in Computers*. Academic Press.Reprinted in Grosz et al.(1986).
- Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the Associationfor Computing Machinery*, 13(10):591-606.
- Woods, W. A. (1973). Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings*, volume 42, pages 441-450.
- Woods, W. A. (1975). What's in a link: Foundations for semantic networks. In Bobrow, D. G. and Collins, A. M., editors, *Representation and Understanding: Studies in Cognitive Science*, pages 35-82. Academic Press, New York.
- Wos, L., Carson, D., and Robinson, G. (1964). The unit preference strategy in theorem proving. In *Proceedings of the Fall Joint Computer Conference*, pages 615-621.
- Wos, L., Carson, D., and Robinson, G. (1965). Efficiency and completeness of the set-of-support strategy in theorem proving. *Journal of the Association for Computing Machinery*, 12:536-541.
- Wos, L., Overbeek, R., Lusk, E., and Boyle, J. (1992). *Automated Reasoning: Introduction and Applications*. McGraw-Hill, New York, second edition.
- Wos, L., Robinson, G., Carson, D., and Shalla, L. (1967). The concept of demodulation in theorem proving. *Journal of the Associationfor Computing Machinery*, 14:698-704.
- Wos, L. and Robinson, G. A. (1968). Paramodulation and set of support. In *Proceedings of the IRIA Symposium on Automatic Demonstration*, pages 276-310. Springer-Verlag.
- Wos, L. and Winker, S. (1983). Open questions solved with the assistance of AURA. In Bledsoe, W. W. and Loveland, D. W., editors, *Automated Theorem Proving: After 25 Years: Proceedings of the Special Session of the 89th Annual Meeting of the American Mathematical Society*, pages 71-88, Denver, Colorado. American Mathematical Society.
- Wright, S. (1921). Correlation and causation. *Journal of Agricultural Research*, 20:557-585.
- Wright, S. (1934). The method of path coefficients. *Annals of Mathematical Statistics*, 5:161-215.
- Wu, D. (1993). Estimating probability distributions over hypotheses with variable unification. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 790-795, Chambery, France. Morgan Kaufmann.

- Yang, Q. (1990). Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6:12-24.
- Yarowsky, D. (1992). Word-sense disambiguation using statistical models of Roget's categories trained on large corpora. In *Proceedings of COLING-92*, pages 454–460, Nantes, France.
- Yip, K. M.-K. (1991). *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. MIT Press, Cambridge, Massachusetts.
- Yoshikawa, T. (1990). *Foundations of Robotics: Analysis and Control*. MIT Press, Cambridge, Massachusetts.
- Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3 . *Information and Control*, 10(2): 189-208.
- Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, 8:338-353.
- Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28.
- Zermelo, E. (1976). An application of set theory to the theory of chess-playing. *Firbush News*, 6:37-42. English translation of German paper given at the 5th International Congress of Mathematics, Cambridge, England, in 1912.
- Zilberstein, S. (1993). *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, University of California, Berkeley, California.
- Zimmermann, H.-J. (1991). *Fuzzy Set Theory—And Its Applications*. Kluwer, Dordrecht, The Netherlands, second revised edition.
- Zobrist, A. L. (1970). *Feature Extraction and Representation for Pattern Recognition and the Game of Go*. PhD thesis, University of Wisconsin.
- Zue, V., Seneff, S., Polifroni, J., Phillips, M., Pao, C., Goddeau, D., Glass, J., and Brill, E. (1994). Pegasus: A spoken language interface for on-line air travel planning. In *ARPA Workshop on Human Language Technology*.
- Zuse, K. (1945). The Plankalkül. Report 175, Gesellschaft für Mathematik und Datenverarbeitung, Bonn. Technical report version republished in 1989.

Index

Page numbers in **bold** are definitions of terms and algorithms; page numbers in *italics* are references to the bibliography.

Symbols

- (in parsing), 697
- $S_i \xrightarrow{c} S_j$ (achieves), 347
- \prec (before), 347
- \prec^R (link), 319
- $\{x/C\}$ (substitution), 265
- \sqcup (gap), 711
- α (learning rate), 576
- γ (discount factor), 507
- ϵ -admissible, 106
- ϵ -ball, 553
- ι (uniqueness operator), 196
- A (functional abstraction), 195
- A-expression, 195
- χ^2 (chi squared), 543
- \Rightarrow (implies), 167
- A (and), 166
- \Leftrightarrow (equivalent), 167
- \neg (not), 167
- \vee (or), 166
- \mapsto (uncertain rule), 461
- \succ (determination), 633
- V, 238
- D_p (always), 258
- $\Diamond p$ (eventually), 258
- \models (entailment), 158
- \vdash (derives), 159
- $\exists!$ (exists a unique), 196
- \exists (there exists), 191
- \forall (for all), 190
- n (Intersection), 200
- U (Union), 200
- \in (Member), 200
- C (subset), 200
- $[p, A; q, B]$ (lottery), 473
- \sim (indifferent), 473
- \succ (preferred), 473
- u_T (best prize), 478
- u_{\perp} (worst catastrophe), 478

A

- a_i (activation value), 568
- A*-SEARCH, 97
- A* search, 115
- AAAI (American Association for AI), 28
- Aarup, M., 390, 859
- ABC computer, 14

- Abeille, A., 687, 894
- Abelson, R., 23, 894
- ABO, *see* bounded optimality, asymptotic
- ABSOLVER, 103
- abstraction, 62, 409, 410, 794
- abstraction hierarchy, 380
- abstraction planning, 389
- abstract operator, 376
- abstract solution, 376
- ABSTRIPS, 389
- Abu-Mostafa, Y., 595, 859
- accelerometer, 783
- accessible, *see* environment, accessible
- accusative case, *see* objective case
- Acharya, A., 117, 316, 859, 864
- achievement (in planning), 349, 357
- acknowledgment, 684
- acoustic model, 760, 762–764
 in disambiguation, 682
- ACRONYM, 769
- ACT*, 645
- ACTION, 48
- action, 31, 39
 - conditional, 411
 - cooperative, 180
 - intelligent, 5
 - intermediate-level (ILA), 787
 - low-level, 787
 - partially instantiated, 346
 - rational, 10, 27, 50
 - sensing, 392, 394, 411, 487
 - simultaneous, 216
- action-utility table, 485
- action-value, *see* agent, action-value
- action-value function, 599
- action description, 344
- action model, 510
- action monitoring, 401
- action potential, 564
- activation function, 567
- activation level, 567
- ACTIVE-ADP-AGENT, 608, 608
- active learning, 599
- active vision, 830
- act phase, 314
- actuator, 777
- adaline, 19, 594
- Adams, J., 240
- adaptive control theory, 601
- adaptive dynamic programming, 603, 623
- adaptive network, *see* neural network
- adaptive planning, 389
- ADD-EDGE, 702, 703
- additivity (of utility function), 601
- add list (in STRIPS), 344
- Adelson-Velsky, G., 144
- adjective phrase, 707
- adjoin (to a set), 199
- adjunct, 671
- ADL (planning formalism), 390
- admissible heuristic, *see* heuristic, admissible
- Adorf, H.-M., 390, 878
- ADP, *see* dynamic programming, adaptive
- ADP-UPDATE, 623
- Advice Taker, 18, 22, 179
- agent, 7, 26, 31, 49
 - action-value, 210
 - active, 607
 - architecture, 26, 842
 - autonomous, 153, 625
 - communicating, 683
 - decision-making, 773
 - decision-theoretic, 471, 508–513
 - design, 786
 - forward-chaining, 314
 - function, 500
 - goal-based, 43, 49, 201
 - greedy, 609
 - ideal, 49
 - ideal rational, 33
 - immortal, 507
 - intelligent, 27, 842, 848
 - knowledge-based, 13, 151, 185, 201, 265, 842
- learning, 525
- limited rational, 246
- logical, 213
- model-based, 201
- naive, 658
- omniscient, 182
- passive, 600
- passive ADP, 623
- passive learning, 623
- planning, 337, 337–341, 773
- problem-solving, 55
- program, 35, 37, 49
- propositional logic, 174
- rational, 7, 31, 50, 493, 607
- reflex, 40, 49, 201, 202, 216, 500, 529, 587, 828
- reinforcement learning, 624

- replanning, 410
 situated planning, 403
 software, 36
 taxi-driving, 39, 507, 526, 843
 tropistic, 201
 uncooperative, 658
 utility-based, 44, 49, 508
 vacuum, 32, 51–52, 58
 wacky, 609
 wumpus world, 174–177,
 201–203, 662
 agent-based software engineering,
 219
 agent function, 411
 aggregation (in planning), 409
 Agmon, S., 594, 859, 887
 agrammatical strings, 712
 Agre, P. E., 411, 859
 agreement (in a sentence), 668
 Aho, A. V., 853, 859
 AI, *see* artificial intelligence
 Aiken, H., 14
 aircraft carrier scheduling, 371, 390
 airport, driving to, 415
 airport siting, 480, 484
 AİSB, 28
 Ait-Kaci, H., 328, 329, 859
 AI Winter, 25
 al-Khowarazmi, 11
 Alami, R., 788, 873
 Alberti, 726, 768
 Albrecht, B., 778, 896
 alchemy, 4
 algorithm, 11
 algorithmic complexity, *see*
 Kolmogorov complexity
 Alhazen, 768
 Aliens (movie), 777
 ALIGN, 754, **754**
 alignment method, 752
 ALIVE, 36
 ALL-RELATED-TO?, 322, **322**
 ALL-RELS-IN, 321
 ALL-RELS-OUT, 321, 322
 Allais, M., 479, 859
 Allen, B., 390, 871
 Alien, J., 28, 259, 364, 688, 859
 Alien, R., 595, 859
 Alien, W., 94, 760
 Almanac Game, 495
 Almuallim, H., 646, 859
 Almulla, M., 117, 889
 Aloimonos, J., 770, 859
 alpha-beta, *see* search, alpha-beta
 alpha-beta pruning, 130, 146, 844
 Alshawi, H., 685, 686, 859
 Alspector, J., 595, 859
 Alterman, R., 389, 859
 altruism, 419
 Alvey report, 24
 ALVINN, 586, 587
 AM, 646, 647
 Amarel, S., 67, 86, 859
 ambiguity, 658, 680–682, 712–715
 Ambros-Ingeron, J., 411, 860
 Amit, D., 595, 860
 Ammon, K., 826, 860
 AMORD, 330
 analogical reasoning, 646
 ANALOGY, 19, 29
 analysis of algorithms, 851
 Analytical Engine, 15, 142, 823
 Anantharaman, T. S., 144, 876
 anaphora, 679
 And-Elimination, 172
 And-Introduction, 172
 Andersen, S., 465, 860
 Anderson, A. R., 839, 840, 860
 Anderson, J., 13, 28, 162, 316, 595,
 645, 860, 875
 annealing, *see* search, simulated
 annealing, 113
 antecedent, 167
 antelope, 658
 anytime algorithm, 844
 aphasia, 565
 APN, *see* probabilistic network,
 adaptive
 apparent motion, 735
 Appelt, D., 258, 694, 720, 874, 876
 APPEND, 395, 402
 applicable operator, 344
 APPLY, 126
 apposition, 707
 approximation hierarchy, 380, 390
 Apte, C., 325, 884
 Arbuckle, T., 862
 arc consistency, 84
 architecture, 35, 786
 agent, 26, 842
 blackboard, 770
 cognitive, 316
 for speech recognition, 25
 hybrid, 137
 open planning, 369
 parallel, 117
 real-time, 329
 robot, 786–790
 rule-based, 316
 subsumption, 411
 types, 50
 Arentoft, M., 390, 859
 ARGS, 299, 303
 argument from disability, 823
 argument from informality, 826
 Aristotle, 6, 9, 179, 212, 257, 564,
 838
 Arlabosse, F., 27, 879
 Arlazarov, V., 144, 859
 Armstrong, D., 839, 860
 Arnauld, A., 10, 471, 493, 860
 article, 664
 artificial insemination, 834
 artificial intelligence, 3, 17
 applications, 26
 conferences, 28
 definition, 4, 29
 foundations, 615
 history of, 8–26, 28
 industry, 24, 28
 journals, 28
 programming language, 18
 as rational agent design, 7
 real-time, 843
 societies, 28
 strong, **29, 818**
 subfields, 4
 weak, 29, 818
 artificial sweeteners, 834
 artificial urea, 834
 asbestos removal, 480
 Ashby, W. R., 594, 860
 Asimov, I., 814, 848, 860
 ASK, 152, 153, 201, 211, 214, 298,
 299, 323, 325, 342, 629,
 660, 721
 assertion (logical), 201
 assignment (in an ATN), 686
 associative memory, 571
 assumption, 326
 Astrom, K., 520, 860
 astronomer, 468
 asymptotic analysis, 852
 Atanasoff, J., 14
 Atkeson, C., 623, 886
 Atkin, L. R., 87, 897
 ATMS, *see* truth maintenance
 system, assumption-based
 ATN, *see* augmented transition
 network
 atom, 667
 atomic event, 425
 atomic sentence, *see* sentence,
 atomic
 attention, 719
 augmentation, 667, **669, 677, 684**
 augmented transition network, 686
 AURA, 297, 313, 330
 Austin, J., 685, 860
 AUTOCLASS, 647
 autoepistemic logic, 466
 auto insurance, 469
 automata, 838, 849

- automated debugging, 646
 automated reasoners, *see* theorem provers
 automatic pilot, 329
 automatic programming, 400, 400
 automotive industry, 774
 autonomous vehicle, 26, 517, 587, 775
 underwater (AUV), 775
 autonomy, 35, 39, 49, 773
 avoiding obstacles, 725
 axiom, 198, 222
 domain closure, 254
 effect, 205
 frame, 206
 independent, 198
 successor-state, 206
 usable (in OTTER), 310
 axon, 564
- B
- b^* (branching factor), 102
 B^* search, 143
b-tree, 704
 Babbage, C., 15, 823
 Bacchus, E., 432, 519, 860
 Bach, E., 259, 860
 bachelor, 231
 Bachmann, P., 853, 860
 BACK-CHAIN, 275, 275, 305, 306
 BACK-CHAIN-LIST, 275, 275
 BACK-PROP-UPDATE, 581
 back-propagation, 21, 24, 578, 578–583, 593, 595
 backed-up value, 108
 backgammon, 133–135, 139, 144, 615
 background, 828, 830
 background assumptions, 735
 background knowledge, 282, 626
 backjumping, 309
 backprojection, 806
 backtracking, 84, 771
 chronological, 309
 dependency-directed, 309
 Backus-Naur form, 186, 655, 667, 854
 backward chaining, 272, 275, 305, 313, 447
 Bacon, R., 9
 Bain, M., 466, 860
 Bajcsy, R., 769, 770, 860
 Baker, C. L., 28, 685, 860
 Baker, J., 770, 860
 Ballard, B., 144, 325, 860, 868
 bandit problem, 610
 Bandyopadhyay, A., 770, 859
 Banerji, R., 552, 559, 886
 bang-bang control, 618
 Bar-Hillel, Y., 685, 720, 860
 Bar-Shalom, Y., 520, 860, 872
 bar codes, 805
 Ban, A., 28, 861
 Barrett, A., 390, 867
 Barrett, R., 330, 861
 Barstow, D. R., 330, 861
 Barto, A., 623, 861
 Barwise, J., 181, 213, 861
 BASE, 672
 baseline, 739
 Bates, M., 16, 142, 899
 Battell, J. S., 141, 874
 Baum, L., 770, 861
 Baum-Welch algorithm, 766
 Bayes rule, 426, 426–427, 431, 434, 434, 436, 449, 760
 Bayes, T., 426, 431, 861
 Bayesian learning, *see* learning, Bayesian
 Bayesian network, *see* belief network
 Bayesian updating, 428, 431, 434, 435, 510
 Beal, D., 143, 867
 Beck, H., 325, 861
 bee, 651
 beer factory scheduling, 371
 beetle, *see* dung beetle
 behavior, *see* action
 behavior-based robotics, *see* robotics
 behavioral module, 789
 behaviorism, 13, 15, 50
 \widehat{Bel} (belief about state), 509
 \widehat{Bel} (prediction), 509
 Belhumeur, P., 740, 867
 belief, 157, 243, 820
 degree of, 424
 BELIEF-NET-ASK, 446, 447, 452
 BELIEF-NET-TELL, 446
 belief function, 462
 belief network, 25, 436, 464, 498, 712, 720
 dynamic, 514, 519
 vs. neural network, 592
 sigmoid, 596
 belief network approximation, 453
 belief network construction, 440
 belief network inference, 445–456
 complexity, 453
 belief network learning, 531
 Bell, C., 390, 861
 Bell, J., 213, 861
 Belle (chess computer), 137
 Bellman, R. E., 5, 86, 503, 504, 520, 861
 Belsky, M. S., 862
 benchmarking, 851
 Bendix, P., 293, 879
 Bennett, J., 313, 330, 874
 Berlekamp, E., 143, 861
 Berliner, H. J., 26, 117, 137, 143–145, 861, 862
 Bernoulli, D., 494, 862
 Bernoulli, J., 12, 432, 476
 Bernstein, A., 143, 862
 Berry, C., 14
 Berry, D. A., 623, 862
 Bertsekas, D. P., 520, 862
 Berwick, R., 690
 BEST-FIRST-SEARCH, 92, 93, 93, 97
 best-first search, 92, 115
 best possible prize, 478
 Beth, E. W., 292, 862
 betting game, 424
 Bezzel, M., 86
 bias, 529
 declarative, 636
 Bibel, W., 293, 390, 862
 Bickford, M., 313, 897
 biconditional, 167
 bidirectional search, 85, 89
 bigram model, 760, 765
 Binder, J., 596, 893
 binding list, 201, 265
 BINDINGS, 347, 372, 374
 binocular stereopsis, 737–742, 750, 768
 binomial nomenclature, 258
 biological naturalism, 819, 834
 Birnbaum, L., 687, 862
 Biro, J., 840, 862
 Birtwistle, G., 331, 862
 bit (of information), 540
 Bitman, A., 144, 859
 Bitner, J., 116, 862
 BKG, 139, 144
 Black, E., 687, 862
 black-box, 625
 blackboard architecture, 770
 Blenko, T., 258, 876
 blind search, 73
 Block, N., 839, 840, 862
 blocking (in belief networks), 444
 blocks world, 19, 23, 260, 359, 382, 383, 404, 827
 Bloom, P., 687, 862
 Blum, A., 595, 862
 Blumberg, B., 36, 883
 Blumer, A., 560, 862
 BMTIP, 826
 BNF, *see* Backus-Naur form
 BO, *see* bounded optimality
 Board, R., 560, 862

- Bobrow, D., 19, 646, 862, 863
 Boddy, M., 844, 868
 Boden, M. A., 840, 863
 body (of a clause), 305
 Boltzmann machine, 571, 595
 Boole, G., 11, 179, 212, 291, 863
 Boolean connective, *see* connective,
 Boolean
 Boolean function, 570
 Boolean logic, 179
 Boolos, G. S., 293, 825, 863
 Booth, A. D., 720, 882
 Booth, T., 687, 871
 Borgida, A., 324, 332, 863
 Boser, B., 586, 595, 881
 BOTTOM-UP-PARSE, 666, **666**, 697
 boundary set, 549
 bounded cutset conditioning, 465
 bounded optimality, 845
 asymptotic (ABO), 846
 Bowden, B., 16, 142, 899
 BOXES, 618
 Boyer, R. S., 293, 313, 328, 330,
 826, 863
 Boyer-Moore theorem prover, 313,
 330
 Boyle, J., 294, 902
 BP (British Petroleum), 539
 Brachman, R., 261, 324, 325, 331,
 332, 863, 868, 882
 Bradshaw, G. L., 881
 Bradtke, S., 623, 861
 Brady, D., 595, 877
 Brady, J., 888
 brain, 3, 9, 16, 563
 aphasia, 565
 computational power, 566
 cortex, 565
 damage, optimal, 572
 prosthesis, 835, 841
 as recurrent network, 570
 states, 819
 super, 4, 12
 in a vat, 821
 vs. computer, 565-566
 brains cause minds, 565, 819
 branch and bound, 116
 branching factor, 74, 632
 effective, 102, 131
 Bransford, J., 722, 863
 Bratko, I., 330, 644, 863
 Bratley, P., 686, 899
 Bratman, M. E., 839, 863
 BREADTH-FIRST-SEARCH, 74
 breadth-first search, 74, 85
 Breese, J., 50, 495, 876
 Brelaz, D., 116, 863
 Bresnan, J., 687, 863
 bridge (card game), 29
 bridges, 800
 Briggs, R., 256, 863
 brightness, 729
 Brill, E., 903
 Broca, P., 565
 Brooks, R., 411, 769, 789, 812, 863
 Brouwer, P., 623, 861
 Brown, A., 363
 Brown, J. S., 260, 647, 868, 882
 Brudno, A., 143
 Brudno, A. L., 863
 Brunelleschi, 768
 Bryson, A., 21, 578, 595, 863
 Buchanan, B., 22, 94, 257, 466, 552,
 559, 863, 864, 870, 882
 Buchler, J., 864
 buffalo, 690
 BUILD, 260, 330
 bunch, 234
 Bundy, A., 330, 331, 645, 864, 899
 Bunt, H. C., 258, 864
 Buntine, W., 646, 887
 burglaralarm, 437-438
 Burstall, R., 258, 330, 864
 Bylander, T., 363, 864
- C**
- C (configuration space), 791
 C-BURIDAN, 411
 C4.5, 559
 Caianello, E., 594, 864
 CALL, 307
 Cambefort, Y., 50, 874
 Camembert cheese, 221
 Cameron-Jones, R., 644, 891
 Campbell, M. S., 144, 876
 Campbell, P. K., 832, 864
 candidate definition, 544
 candidate elimination, 549
 can machines think, 822, 831
 Canny, J., 733, 796, 811, 864
 canonical distribution, 443
 canonical form, 270
 Cantor, 826
 Cao, E., 646, 887
 Capek, J., 810
 Capek, K., 809
 Carbonell, J., 560, 646, 864, 885,
 900
 Cardano, G., 12, 431
 Carnap, R., 424, 430, 432, 864
 Carnegie Mellon University, 17,
 138, 586, 707
 Carson, D., 293, 902
 cart-pole problem, 617
 case (of nouns), 668
 ease-based planning, 389
 Casimir, A., 907
 Cassandra, A., 520, 864
 CATEGORY, 666
 category, 228, **229**, 317, 323
 category theory (not), 228
 Caterpillar English, 692
 causal influence, 441
 causal link, 347, 350, 368, 375
 causal network, *see* belief network
 causal rules, 209
 causal support, 448
 causal theory, 254
 causation, 169, 209, 429
 caveman, 626
 CCD, *see* charge-coupled device
 cell decomposition, 796, 809, 812
 exact, 797
 cell layout, 69
 Census Bureau, 301
 cerebral cortex, *see* brain
 certainty equivalent, 478
 certainty factor, 23, 461, 466
 CFG, *see* grammar, context-free
 chaining, 280
 chain problem, 89
 Chakrabarti, P., 117, 864
 Chambers, R., 618, 622, 885
 chance node (decision network), 484
 chance node (game tree), 133
 chance of winning, 127
 Chang, C.-L., 294, 864
 Chang, K., 465, 871
 change, 203, 234
 continuous, 237
 dealing with, 177
 channel routing, 69
 Chapman, D., 25, 363, 390, 411,
 859, 864
 Chapuis, A., 810, 864
 charge-coupled device, 727, 772
 Charniak, E., 5, 23, 328, 331, 688,
 720, 864, 865, 873
 chart, 697
 CHART-PARSE, 702
 CHAT, 693, 694, 720
 Chateau Latour, 834
 Chatila, R., 788, 873
 checkers, 18, 138, 142, 148, 559,
 823
 cheese factory (Camembert), 221
 Cheeseman, P., 25, 467, 865
 Chellas, B. F., 261, 865
 chemical transmitter, 564
 chemistry, 22
 Chen, K., 145, 879
 Cherniak, C., 50, 865
 Chernobyl, 775
 Chervonenkis, A., 560, 899

- chess, 15, 20, 26, 122, 137-138, 412, 820
 automaton, 141
 program, 16
 ratings, 137
CHESS 4.5, 87
 χ^2 pruning, 543
 Chickering, M., 596, 875
 Chierchia, G., 28, 685, 865
CHILDREN, 666
 chimpanzees, 651
 Chinese room, 831-834, 840
 Chinook, 138, 148
 Chitrao, M., 688, 865
 choice point, 306, 309
 Chomsky, N., 15, 653, 656, 685, 686, 865
 choose, 355, 855
CHOOSE-ATTRIBUTE, 537, 540, 541
CHOOSE-BEST-ACTION, 38
CHOOSE-BEST-CONTINUATION, 402
CHOOSE-DECOMPOSITION, 374, 379
CHOOSE-LITERAL, 642-644
CHOOSE-OPERATOR, 356, **356**, 357, 358, 374, 379, 382, 384, 385, **385**
 Christmas, 831
 chronological backtracking, 326
cHUGIN, 465
 Chung, K. L., 433, 865
 Church, A., 11, 292, 329, 865
 Church, K., 688, 702, 720, 865
 Churchland, P. M., 839, 840, 865
 Churchland, P. S., 836, 839, 840, 865
CIGOL, 646
 circuit verification, 226
circumscription, **459**, 466
 city block distance, 102
 clarification, 680
 Clark, K., 329, 466, 865
 Clark, R., 687, 865
 Clarke, A., 465
 Clarke, M., 144, 865
 class, *see* category
CLASSIC, 298, 325
 classification (in description logic), 323
 classification (of examples), 534
 class probability, 561
 clause, 182
CLINT, 646
 Clinton, H., 308
CLIPS, 298
 clobbering, 353
 clock, 33
 Clocksin, W., 330, 865
CLOSE-ENOUGH, 504
 closed class, 664
 Clowes, M. B., 746, 769, 865
CLP, *see* logic programming, constraint
CLP(R), 329
 clustering, 453, 465
CMU, *see* Carnegie Mellon University
CNF, *see* conjunctive normal form
CNLP, 411
 coarticulation, 762, 765
 Cobham, A., 11, 865, 869
 coercion, **409**, 410, 411
 cognitive architecture, 316
 cognitive psychology, 13
 cognitive robotics, 259
 cognitive science, 6, 13, 17, 28
 Cohen, J., 328, 810, 865, 866
 Cohen, P., 720, 866
 coherence relation, 717
 coin flip, 383, 461, 462, 476, 540
 collection, *see* category
 collective computation, *see* neural network
 Collins, A., 646, 864
 Colmerauer, A., 328, 329, 685, 866
 Colomb, R., 328, 866
 Colossus, 14
 command, 683
 commitment
 epistemological, 165
 ontological, 165, 185, 417, 459
 premature, 381
 common ground, 661
 Common Lisp, *see* Lisp
 common sense, 458
 commonsense ontology, 317
 commonsense summer, 258
 communication, 161, 254, 651-685
 competitive ratio, 808
 compilation, 307, 379, 788
 compilation time, 309
 complement (of a verb), 670, 710
COMPLETE?, 702
 completeness
 of a plan, 349
 of a proof procedure, 160, 178, 277
 of resolution, 286-290
 of a search algorithm, 73, 85
 theorem, 277
 complete plan, *see* plan, complete
 complete proof procedure, 277
COMPLETER, 698-701, **702**, 702, 703
 completer, 698
 complexity
 sample, 554
 space, 73, 85
 time, 73, 85
 complexity analysis, 74, **852**
 complex sentence, *see* sentence, complex
 complex term, 211
 compliant motion, 784, 795
COMPOSE, 273, 303
 composite decision process, 116
 composite object, *see* object, composite, 251
 composition (of substitutions), 273
 compositionality, 163, 672
COMPOUND, 299
COMPOUND-ACTION, 663
COMPOUND?, 303
 compounding, 703
 compression, 615
 computability, 11
 computational learning theory, **553**, 557, 560, 595
 computer, 4, 14-15
 vs. brain, 565-566
 Computers in Biomedicine, 23
 computer vision, *see* vision
 concept, *see* category
 Conceptual Graphs, 298
 concerto, 830
 conclusion (of an implication), 167
 condition-action rule, *see* rule
CONDITION-PART, 395
CONDITIONAL-PLANNING-AGENT, 395
CONDITIONAL?, 395
 conditional effect, *see* effect, conditional
 conditional independence, **429**, 431, 434, 449, 464, 509
 conditional link, 396
 conditional planner, *see* planning, conditional
 conditional probability, *see* probability, conditional
 conditional probability table, 438
 conditional step, 396
 conditioning (in belief networks), **453**
 conditioning (in planning), 398
 conditioning case, 438
 Condon, E., 142, 866
 Condon, J., 137, 144, 866
 configuration space, **791**, 790-796, 809
 generalized, 792
 configuration space obstacle, 791
 confirmation theory, **10**, 432
 conflict resolution, 298, 314, 315
 confrontation, 382

- conjunct, 166
 conjunction, 166
 conjunction (natural language), 664
 conjunctive normal form, 182, **278**,
 282, 290, 292
 conjunctive queries, 309
 connectionism, *see* neural network
 connection method, 293
 connective
 Boolean, 165
 logical, 16, 165, 189
 CONNIVER, 330
 consciousness, 9, 29, 564, 823,
 831–836
 consequent, 167
 consistency, 116, 277, 545
 CONSISTENT, 356, 358, 382, 385
 CONSISTENT-DET?, **635**, 635
 consistent decomposition, 373
 consistent plan, *see* plan, consistent
 conspiracy number, 143
 constant symbol, 186, 188, 211
 constraint, 83
 binary, 83
 temporal, 370
 time, 368
 unary, 83
 constraint logic programming, *see*
 logic programming
 constraint propagation, 84
 constraint satisfaction, 19, 65,
 83–84, 91, 328, 388
 constraint satisfaction problem, 83
 heuristics for, 104
 construction industry, 775
 constructive induction, *see*
 induction, constructive
 CONTENTS, 663
 context (in planning), 395
 context-free grammar, *see* grammar,
 context-free
 context-sensitive grammar, *see*
 grammar, context-sensitive
 contingency, 392
 contingency problem, *see* problem.
 contingency
 continuation, 307
 continuity (of preferences), 474
 continuous, *see* environment,
 continuous
 contour, 117, 735, 745–749
 contour (of a state space), 98
 contradiction, 639
 contrapositive, 312
 control
 lateral, 749
 longitudinal, 750
 control systems, 463
 control theory, 364, 498, 510, 539,
 594, 617, 780
 adaptive, **601**, 622
 control uncertainty, 803
 conventional signs, 651
 conversion to normal form, 281–282
 CONVINCE, 465
 convolution, 731, 771
 Conway, J., 143, 861
 Cook, S., 12, 173, 853, 866
 Cooper, G., 465, 596, 866, 876, 896
 coordinate frame, 734
 Copeland, J., 258, 840, 866
 Core Language Engine, 685
 Cormen, T. H., 853, 866
 corpus, 704
 correspondence theory, 821
 count noun, 242
 Covington, M. A., 688, 866
 Cowan, J., 19, 596, 866, 902
 Cowell, R., 596, 897
 Cox, R., 424, 432, 866
 CPOP, 394, 395, 398, **399**, 401,
 406, 411
 CPSC, 444, 456
 CPT, *see* conditional probability
 table
 crack, 746
 Cragg, B., 594, 866
 Craik, K. J., 13, 180, 866
 Cray Blitz, 144
 creativity, 15, 817, 823
 Cresswell, M., 261, 877
 Crevier, D., 28, 866
 crisis management, 775
 critic (in learning), **526**, 562
 critic (in planning), 379
 criticality level, 380
 critical point, 798, 801
 Crocker, S., 143, 873
 Crockett, L., 207, 866
 Croft, B., 258, 876
 Croft, W., 685, 876
 cross-correlation, 735, 740
 cross-indexing, 302
 cross-over, 620
 cross-validation, **543**, 573, 597
 cross ratio, 754
 crossword puzzle, 118
 cryptarithmic problem, 65, 91
 CSP, *see* constraint satisfaction
 problem
 Culberson, J., 142, 894
 Cullingford, R., 23, 866
 cult of computationalism, 818
 cumulative learning, 638, 644
 curiosity, 612
 current-best-hypothesis, 546, 559,
 576
 CURRENT-BEST-LEARNING, 547,
 547, 548, 555
 Currie, K., 369, 390, 866, 871
 Curry, H. B., 329, 867, 894
 CUTOFF-TEST, 126, 132
 cutset, 455
 cutset conditioning, 454
 bounded, 455
 Cybenko, G., 595, 867
 CYC, 258, 317, 828, 844
- D
- D'Ambrosio, B., 465, 895
 d-separation, **444**, 449
 DAG, *see* directed acyclic graph
 Dagum, P., 465, 867
 Dahl, O.-J., 331, 862, 867
 DALTON, 647
 Daniels, C., 117, 883
 DANTE-II, 776
 Dantzig, G., 12, 867
 Darlington, J., 330, 864
 Darrell, T., 36, 883
 Dartmouth workshop, 17
 Darwiche, A., 465, 867
 Darwin, C., 619
 data-directed inference, 274
 data-driven inference, 274
 database, 328, 693–694
 data extraction, 696
 data fusion, 512
 dative case, 668
 Davidson, D., 259, 867
 Davies, T., 633, 646, 685, 867, 876
 Davies, T. R., 646
 Da Vinci, L., 768
 Davis, E., 182, 258, 260, 261, 825,
 867
 Davis, M., 286, 292, 867
 Davis, R., 330, 646, 867
 Dawid, P., 596, 897
 Dayan, P., 604, 867
 DBN, *see* belief network, dynamic
 DCG, *see* grammar, definite clause
 DDN, *see* decision network,
 dynamic
 deadlines, 368
 Dean, M., 313, 888
 Dean, T., 364, 389, 520, 844, 868
 Debreu, G., 483, 868
 debugging, 222
 Dechter, R., 99, 868
 decision
 one-shot, 472
 problem, 11
 rational, 416, 471, 491

- sequential, 472, 487, **498**, 510, 603
 under uncertainty, 418
- DECISION-LIST-LEARNING, 556, 556, 557
- DECISION-THEORETIC-AGENT, 508, **511**
- DECISION-TREE-LEARNING, 535, 537, **537**, 538, 542, 543, 545, 557, 561, 562, 635, 636, 638
- decision analysis, 491
- decision analyst, 491
- decision cycle, 508
- decision list, 555
- decision maker, 491
- decision network, 436, 465, 471, **484**, 484–486, 493, 494
- dynamic, 516, 519
 evaluation, 517
 evaluation, 486
- decision node, 485
- decision theory, 12, 25, **419**, 493
- decision tree, 494, 531, 531
 expressiveness, 532
 learning, 530, 559
 pruning, 542
- declarative approach, 304
- declarative bias, *see* bias, declarative
- declarativism, **153**, 219
- decomposability (of lotteries), 474
- decomposition, 390
- de Dombal, F. T., 432, 433, 867
- deduction, 163
- Deep Blue, 138
- deep structure, 686
- Deep Thought, 138, 144
- default logic, *see* logic, default
- default reasoning, *see* reasoning, default
- default value, 229, 319
- deferring, *see* planning, deferred
- de Finetti, B., 423, 432, 435, 867
- definite clause grammar, *see* grammar, definite clause
- definition (logical), 198
- definitions, 323
- degree of belief, 417
 interval-valued, 459
- degrees of freedom, **777**, 778, 781, 791
- de Groot, A., 142, 867
- DeGroot, M. H., 433, 868
- DeJong, G., 645, 868
- de Kleer, J., 260, 328, 330, 332, 867, 868, 871, 901
- delete list (in STRIPS), 344
- Del Favero, B., 465, 895
- deliberate ambiguity, 682
- deliberation, 403, 843
- demodulation, 284, **284**, 293
- demodulator, 310, 333
- De Morgan, A., 212, 868
- De Morgan rules, 193
- demotion, 353
- Dempster's rule, 462
- Dempster, A., 462, 466, 596, 868
- Dempster-Shafer theory, 457, 459, 462, 466
- Den* (denotation), 245
- DENDRAE, 22–23, 257
- dendrite, 564
- Denker, A., 26, 138
- Denker, J., 585, 586, 595, 621, 881
- Dennett, D. C., 50, 817, 820, 834, 840, 847, 868
- Deo, N., 87, 868
- DEPTH, 72, 75
- depth (of a node), 72
- DEPTH-FIRST-SEARCH, 78, 91
- depth-first search, 77, 85, 305
- DEPTH-LIMITED-SEARCH, 79
- depth-limited search, 78, 85
- depth limit, 129
- depth of field, 727
- De Raedt, L., 868
- derived sentences, 160
- Derr, W., 142, 866
- de Sarkar, S., 117, 864
- Descartes, R., 9, 10, 768, 838
- Descoite, Y., 390, 868
- description logic, *see* logic, description, 331
- descriptive theory, 479
- desire, 820
- detachment, 460, 460
- determination, **633**, 647
 minimal, 635
- determinations, 646
- determiner, **708**, **708**
- deterministic, *see* environment, deterministic
- deterministic node, 443
- Devanbu, P., 325, 868
- DEVISER, 387, 389
- Devol, G., 774, 810
- Dewar, H., 686, 899
- Dewey Decimal system, 230
- DFS-CONTOUR, 106, **107**, 107
- diachronic, 203
- diagnosis, 416
 dental, 416
 medical, 209, 443, 461, 487, 848
- diagnostic inference, *see* inference, diagnostic
- diagnostic rules, **209**
- diameter (of a state space), 78
- Dickmanns, E., 587, 750, 770, 869
- dictionary, 21, **251**, **704**
- Dietterich, T., 560, 859, 869, 896
- differentiation, 629
- diffusely reflected light, 729
- Digital Equipment Corporation, 24, 316
- Dijkstra, E. W., 87, 869
- Dill, D., 313, 888
- Dincbas, M., 330, 869
- Dingwell, W. O., 653, 869
- directed acyclic graph, 437, 465, 570
- disabilities, 840
- DISAMBIGUATE, 662, 663
- disambiguation, 658, **673**, 678, 680–682, 685, 712
- discontinuities, 730
- discontinuous function, 571
- discount factor, **507**, 624
- discounting, 507, 519, 520
- discourse, 715
 coherent, 717–720
 understanding, 715–719
- DISCOURSE-UNDERSTANDING, 715
- discrete, *see* environment, discrete
- discrete event, *see* event, discrete
- discretization, 544
- discrimination net, 559
- dish, 250
- disjoint sets, 231
- disjunct, 166
- disjunction, 166
- disjunctive effects, 409
- disjunctive normal form, 292
- disparity, 737, 738
- distributed encoding, 577
- divide-and-conquer, 341, 379, 407
- Dixon, J. K., 143, 897
- DNA, 619
- DO, 663
- dolphins, 651
- domain (in a CSP), 83
- domain (in knowledge representation), 197
- domain (of a random variable), 420
- domain constraint, 785
- dominance
 stochastic, 481
 strict, 481
- domination (of heuristics), 102
- Donskoy, M., 144, 859
- Doran, J., 86, 115, 117, 869
- Double-Negation Elimination, 172
- Dow, R. J. F., 595, 896
- Dow Jones, 696
- downward solution, **376**, **389**, **391**
- Dowty, D., 685, 869

- Doyle, J., 50, 330, 332, 459, 494,
 868, 869, 885, 901
 Drabble, B., 390, 869
 Draper, D., 411, 869, 870
 Dreussi, J., 389, 896
 Dreyfus, H. L., 9, 817, 818, 827,
 828, 869
 Dreyfus, S. E., 86, 87, 504, 520,
 827, 828, 861, 869
 drilling rights, 487
 driving
 automated, 29, 586
 dropping conditions, 548
 Droz, E., 810, 864
 DT-AGENT, 419
 dualism, 9, 838
 Dubois, D., 466, 869
 duck, mechanical, 810
 Duda, R., 23, 466, 869
 dung beetle, 35, 50, 203
 Dunham, B., 21, 871
 du Pont, 24
 Dürer, A., 768
 Durrant-Whyte, H., 520, 882
 Dyer, M., 23, 869
 DYLAN, 329
 dynamic, *see* environment, dynamic
 dynamical systems, 520
 dynamic belief network, *see* belief
 network, dynamic
 dynamic belief networks, 520
 dynamic decision network, *see*
 decision network, dynamic
 dynamic logic, 261
 dynamic programming, 87, 116,
 503, 520, 603, 623, 697,
 742, 765
 adaptive, 603, 623
 dynamic weighting, 116
 Dzeroski, S., 869, 879
 Dzeroski, S., 646
- E**
- E(cat, i)* (event), 236
 \mathcal{E}_0 (English subset), 662
 \mathcal{E}_1 (English subset), 670
 \mathcal{E}_2 (English subset), 680
 Earley, J., 720, 869
 earthquake, 437
 Eastlake, D., 143, 873
 Ebeling, C., 137, 144, 862, 869
 EBL, *see* explanation-based learning
 Eckert, J., 14
 economics, 50, 476, 507, 847
 edge (in a chart), 697
 edge (in an image), 730
 edge (in a scene), 745
 edge detection, 730, 733
- Edinburgh, 646, 810
 Edmonds, D., 16
 Edmonds, J., 11–12, 869
 EDVAC, 14
 Edwards, D., 143, 685, 874, 876
 Edwards, P., 840, 870
 Edwards, W., 493, 900
 EFFECT, 358, 381, 382, 385
 effect, 344
 conditional, 381, 389
 disjunctive, 383
 knowledge, 395
 universally quantified, 383, 389
 effect axiom, *see* axiom, effect
 effective branching factor, 116
 effector, 31, 777
 EFFECTS, 349, 408
 egomotion, 736
 Ehrenfeucht, A., 862
 8-puzzle, 63, 101, 103, 115
 8-queens problem, 64, 83, 86, 89
 Einstein, A., 3
 electrocutaneous feedback, 777
 electronic circuits domain, 223–226
 ELEMENTS, 321
 ELIZA, 20, 849
 Elkan, C., 213, 463, 870
 Elliot, G., 116, 874
 ELSE-PART, 395
 emergent property, 833
 empiricism, 9
 Empson, W., 687, 870
 EMPTY?, 72
 EMV, *see* expected monetary value
 encoded message model, 659, 684
 end effector, 777, 781
 Enderton, H. B., 213, 292, 870
 Engelberger, G., 810
 Engelberger, J. F., 810, 870
 English, 19, 21, 29
 subset, 662
 ENIAC, 14
 ENQUEUE-AT-END, 74
 ENQUEUE-AT-FRONT, 78
 entailment, 158, 163, 170, 277
 entailment constraint, 626, 637, 644
 Entscheidungsproblem, 11
 environment, 31, 40, 790
 accessible, 46, 122, 500
 artificial, 36
 continuous, 46, 774
 deterministic, 46
 discrete, 46
 dynamic, 46, 774
 episodic, 46
 game-playing, 147, 624
 inaccessible, 46, 500, 773
 nondeterministic, 46, 773
 nonepisodic, 46, 773
 properties, 46
 properties of, 773
 semidynamic, 46
 static, 46
 taxi, 39
 vacuum, 65
 environment class, 49
 environment history, 499
 environment simulator, 47
 EPAM (Elementary Perceiver And
 Memorizer), 559
 epiphenomenalism, 836
 epipolar line, 740
 episodic, *see* environment, episodic
 epistemological commitment, 165
 epistemological level, 153
 epoch, 576, 600
 equality (logical), 193–194
 equality substitution, 291
 equality symbol, 193
 equilibrium, 604
 equivalence (logical), 167
 Erman, L. D., 770, 870
 Ernst, G., 115, 888
 Ernst, H. A., 810, 870
 Erol, K., 389, 870
 error (of a hypothesis), 553
 error recovery (in tokenization), 704
 error surface, 580
 ERS-1, 370
 Eskimos, 162
 Essig, A., 432, 873
 estimation phase, 509
 Etchemendy, J., 181, 861
 Etzioni, O., 411, 814, 846, 848, 870,
 907
Eij, *see* utility, expected
 Euclid, 767
 Eureka, 94
 EURISKO, 647
 European Space Agency, 367
 Euthyphro, 9
 EVAL, 126, 132
 EVAL-FN, 93
 EVAL-TRUTH, 182
 evaluation function, 92, 115, 123,
 126–128
 accuracy, 127
 linear, 128
 Evans, T. G., 19, 29, 870
 event, 235
 discrete, 237
 liquid, 237
 EVENT-VAR, 677, 680
 event calculus, 235, 234–236, 259,
 675
 event category, 262

- evidence, 417
 incorporating, 428
EVIDENCE-EXCEPT, 451, 452, **452**, 452
- EVIDENCE?**, 452
 evidence variable, 445
 evidential support, 448
 evolution, 30, 162, 619
 machine, 21
 evolutionary programming, 620
 example, 529, 534
 exceptions, 229, 319
 excitatory synapse, 564
 exclusive or, 168, 596
 execution, 57
 execution monitoring, **392**, 401, 402, 410, 411, 788
 exhaustive decomposition, 231
 Existential Elimination, 266
 existential graph, 316, 323
 Existential Introduction, 266
EXPAND, 72, 73
EXPANSION, 385
 expansion (of states), 70
 expected monetary value, 476
 expected utility, *see* utility, expected
 expected value (in a game tree), 133
 expectimax, 144
 expectimax value, 133
 expectiminimax, 135
 complexity, 135
EXPECTIMINIMAX-VALUE, 135
 expectimin value, 135
 expert system, 257, 491, 539, 647, 827, 848
 commercial, 316
 first, 22
 first commercial, 24
 HPP project, 22
 logical, 458
 medical, 26, 466
 metalevel reasoning, 330
 Prolog-based, 304
 with uncertainty, 25
 explaining away, 447, 461
explanation, **326**, 630
 natural language, 721
 explanation-based learning, 627, 644, 645
 explicit representation, 615
 exploration, 623
 exploration function, **612**, 613
 exploration problem, *see* problem, exploration
 expressiveness vs. efficiency, 531
EXTENDER, 698
 extension (of a causal link), 405
 extension (of a concept), 545
 extrinsic property, 243
 eyes, 724, 727, 728, 768
- F**
- \mathcal{F} (free space), 791
 f -COST, 107
 factoring (in resolution), 280
 Fahlman, S. E., 19, 260, 330, 331, 870
 fail, 355, 855
 false negative, 545
 false positive, 546
 family tree, 637
 Farhat, N., 595, 870
 Faugeras, O., 769, 770, 870
 fault tolerance, 566
 fax, 664
 feature (of a state), 104
 feature (speech), 758
 feature detector, 586
 feed-forward network, *see* neural network, feed-forward
 feedback loop, 461
 feedforward circuit, 789
 Feigenbaum, E. A., 22, 28, 94, 257, 329, 559, 828, 861, 864, 870, 882
 Feldman, J., 28, 494, 495, 870
 Feldman, R., 646, 898
 Fermat, P., 12, 431
FETCH, 299–302, 334
 field of influence, 805
 Fifth Generation project, 24, 308, 329
 figure of speech, 714, 715
 Fikes, R., 330, 332, 363, 411, 645, 863, 870, 894
 filler, 710
FILTER, 311
FIND-AND-INFER, **273**, **273**, **273**
FIND-TRANSFORM, 753, **753**, 754, 772
 Findlay, J., 258, 871
FINISH, 406, 408
 finite-state machine, 656, 788
 Firby, J., 389, 868
FIRST, 273, 275, 303, 338, 395, 402
 first-order logic, *see* logic, first-order
 first-order probabilistic logic, 843
 Fischer, M., 259, 871
 Fisher, R., 432, 871
FITNESS-FN, 620
 fitness function, 619, 619
 fixation, 739
 Flakey, 789
 flat tire, 393, 412
 floor-planning, 91
 Flores, E., 827, 902
- Floyd, R.**, 87, 871
fluent, **241**, 679
 fly eyes, 737, 749
FMP, *see* planning, fine-motion
Focus, 646
 focus, 727
 focus of expansion, 736
 Fodor, J. A., 653, 832, 839, 871
FOIL, 642, **643**, 643, 644, 646, 648
FOL, *see* logic, first-order
 foliation, 793
 folk psychology, 13, 261, 840
FOPC, *see* predicate calculus
FORBIN, 389, 390
 Forbus, K. D., 260, 328, 332, 871
 force sensor, *see* sensor, force
 foreshortening, 743
 forgotten node, 108
 formal language, *see* language, formal
 formulate, search, execute, 57
FORMULATE-GOAL, 57
FORMULATE-PROBLEM, 57
 Forsyth, D., 769, 871, 893
 Fortmann, T. E., 520, 860, 872
 FORTRAN, 623
 forward-backward algorithm, 766
FORWARD-CHAIN, **273**, **273**, 274, 294
 forward chaining, 272, 273, 298, 313
 forward checking, 84
 Fox, M., 369, 390, 871
 Fraenkel, 826
FRAIL, 298
 frame (representation), 23
 frame (speech), 758
 frame axiom, *see* axiom, frame
 frame problem, **207**, 213
 inferential, 207
 representational, 207
 frames paper, 331
 frame system, 298, 316
 Frean, M., 595, 871
FREDDY, 70, 810
 Fredkin, E., 849
 Fredkin Prize, 144
 Freedman, P., 788, 873
 free space, 791
 freeways, 800
 free will, 9
 Frege, G., 11, 179, 212, 291, 825, 871
 frequentism, 430
 Friedberg, R., 21, 623, 829, 871
 fringe, 72
 Fristedt, B., 623, 862
 frontier, 72

- Fu, K., 687, 871
 Fuchs, J., 390, 871
 function, 185
 learning a, 529, 558
 functional decomposition, 249
 functional dependency, 633, 646
 functionalism, 50, 819, 835, 839
 functionalist, 836
 functional programming, 329, 329
 function symbol, 188, 188, 211
 Fung, R., 465, 871
 Furukawa, K., 329, 871
 Furuta, K., 618, 871
 fuzzy logic, *see* logic, fuzzy
 fuzzy set, 466
 fuzzy set theory, 463
- G**
- $G_\sigma(x)$ (Gaussian), 733
g (activation function), 568
 G-set, 550
 Gabbay, D., 180, 871
 Gabor, Z., 495
 gain ratio, 544, 562
 Gala, S., 325, 861
 Galileo, G., 3, 526, 641
 Gallaire, H., 328, 872
 Gallier, J. H., 213, 292, 872
 Gamba, A., 594, 872
 Gamberini, L., 594, 872
 gambling, 12, 474
 game, *see also* backgammon,
 bridge, checkers, chess, Go,
 Othello
 perfect information, 122
 playing, 122–141
 theory, 122, 847
 three-player, 147
 zero-sum, 148
 game show, 476
 gap, 710
 Garding, J., 743, 769, 872
 Gardner, M., 292, 872
 Garey, M. R., 853, 872
 GARI, 390
 Garside, R., 688, 572
 Gaschnig, J., 23, 116, 466, 869, 872
 GASOIL, 539
 Gasquet, A., 390, 871
 gates, 223
 gathering, 253–254
 Gauss, C., 872
 Gauss, C. E., 86
 Gauss, K. F., 520
 Gaussian elimination, 506
 gaussian function, 733
 Gawron, J. M., 770, 878
 Gazdar, G., 686, 872
 Geffner, H., 466, 872
 Geiger, D., 596, 875
 Gelatt, C., 117, 879
 Gelb, A., 520
 Gelernter, H., 18, 329, 872
 Gelfond, M., 466, 872
 gene, 619
 GENERAL-SEARCH, 71, 73, 74, 77,
 78, 92, 93, 115
 General Electric, 776
 generalization, 546, 547, 584
 generalization hierarchy, 552
 generalized cylinder, 752, 769
 general ontology, 226–247
 General Problem Solver, 6, 363
 general search, 85
 GENERATE-DESCRIPTION, 662, 663
 generation (of language), *see* natural
 language, generation
 generation (of states), 70
 generative capacity, 656, 671
 Genesereth, M. R., 153, 180, 213,
 219, 286, 293, 295, 309, 313,
 330, 363, 811, 872, 892, 897
 GENETIC-ALGORITHM, 620
 genetic algorithm, 21, 572, 595,
 611, 619–621, 623
 genetic engineering, 838
 Gentner, D., 646, 872
 Gentzen, G., 179, 291, 872
 Geometry Theorem Prover, 18
 Georgeff, M. P., 364, 872
 Gerberich, C., 329, 872
 Gestalt school, 768
 GHC, 329
 Ghose, S., 117, 864
 Gibson, J., 769, 872
 Gift of the Magi, 378
 Gilmore, P., 292, 872
 Gini, M., 411, 888
 Ginsberg, M. L., 29, 295, 411, 465,
 466, 572, 873, 897
 Ginsberg, M.L., 867
 Giralt, G., 788, 873
 gist, 162
 Givan, R., 884
 Glanc, A., 810, 873
 Glass, J., 903
 GLAUBER, 647
 Gleitman, L., 687, 900
 GLOBAL-TRAIL-POINTER, 307
 Glover, E., 117, 873
 Glymour, C., 596, 897
 Go, 122, 139
 goal, 42, 55, 56, 211
 concept, *see* goal predicate
 formulation, 56
 maintenance, 400
 predicate, 531
 representation, 339
 test, 60, 85
 goal (inferential), 201
 GOAL-TEST, 60, 73, 107, 110
 goal predicate, 531
 God, existence of, 432
 Goddeau, D., 26, 903
 Gödel, K., 11, 265, 277, 292, 824,
 873
 Goetsch, G., 117, 862
 GOFAI, *see* good old-fashioned AI
 gold, 153
 Gold, E. M., 559, 687, 873
 Goldbach's conjecture, 647
 Goldberg, D. E., 619, 873
 Golden, K., 390, 861
 Goldman, R., 688, 720, 864, 873
 Goldszmidt, M., 467, 873
 Golea, M., 595, 883
 GOLEM, 641, 646
 GOLUX, 330
 Gomard, C. K., 645, 878
 Good, I. J., 142, 465, 873
 good and evil, 471
 Goodman, N., 258, 645, 873, 882
 good old-fashioned AI, 827, 839
 good style, 217
 gorillas, 653
 Gorry, G., 432, 873
 Gould, S. J., 458, 573
 Gower, A., 144, 877
 GPS, 6, 10, 17
 GPSG, 686
 graceful degradation, 518, 566
 gradient descent, 111, 577, 580, 616
 Graham, S. L., 720, 873
 grammar, 854
 attribute, 685
 categorial, 687
 context-free, 656, 684, 685
 probabilistic, 683, 687
 context-sensitive, 656
 definite clause, 697
 definite clause (DCG), 667, 685
 dependency, 687
 formal, 662
 generalized phrase structure
 (GPSG), 686
 head-driven phrase structure
 (HPSG), 686
 lexical-functional (LFG), 687
 multistratal, 686
 phrase structure, 684
 recursively enumerable, 656
 regular, 656
 semantic, 687
 transformational, 686, 686

- tree-adjoining (TAG), 687
 grammatical formalisms, 656, 686
 Grand Canyon, 682
 Grand Prix, 141
 granularity, 380
 grasping, 725
 gravity, 826
 Grayson, C. J., 477, 873
 greedy, *see* search, greedy
 GREEDY-SEARCH, 93
 Green, C., 19, 212, 313, 330, 363, 873
 Greenbaum, S., 685, 891
 Greenblatt, R., 143, 144, 873, 887
 Gregory, S., 329, 865
 Greiner, R., 646, 873
 Grice, H. P., 685, 873
 Griesmer, J., 325, 884
 Grimes, J., 720, 873
 Grishman, R., 688, 865
 Grosof, B., 646, 893
 Grosz, B., 688, 694, 719, 720, 874
 ground clause, 289
 grounding, 821
 ground literal, 300
 ground resolution theorem, 289
 ground term, 190
 Grove, A., 432, 519, 860
 GSAT, 114, 116, 182, 183, 183, 184
 Gu, J., 116, 874
 Guard, J., 313, 330, 874
 Guha, A., 595, 874
 Guha, R., 242, 258, 828, 844, 582
 Gupta, A., 316, 859
 Gutfreund, H., 595, 860
 Guy, R., 143, 861
- H**
- H_{MAP} (MAP hypothesis), 588
 H_{ML} (ML hypothesis), 589
 Haas, A., 260, 874
 HACKER, 330, 363
 Hacking, I., 433, 874
 Hager, G., 258, 876
 HAL computer, 465
 Hald, A., 433, 874
 Halpern, J., 260, 432, 519, 860, 874
 halting problem, 277, 824, 824
 hamburger, 821
 Hamming, R. W., 433, 874
 Hammond, K., 389, 874
 ham sandwich, 715
 hand-eye machine, 810
 Hanks, S., 411, 843, 869, 870, 874
 Hanna, F. K., 647, 892
 Hansen, J., 329, 872
 Hanski, I., 50, 874
 Hansson, O., 116, 874
- happy graph, 538
 Haralick, R., 116, 874
 Harel, D., 261, 874
 Harkness, K., 141, 874
 Harman, G. H., 839, 874
 Harp, S. A., 595, 874
 HARPY, 770
 Harris, L. R., 720, 874
 Harrison, M. A., 720, 873
 Hart, P., 23, 115, 116, 411, 466, 645, 869, 870, 874
 Hart, T., 143, 874
 Hartman, E., 595
 Harvard, 14, 479
 hash table, 300, 704
 Haugeland, J., 5, 28, 827, 840, 874
 Haussler, D., 560, 646, 862, 874
 Hawkins, J., 594, 874
 Hayes, J. E., 144, 875
 Hayes, P. J., 50, 213, 258–260, 330, 331, 820, 875, 884
 Hayes-Roth, E., 770, 870
 Hazan, M., 250, 875
 HD-POP, 374, 374, 389
 head (of a clause), 305
 hearer, 652
 HEARSAY-II, 770
 Heath Robinson, 14
 Hebb, D. O., 16, 19, 594, 622, 875
 Heckerman, D., 25, 27, 457, 461, 462, 465, 466, 495, 596, 875, 876
 Hegde, S. U., 595, 886
 Held, M., 116, 875
 Helman, D. H., 720, 875
 Helmholz, H., 12
 Hendler, J., 364, 389, 859, 870
 Hendrix, G., 323, 875
 Henglein, E., 330, 889
 Henrion, M., 50, 444, 456, 465, 495, 875, 876, 890
 Henry, O., 378
 Hephaistos, 810
 Heppenheimer, T., 810, 875
 Herbrand's theorem, 287
 Herbrand, J., 287, 292, 328, 875
 Herbrand base, 287
 Herbrand universe, 286, 292, 293
 hermeneutic, 94
 Herskovits, E., 596, 866
 Hertz, J., 595, 875
 heuristic, 94, 115, 118
- admissible, 97
 - composite, 104
 - function, 93, 101
 - least-constraining-value, 105, 116
 - Manhattan, 102, 106, 118
 - min-conflicts, 114
- monotonic, 97
 most-constrained-variable, 105, 116
 most-constraining-variable, 105
 most-constraining conjunct, 309
 search, *see* search, heuristic
 straight-line, 93
 Heuristic Programming Project, 22, 94
 heuristic repair, 114
 Hewitt, C., 330, 875
 hidden Markov model, 25, 712, 762, 770
 hidden unit, 571
 hierarchical abstraction, 654
 hierarchical decomposition, *see* planning, hierarchical
 hierarchical task network, *see* planning, hierarchical
 higher-order logic, *see* logic, higher-order
 Hilare II, 788
 Hilbert, D., 11
 HILL-CLIMBING, 112
 hill-climbing, 111, 119, 572, 590
- random-restart, 112
- Hintikka, J., 260, 875
 Hinton, G., 24, 595, 875, 893
 Hirsh, H., 645, 875
 Hirst, G., 688, 875
 Hitachi, 369
 HITECH, 26, 137, 144
 HMM, *see* hidden Markov model
 Ho, Y., 21, 578, 595, 863
 Hobbs, J. R., 258, 261, 685, 687, 718, 720, 876
 Hoff, M., 19, 594, 601, 622, 901
 Hogg, D., 29, 895
 holistic context, 828
 Holland, J. H., 623, 876
 Holloway, J., 144, 887
 holonomic, *see* robot, holonomic
 Holy Grail, 159
 homeostatic, 594
 hominids, 653
 homophone, 757, 772
 homo sapiens, 3
 homunculus, 819
 Hopcroft, J. E., 853, 859
 Hopfield, J., 24, 595, 876
 Hopfield network, 571
 Hopkins Beast, 810
 Horch, K. W., 832, 864
 horizon, 726
 horizon problem, 129
 Horn, A., 174, 292, 876
 Horn, B., 769, 770, 863, 876
 Horn clause, 174, 310, 640, 642, 667

- guarded, 329
 Horn Normal Form, 270, 290
 Horn sentence, 174, 270
 Horowitz, M., 313, 888
 Horvitz, E., 25, 50, 465, 495, 876
 house building, 372, 384
 Howard, R., 484, 493, 494, 520,
 876, 886
 HPP, *see* Heuristic Programming
 Project
 HPSG, 686
 HST, *see* Hubble space telescope
 Hsu, F.-H., 144, 876
 Hsu, K., 595, 877
 Hu, V., 595, 859
 Huang, T., 27, 520, 877, 880
 Hubble space telescope, 114, 370,
 390
 Hubel, D. H., 770, 877
 Huber, R. J., 832, 864
 Hucka, M., 788, 881
 Huddleston, R. D., 685, 877
 Huffman, D., 19, 746, 769, 877
 Hughes, G., 261, 877
 HUGIN, 465, 520
 human judgment, 446, 458, 466, 479
 human performance, 4
 human preference, 507
 Hume, D., 9, 430, 877
 Hunt, E., 559, 877
 Hunter, G., 293, 877
 Hunter, L., 877
 Hurst, S., 539, 894
 Hutchinson, C., 29, 895
 Huttenlocher, D., 753, 769, 877
 Huygens, C., 431, 877
 Hwang, C. H., 258, 686, 877
 Hyatt, R., 144, 877
 hypothesis, 529
 approximately correct, 553
 consistent, 529
 null, 542
 space, 544, 545
- I**
- IBM, 14, 138
 IBM 701 computer, 14
 IBM 704 computer, 138
 ice cream, 418
 Ichiyoshi, N., 329, 888
 ICON, 856
 ID3, 559
 IDA*, 117
 IDA*, 107
 IDEAL-PLANNER, 337, 338
 ideal mapping, 34
 ideal rational agent, 33
 identification in the limit, 557
- identity relation, 194
 IDS, 103
 ignorance, 459, 462
 practical, 417
 theoretical, 416
 IJCAI (International Joint
 Conference on AI), 28
 ILA, *see* action, intermediate-level
 ILP, *see* logic programming,
 inductive
 image, 725
 image formation, 725–731, 767
 image processing, 767
 image reconstruction
 Bayesian, 370
 implementation level, 153
 implication, 167
 Implication-Elimination, 172
 implicative normal form, 278, 282,
 290, 292
 implicit representation, 615
ini (sum of inputs), 568
 inaccessible, *see* environment,
 inaccessible
 incomplete information, 392
 incompleteness, 277
 theorem, 11, 288, 824
 incorporation, 658, 716
 incremental learning, 529, 549
 independence
 conditional, 444
 indeterminacy
 bounded, 401
 unbounded, 402
 index function, 755
 indexical, **679**
 indexing
 clever, 302
 combined, 301
 table-based, 300–301
 tree-based, 301, 301–302
 indifference, 432
 individuation, 242
 INDUCE, 529, 530
 induction, 9, 529
 constructive, 638
 mathematical, 11
 inductive inference
 pure, 529
 inductive learning, *see* learning
 inductive logic programming (ILP),
 646
 INF, *see* implicative normal form
 INFER, 311, **311**
 inference, 152, 163–165, 265
 causal, 447
 data-driven, 274
 diagnostic, 447
- intercausal, **447**
 logical, 163
 mixed, 447
 probabilistic, 436
 inference procedure, 223
 inference rule, 171, 171
 inferential frame problem, 213
 infinite horizon problems, 520
 infinite loop, 203
 infinite regress, 819
 influence diagram, *see* decision
 network, 877
 information (theory), 540
 INFORMATION-GATHERING-AGENT,
 490
 information gain, 541, 542
 information highway, 848
 information retrieval, 694–695
 information theory, 540–543, 559
 information value theory, 487
 informed search, 73, 92
 infrared, 512
 Ingerman, P., 685, 877
 inheritance, 230, **230**, 317, 328, 332
 multiple, 320
 with exceptions, 319–320
 inhibitory synapse, 564
 INITIAL-STATE, 60, 73, 107, 110,
 112, 113
 INITIALIZER, 699, 700
 initializer, 698
 initial state, 60, 85, 123
 input function (of a neuron), 567
 input generalization, 615
 input resolution, 285
 input unit, 571
 insurance premium, 478
 INTEGER, 856, **856**
 integrability, 745
 INTELLECT, 720
 intelligence test, 90
 intention (discourse), 657
 intentionality, 831, 839, 840
 intentional stance, 50, 820, 840
 intentional state, 820
 intercausal inference, *see* inference
 intercausal reasoning, *see* reasoning
 interleaving, 59
 intermediate form, 676
 internal model, 203
 internal state, 42
 INTERPLAN, 363
 INTERPRET-INPUT, 41
 interpretation, 161, 165, 187
 logical, 178
 pragmatic, 658, 678–680, 684,
 685
 semantic, 658

- interreflections, 745
 interval, 235
 intractability, 11, 21, 29
 intrinsic property, 243
 introspection, 6, 13, 459
 invariant
 geometric, 754
 projective, 754
 invariant shape representation, 755
 inverse method, 293
 inverse resolution, *see* resolution,
 inverse
 inverted pendulum, 617
 IPEM, 411
 IPL, 17
 IQ test, 19, 29
 IR, *see* information retrieval
 irrational behavior, 473, 824
 irrationality, 4
 ISA links, 331
 Isis, 369, 390
 ITEP, 144
 ITERATIVE-DEEPENING-SEARCH, 79,
 103, 106
 iterative deepening, *see* search,
 iterative deepening
 iterative improvement, 111, 115
 ITOU, 646
- J**
 Jackel, L., 586, 595, 881
 Jackson, P., 29, 877
 Jacobs, P., 696, 877
 Jacquard, J., 15
 Jacquard loom, 15, 774
 Jaffar, J., 329, 877
 Jaguar, 369
 James, H., 13
 James, W., 13
 Japanese, 703
 Jaskowski, S., 291, 877
 Jefferson, Prof., 830
 Jeffrey, R. C., 293, 432, 493, 494,
 863, 877
 Jelinek, E., 687, 759, 770, 862, 877
 Jensen, E., 465, 860
 Jensen, E V., 465, 860, 877
 Jerison, H. J., 653, 877
 Jessell, T. M., 595, 878
 Jevons, W., 292
 Jochem, T., 587, 877
 Johnes, K. E., 832, 864
 Johnson, C., 559, 864
 Johnson, D. S., 853, 872
 Johnson, M., 720, 722, 863, 881
 Johnson, W., 86, 878
 Johnson-Laird, P., 28, 878
 Johnston, M., 116, 390, 878, 886
- joint, *see* joint probability
 distribution
 joint (of a robot), 777, 809
 joint encoder, 782
 joint probability distribution, 425,
 431, 436, 439
 jokes, 682
 Jones, N. D., 645, 878
 Joshi, A., 687, 720, 878, 894
 Joskowicz, L., 260, 894
 Joule, J., 641
 JTMS, *see* truth maintenance
 system, justification-based
 Juang, B.-H., 770, 891
 Judd, J., 595, 878
 judgment, *see* human judgment
 juggling, 599
 Julesz, B., 768, 878
 junction type, 748
- K**
 k-DL (decision list), 555
 k-DT (decision tree), 555
 Kaelbling, L. P., 180, 520, 623, 788,
 864, 868, 878
 Kahan, W., 466
 Kahneman, D., 443, 479, 878, 899
 Kaindl, H., 117, 143, 878
 Kalman, R., 510, 520, 878
 Kalman filtering, 510, 520, 587
 Kambhampati, S., 390, 878
 Kanade, T., 737, 899
 Kanal, L. N., 116–117, 467, 878,
 880, 881, 887
 Kanazawa, K., 520, 868
 Kandel, E. R., 595, 878
 Kanoui, H., 328, 866
 Kaplan, D., 259, 878
 Kaplan, R., 720, 884
 Karger, D., 87, 878
 Karp, R. M., 12, 69, 116, 853, 875,
 878
 Kasami, T., 720, 878
 Kasparov, G., 137, 144
 Kassirer, J., 432, 873
 Kastner, J., 325, 884
 Kautz, H. A., 258, 466, 876, 878
 Kay, M., 692, 770, 878
 KB, *see* knowledge base
 KB-AGENT, 152, 202
 KBIL, *see* inductive learning,
 knowledge-based
 Kearns, M. J., 560, 878
 Kedar-Cabelli, S., 645, 886
 Kedzier, D., 539, 894
 Keeler, J., 595, 890
 Keeney, R. L., 479, 484, 494, 878,
 879
- Keller, R., 645, 886
 Kelly, J., 865
 Kemp, M., 768, 879
 Kenley, C., 465, 895
 Kent, C., 244
 Kepler, J., 768
 Ketchpel, S., 219, 872
 Keynes, J. M., 432, 879
 Khorsand, A., 117, 878
 KIDS, 330
 Kierulf, A., 145, 879
 Kietz, J.-U., 646, 879
 Kilimanjaro, 651
 Kim, J. H., 465, 879
 kind, *see* category
 kinematics, 781
 King, C., 447
 King, R., 641, 879, 887
 King, S., 27, 879
 kinship, 197
 Kirkpatrick, S., 117, 879
 Kirman, J., 520, 868
 Kirousis, L., 749, 879
 Kister, J., 143, 879
 Kjaerulff, U., 520, 879
 KL-ONE, 298, 332
 KL1, 329
 Klein, E., 686, 872
 Knight, B., 142, 894
 Knight, K., 5, 328, 879, 892
 Knoblock, C., 86, 879
 knowing what, 246
 knowing whether, 246
 knowledge
 acquisition, 23, 217
 and action, 10, 247
 background, 152
 built-in, 35
 commonsense, 18
 diagnostic, 427
 model-based, 427
 knowledge-based approach, 827
 knowledge-based system, 22–24,
 615
 knowledge base, 151, 178, 213
 large, 844
 properties, 218
 knowledge effect, 247
 knowledge engineer, 440
 knowledge engineering, 217, 221,
 368
 for decision-theoretic systems,
 492
 for planning, 359
 vs. programming, 219
 with uncertainty, 456
 knowledge level, 153, 179
 knowledge map, *see* belief network

- knowledge precondition, 247
 knowledge representation, 5, 15, 18, 23, 157, 257
 analogy, 213
 language, 152, 178
 knowledge source (KS), 770
 knows that, 246
 Knuth, D., 132, 143, 293, 685, 853, 879
 Knuth–Bendix algorithm, 293
 KODIAK, 298
 Koenderink, J., 769, 879
 Kohn, W., 329, 879
 Kohonen, T., 595, 880
 Koko, 653
 Koller, D., 27, 87, 432, 519, 520, 596, 860, 877, 878, 880, 893
 Kolmogorov, A. N., 431, 432, 520, 59, 880
 Kolmogorov complexity, 559
 Kolodner, J., 23, 646, 880
 Konolige, K., 260, 880
 Koopmans, T., 520, 880
 Korf, R. E., 87, 117, 880
 Kotok, A., 143, 880
 Kowalski, R., 213, 259, 292, 304, 328, 330, 880
 Koza, J. R., 623, 880
 Kripke, S. A., 260, 880
 Krogh, A., 595, 875
 Krotkov, E., 778, 896
 Kruppa, E., 768, 880
 KRYPTON, 332
 Kube, P., 258, 876
 Kuehner, D., 293, 880
 Kukich, K., 720, 880
 Kulikowski, C. A., 560, 901
 Kumar, V., 116, 117, 878, 880, 881, 887
 Kurzweil, R., 5, 28, 881
 Kyburg, H. E., 432, 881
- L**
- Ladkin, P., 259, 881
 Ladner, R., 259, 871
 Lafferty, J., 687, 862
 Laird, J., 26, 316, 645, 788, 843, 881, 887
 Laird, P., 116, 886
 Lakoff, G., 258, 720, 881
 lambda calculus, 329
 Lambertian surface, 729, 743, 771
 La Mettrie, J. O. de, 838, 849, 881
 landmark, 805, 809
 Langley, P., 881
 Langlotz, C., 25, 876
 language, 651
 as action, 685
 analysis, 658
 formal, 654
 interfaces, 23, 693
 model, 760
 in disambiguation, 682
 module, 653
 natural, 7, 161, 654
 origin, 653
 perception, 657
 problem-solving, 329
 processing, 16
 situated, 659, 659, 684
 and thought, 162, 653
 translation, 21, 632, 691–693, 720
 understanding, 19, 23
 Lansky, A. L., 364, 872
 Laplace, P., 12, 430, 432, 458
 Larson, G., 626
 laser range finder, 785
 Lassez, J.-L., 328, 329, 877, 881
 Latin, 668
 Latombe, J.-C., 390, 811, 868, 881
 lattice theory, 313
 Lauritzen, S., 465, 596, 877, 881, 897
 LAWALY, 389
 Lawler, E., 116, 881
 Laws, K. I., 685, 876
 laziness, 416, 558
 leak node, 443
 leaping to conclusions, 626
 learning, 153, 525, 823, 830
 action-value, 599
 active, 599
 architecture, 844
 assessing performance, 538
 Bayesian, 588, 588–589, 593
 belief network, 531, 588–592
 blocks-world, 19
 cart-pole problem, 617
 checkers, 18
 curve, 538
 decision lists, 555–556
 decision trees, 531–536, 641
 determinations, 634
 driving, 586–587
 element, 525, 558
 explanation-based, 788
 and failure, 403
 to fly, 539
 game-playing, 617
 handwriting recognition, 586
 human, 624
 incremental, 529, 549, 626
 inductive, 558, 566, 829
 knowledge-based, 628, 637, 645
 knowledge in, 625–628
 linearly separable functions, 575–577
 new predicates, 638, 640
 PAC, 553, 560, 632
 passive, 599
 pronunciation, 585–586
 Q, 613
 rate, 576, 579, 604, 613
 restaurant problem, 531, 620
 speedup, 527
 top-down, 641–644
 utility function, 599
 least-commitment, 549
 least-constraining-value, *see* heuristic
 least commitment, 271, 346, 374
 least mean squares, 601
 Le Cun, Y., 586, 595, 881
 Lederberg, J., 22, 94, 257, 870, 882
 Lee, K.-E., 623, 770, 881, 900
 Lee, R., 294, 864
 Leech, E., 688, 872
 Leech, G., 685, 891
 Lefkowitz, D., 145, 882
 left-corner parser, 699
 legal reasoning, 29
 Leibniz, W., 9, 15, 179, 432, 827
 Leiserson, C. E., 853, 866
 Lemmer, J. E., 467, 878
 Lenat, D. B., 242, 258, 263, 646, 647, 828, 844, 867, 882
 LENGTH, 666, 702, 851
 lens, 727
 León, M., 692, 900
 Leonard, H. S., 258, 882
 Leonard, J., 520, 882
 LePape, J.-P., 330, 869
 Lesh, N., 411, 870
 Leśniewski, S., 258, 882
 Lesser, V. R., 770, 570
 levels of description, 152
 Levesque, H., 184, 259, 261, 331, 332, 863, 882, 894, 895
 Levy, D. N., 144, 875
 Lewis, D. K., 50, 685, 839, 882
 Lewis, R., 641, 879
 LEX, 552, 559
 lexicon, 664, 686, 698, 703–704
 LFG, *see* grammar,
 lexical-functional
 Li, M., 560, 882
 liability, 848
 Lieberman, L., 769, 860
 LIFE, 297, 329
 life insurance, 479
 Lifschitz, V., 363, 390, 466, 872, 882
 lifting lemma, 286, 289

- light-beam sensors, 785
 Lighthill, J., 21, 882
 Lighthill report, 21, 24
 likelihood, relative, 427
 likelihood weighting, 456, 465
 limb (in a scene), 745
 limited rationality, 8
 Lin, S., 115
 Linden, T. A., 411, 882
 Lindsay, R. K., 257, 720, 882
 linear input resolution, 312
 linearization, 346, 347
 linear plan, *see* plan, linear
 linear resolution, 285
 linear separability, 574, 593
 line labelling, 745
 linguistics, 15–16, 28
 computational, 16
 Linguistic String Project, 685
 link (in a neural network), 567
 link (of a robot), 777, 809
 linked list, 329
 linking curve, 800
 LINKS, 347, 372, 375
 Linnaeus, 258
 Lipkis, T., 332, 894
 LIPS, *see* logical inferences per second
 liquid event, *see* event, liquid
 liquids, 260
 Lisp, 18, 197, 329, 654
 Common, viii, 329, 330, 855
 lists, 200
 literal (sentence), 167, 182
 Littman, M., 520, 864
 lizard toasting, 627
 LLA, *see* action, low-level
 Lloyd, J., 328, 882
 LMS, *see* least mean squares
 LMS-UPDATE, 602, 602, 623
 LMT (Lozano-Perez, Mason, and Taylor), 795
 local encoding, 577
 locality, 174, 460
 locally finite graph, 100
 locally structured system, 441
 Locke, J., 9
 Locke, W. N., 720, 882
 locking, 312
 locomotion, 777, 778
 Lodge, D., 848
 logic, 7, 11, 151, 158
 autoepistemic, 466
 combinatory, 329
 default, 459
 description, 298
 dynamic, 259
 first-order, 185, 211
 semantics, 186
 syntax, 186
 first-order probabilistic, 843
 fuzzy, 166, 417, 459, 463–466
 higher-order, 195
 inductive, 432, 432
 modal, 245, 258, 260
 nonmonotonic, 321, 459
 notation, 7
 propositional, 151, 165
 limitations, 176
 semantics, 168
 syntax, 166
 temporal, 165, 258, 259
 terminological, 298
 logical connective, 16, 165, 189
 logical inferences per second, 307
 logical level, 153
 logical omniscience, 246
 logical piano, 292
 logical positivism, 10
 logicism, 7, 16
 logic programming, 293, 304–310,
 328, 343
 constraint, 308, 328
 inductive (ILP), 628, 636–645
 language, 297
 logic sampling, 455, 465, 515
 Logic Theorist, 292
 Logic Theorist (LT), 17
 logic variables, 306, 308
 LOGIN, 328, 329
 long-distance dependency, 710
 long-term memory, 316
 Longuet-Higgins, H., 769, 882
 LOOKUP, 38, 321
 lookup table, 38, 572
 LOOM, 298
 lottery, 473
 standard, 478
 love, 823
 Lovejoy, W., 520, 883
 Lovelace, A., 15, 823
 Loveland, D., 293, 294, 883
 Lowe, D., 769, 883
 Löwenheim, L., 212, 883
 Lowerre, B., 770, 883
 Lowrance, J., 466, 893
 Lowry, M. R., 330, 883
 Loyd, S., 86, 883
 Lozano-Pérez, T., 795, 883
 LT, 17
 LT (Logic Theorist), 17
 Luby, M., 465, 867
 Lucas, J., 824, 883
 Luger, G. R., 5, 883
 LUNAR, 23, 693, 720
 Lunokhod robots, 775
 Lusk, E., 294, 902
 M
 M^a (model), 499
 M (model), 605
 MacHack 6, 143
 machine evolution, 21
 machine learning, 5, 7, 463, 687,
 829
 machine translation, *see* language,
 translation
 Machover, M., 213, 861
 Mackworth, A., 749, 769, 883
 Macpherson, P., 466, 899
 macro-operator, 787
 macrop, 389
 Maes, P., 36, 883
 Magerman, D., 687, 862, 883
 Mahajan, S., 623, 881
 Mahanti, A., 117, 883
 Mahaviracarya, 431
 Maher, M., 328, 881
 MAJORITY-VALUE, 537
 majority function, 533, 573
 MAKE-ACTION-QUERY, 152, 177,
 202, 663
 MAKE-ACTION-SENTENCE, 152,
 202, 338, 395, 408
 MAKE-GOAL-QUERY, 337, 338, 408
 MAKE-MINIMAL-PLAN, 356
 MAKE-NODE, 73, 107, 110, 112,
 113, 666
 MAKE-PERCEP-SENTENCE, 152,
 177, 202, 338, 395, 402,
 408, 663
 MAKE-PLAN, 399, 408
 MAKE-QUEUE, 72, 73, 110
 Malik, J., 27, 520, 737, 743, 749,
 769, 877, 880, 883
 Manhattan, *see* heuristic, Manhattan
 MANIAC, 587
 Manin, Y., 293, 883
 manipulation, 254, 725, 777
 manipulator, 780
 Mann, W. C., 719, 720, 883
 Manna, Z., 213, 293, 330, 883
 manufacturing, 390
 MAP, *see* maximum a posteriori
 map-coloring problem, 91, 105
 mapping, 34
 ideal, 34
 Marchand, M., 595, 883
 Marin, J., 559, 877
 Mark I/II/III, 14
 Markov, A. A., 500, 770, 883
 Markov blanket, 465
 Markov chain, 514
 Markov decision problem, 411, 500

- partially observable, 500
 Markov decision problems, 520
 Markov model, 762
 Markov property, 500, 509, 762, 765
 Marr, D., 769, 883
 Marriott, K., 328, 881
 Marsland, A. T., 144, 884
 Martelli, A., 116, 143, 328, 884
 Martin, C., 28, 884
 Martin, J., 720, 884
 Martin, P., 258, 694, 720, 874, 876
 MARVEL, 26
 Maslov, S., 293, 884
 Mason, M., 411, 795, 883, 884
 mass noun, 242
 mass spectrometer, 22
 MATCH, 666
 match phase, 314
 material advantage, 127
 material handling, 774
 materialism, 9, 819
 eliminative, 840
 material value, 127
 Mates, B., 179, 884
 mathematical induction schema, 288
 mathematical objection, 824, 826
 mathematics, 11–12
 Matheson, J., 484, 494, 876, 886
 Mauchly, J., 14
 MAUT, *see* multiattribute utility theory
 MAX, 110, 132
 MAX-VALUE, 130–132, 132
 maximum a posteriori, 588
 maximum expected utility, 472, 493, 502
 maximum likelihood, 589
 Maxwell, J., 458, 720, 884
 Mayer, A., 116, 874
 Mays, E., 325, 884
 McAllester, D., 143, 312, 330, 332, 364, 884
 MCC, 24
 McCarthy, J., 17, 18, 50, 179, 212, 213, 230, 257, 259, 329, 459, 820, 832, 884
 McCartney, R. D., 330, 883
 McCawley, J. D., 685, 885
 McClelland, J. L., 24, 893
 McConnell-Ginet, S., 28, 685, 865
 McCorduck, P., 810, 885
 McCulloch, W. S., 16, 19, 563, 570, 594, 885
 McCune, W., 310, 330, 885
 McDermott, D., 5, 213, 317, 328, 330, 331, 390, 411, 459, 865, 555, 896, 595
 McDermott, J., 24, 316, 885
- McDonald, D., 707
 McDonald, R., 185
 McGuinness, D., 324, 332, 863
 MDL, *see* minimum description length
 MDP, *see* Markov decision problem
 Mead, C., 595, 885
 meal, 251
 meaning, 161
 means–ends analysis, 10, 10
 mean square error, 623
 measure, 231, 386
 measure fluent, 386
 medical diagnosis, 22, 23, 27, 465
 Meehan, J., 328, 865
 Meet, 239
 meganode, 453
 Megarian school, 179, 258, 291
 Megiddo, N., 847, 885
 Melcuk, I., 687, 885
 Mellish, C., 330, 865
 MEMBER, 307, 307, 322
 MEMBER?, 322
 MEMBERSHIPS, 321, 322
 memoization, 87, 452, 629
 memory requirements, 75, 77, 79
 meningitis, 426–427, 434, 435
 mental model, in disambiguation, 682
 mental objects, 243–247
 mental states, 819
 Mercer, R., 687, S62
 mereology, 258
 Merkhofer, M., 494, 886
 MERLIN, 331
 Mero, L., 116, 885
 meta-comment, 718
 Meta-DENDRAL, 552, 559, 641
 METALOG, 330
 metamer, 730
 metaphor, 715, 720, 822
 metaphysics, 10
 metareasoning, 140, 309, 364
 decision-theoretic, 844
 metonymy, 714, 720
 Metropolis, N., 117, 885
 Metropolis algorithm, 117
 MEU, *see* maximum expected utility
 Mézard, M., 573, 595, 555
 MGSS*, 143
 MGU, *see* unifier, most general
 Michalski, R. S., 560, 555
 Michaylov, S., 329, 877
 Michie, D., 28, 70, 86, 115, 117, 144, 221, 539, 560, 618, 622, 810, 869, 885, 894
 MICRO-PLANNER, 330
 microelectronics industry, 774
- micromort, 480, 494, 497
 microworld, 19, 19, 21, 827
 Middleton, B., 444, 456, 890
 Miles, E., 596, 885
 Mill, J. S., 10, 546, 885, 886
 Miller, A., 494, 886
 Miller, D., 389, 886
 Miller, G., 595, 704, 886
 Milne, A. A., 218, 886
 MIN, 107, 132
 min-conflicts, *see* heuristic, min-conflicts, 116
 MIN-VALUE, 130, 132, 132
 mind, 5, 817, 838
 conscious, 818
 dualistic view, 838
 and mysticism, 565
 philosophy of, 817, 838, 840
 as physical system, 8, 9
 and symbolism, 180
 theory of, 6
- MINIMAL-CONSISTENT-DET, 635, 635
- minimax, *see* search, minimax
 MINIMAX-DECISION, 126, 147
- MINIMAX-VALUE, 126, 126, 135, 147
- minimax decision, 124
 minimization (logical), 236
 minimum description length, 560
 minimum spanning tree, 116, 119
 Minker, J., 328, 872, 886
 Minsky, M. L., 16, 18, 21, 23, 24, 28, 331, 465, 577, 578, 594, 595, 827, 886
- Minton, S., 116, 645, 886
- missing attribute values, 543
 missionaries and cannibals, 67, 86, 88
- MIT, 17, 18, 781, 810
- Mitchell, D., 184, 895
- Mitchell, T., 552, 559, 560, 645, 788, 812, 843, 863, 864, 885, 556
- Mitchell, T. M., 645
- MIT1, 777
- ML, *see* maximum likelihood
- ML (programming language), 329
- mobile robots, 520
- mobot, 775
- modal logic, *see* logic, modal model
- causal, 443
 (in logic), 170
 (in representation), 13
 theory, 212
 trees, 616
- model-based reasoning, 209

- model-based recognition, 785
 modification operator, 346
 modularity, 210
Modus Ponens, 172, 245, 285, 290, 291, 294, 819
 Generalized, 269, 269–270
MOLGEN, 390
 monkey and bananas, 366
 monotonicity (of a heuristic), 97, 116
 monotonicity (of a logic), 173, 321, 459
 monotonicity (of preferences), 474
Montague, R., 258, 259, 685, 878, 886, 899
Montanari, U., 116, 143, 328, 884
Monty Python, 159
Mooney, R., 645, 868
Moore, A., 623, 886
Moore, J., 331, 886
Moore, J. S., 293, 313, 328, 330, 826, 863
Moore, R., 132, 143, 260, 261, 466, 876, 879, 887
 morality, 817
Moravec, H., 835, 849, 887
More, T., 17
Morgan, J., 720, 866
Morgenstern, L., 260, 887
Morgenstern, O., 12, 142, 493, 847, 900
 morphology
 analysis, 695, 703
 derivational, 703
 inflectional, 703
Morris, P., 467, 873
Morrison, E., 142, 887
Morrison, P., 142, 887
Mosher, R., 776
 most-constrained-variable, *see* heuristic
 most-constraining-variable, *see* heuristic
 most general unifier, *see* unifier, most general
Mostow, J., 116, 118, 887
Motet, S., 27, 879
 motion, 735–737
 compliant, 803
 guarded, 803
 motion parallax, 737, 768
 motion planning, 796–808, 811
 complexity, 811
 motor subsystem, 410
Motzkin, T., 594, 859, 887
Mourelatos, A. P., 259, 887
Moussouris, J., 144, 887
MPI, *see* mutual preferential independence
MRS, 297, 309, 313, 330, 457
MST (minimum spanning tree), 119
Muggleton, S., 466, 641, 646, 860, 879, 887
 multiagent domain, 244
 multiattribute utility theory, 480, 480, 494
 multilayer network, *see* neural network, multilayer
 multiple-state problem, *see* problem multiple inheritance, 320
 multiply connected network, 453
Mundy, J., 769, 887, 893
MUNIN, 465
Murphy's Law, 59, 87
 music, 15
 mutation, 21, 619, 620
 mutually utility-independent, 484
 mutual preferential independence, 483
MYCIN, 23–24, 461, 466
 myopic policy, 490
Myrhaug, B., 331, 862, 867
 mysticism, 565
- N
- n*-armed bandit, 611
Nadal, J.-P., 573, 595, 885
Nagel, T., 839, 887
Nalwa, V. S., 13, 770, 887
NAME, 321
 naming policy, 660
 narrow content, 821
NASA, 390, 693, 776
NASL, 411
Nasr, R., 328, 859
 natural deduction, 291
 natural kind, 232, 319
 natural language, *see* language, natural
 Natural Language Inc., 694
 natural language processing (NLP), *see* language
 natural science, 654
 natural stupidity, 317
Nau, D., 870
Nau, D. S., 116, 143, 389, 390, 878, 887
Naur, P., 685, 887
Navathe, S., 325, 861
 navigation, 252–253, 725, 796–808
 landmark-based, 796
NavLab, 586
NAVTO, 787
Neal, R., 596, 887
Neapolitan, R. E., 467, 887
 neat, 25
 needle in a haystack, 160
 negated literals, 312
 negation, 167
 negation as failure, 304, 343
 negative example, 534
 negligence, 848
Neisser, U., 594, 895
Nelson, H., 144, 877
NERF, 572
NETL, 298, 331
NETtalk, 585
Netto, E., 86, 887
NEURAL-NETWORK-LEARNING, 576, 577, 580
NEURAL-NETWORK-OUTPUT, 577, 597
 neural computation, *see* neural network
 neural network, 16, 19, 24, 128, 139, 530, 563, 563, 593, 829
 vs. belief network, 592
 efficiency, 583
 expressiveness, 16, 583
 feed-forward, 570, 593
 hardware, 16
 learning, 16
 multilayer, 21, 571, 593
 and noise, 584
 nonlinearity in, 567
 recurrent, 570
 second best, 585
 neurobiology, 565, 769
Neurogammon, 617
 neuron, 16, 452, 563, 564, 819, 833, 835
NEW-CLAUSE, 643
NEW-LITERALS, 642–644
NEW-VARIABLE, 307
Newborn, M., 117, 889
Newell, A., 6, 10, 17, 26, 86, 94, 115, 143, 179, 292, 316, 329, 331, 645, 843, 887, 886–888, 896, 898
Newton, I., 3, 509
NEXT-SUCCESSOR, 110
Neyman, A., 847, 888
Nicholson, A., 520, 868, 888
Nicholson, A. E., 520
Nievergelt, J., 145, 879
Nilsson, N. J., 28, 87, 115, 116, 118, 143, 144, 213, 286, 293, 363, 411, 594, 645, 810, 811, 870, 872, 874, 888, 900
Nim, 142
Nitta, K., 329, 888
Nixon, R., 320, 682, 714
Nixon diamond, 320

- NL-MENU, 720
 NLP, *see* natural language processing
 NOAH, 330, 363, 411
 Nobel prize, 50
 node, search, 71
 noise, 535, 542–543, 552, 563, 584, 588, 636, 731, 795
 noisy-OR, 443
 nominal compounds, 706, 721
 nominalist, 258
 nominative case, *see* subjective
 noncomputability, 11
 nondeterminism, 855–856
 nondeterministic, *see* environment, nondeterministic
NONDETERMINISTIC-CHART-PARSE, 699
 nondeterministic algorithm, 855
 nonepisodic, *see* environment, nonepisodic
 nonholonomic, *see* robot, nonholonomic
 NONLIN, 363
 NONLIN+, 389, 390
 nonlinear plan, *see* plan, nonlinear
 nonmonotonicity, 321, 459
 nonmonotonic logic, *see* logic, nonmonotonic
 Nono, 267
 nonterminal symbol, 655, 656, 854
 normalization, 428
 normalizing constant, 451
 Normann, R. A., 832, 864
 normative theory, 479
 North, O., 267
 North, T., 21, 871
 Norvig, P., 328, 330, 688, 770, 878, 888
 notation
 arithmetic, 7
 logical, 7
 notational variations for FOL, 196
 noun phrase, 655, 709
 Nowatzyk, A., 144, 876
 Nowick, S., 313, 888
 NP, *see* noun phrase
 NP-completeness, 12, 21, 852, 853
 nuclear power, 468, 838
 NUMBER-OF-SUBKINDS, 322
 number theory, 647
 Nunberg, G., 720, 888
 Nussbaum, M. C., 888
 Nygaard, K., 331, 862, 867
- O*
 $O()$ notation, 852
- \mathcal{O} (configuration space obstacle), 791
 O'Brien, S., 720, 891
 O'Keefe, R., 330, 888
 O-PLAN, 369–371, 387, 390
 object, 165, 185, 188
 composite, 234
 object-oriented programming, 15
 object creation, 384
 objective case, 668
 objectivism, 430
 object recognition, 725, 751–755
 observation sentences, 10
 obstacle avoidance, 808
 OCCUR-CHECK, 303
 occur-check, 303, 305
 Ochiai, T., 618, 871
 Ockham's razor, 534, 558, 559, 589, 626, 644
 Ockham, W., 534, 559
 Ockham algorithm, 560
 octant, 747
 odometry, 783
 Oetzel, R. M., 596, 900
 offline cost, 61
 Ogasawara, G., 27, 520, 877, 880
 Oglesby, F., 313, 330, 874
 Olalainy, B., 390, 871
 Olawsky, D., 411, 888
 Olesen, K., 465, 860, 877, 888
 Oliver, R., 494, 888
 Olson, C., 754, 888
 Olum, P., 769, 872
 omniscience, 32, 558
 logical, 246
 one-shot decision, *see* decision, one-shot
 online algorithm, 796, 806, 806–809
 online cost, 61
 online navigation, 813
 Ono, N., 618, 871
 ONTIC, 312, 330
 ontological commitment, 165, 185, 417, 459
 ontological engineering, 217, 222
 ontological promiscuity, 258
 ontology, 222, 257
Op (STRIPS Operator), 344
 OP, 299, 303
 opacity, 244
 open-coding, 306
 open class, 664
 operability, 632
 operability and generality, 632
 operations research, 86, 87, 116, 117, 367, 498, 500
 OPERATOR, 72
 operator, 60, 85, 123
- abstract, 372
 primitive, 372
 operator expansion, *see* planning, hierarchical
 operator reduction, *see* planning, hierarchical
 OPERATORS, 60, 73, 126
 operator schema, 344
 OPS-5, 298, 314, 316
 optical character recognition, 657
 optical flow, 735, 750, 769
 optimality (of a search algorithm), 73, 85
 optimally efficient algorithm, 99
 optimal solution, 76
 optimistic prior, 610
 optimization problem, 619
 optimizer, peephole, 379
 OPTIMUM-AIV, 367, 369, 390
 Opus, 320
 OR, *see* operations research
 Or-Introduction, 172
 oracle, 856
 orderability, 474
 ordering constraints, 350
 ORDERINGS, 347, 372, 374
 ordinal utility, *see* utility, ordinal
 Organon, 179, 257
 Ortony, A., 720, 888
 Osherson, D. N., 559, 889
 Othello, 138, 623
 OTTER, 297, 310, 311, 311, 313, 330, 333
 outcome of a lottery, 473
 output unit, 571
 Overbeek, R., 294, 902
 overfitting, 542, 542–543, 572, 588
 overgeneration, 668
 overshooting, 577
 OWL, 298
- P**
- P* (probability), 420
P (probability vector), 421
 PAC, *see* probably approximately correct
 packed forest, 697, 703
 packed tree, 723
 Paek, E., 595, 870
 PAGE description, 36, 37, 248
 Paige, R., 330, 889
 Palay, A. J., 143, 889
 Palmer, R. G., 595, 875
 Palmieri, G., 594, 872, 889
 Pandemonium, 594
 Pang, C., 87, 868
 Panini, 15, 685
 Pao, C., 26, 903

- Papadimitriou, C., 749, 847, 879, 889
 Papert, S., 21, 24, 577, 578, 594, 595, 886
PARADISE, 140
 paradoxes, 259
 parallel distributed processing, *see* neural network
 Parallel Inference Machine, 308
 parallelism, 566
 AND-, 308
 OR-, 308
 parallel lines, 726
 parallel search, 117, 117
 paramodulation, 284, 293
PARENT-NODE, 72
 parent node, 72
 parity function, 533
 Parker, D., 595, 889
PARLOG, 329
 Parrot, Y., 390, 859
PARSE, 662, 663
 parse forest, 664
 parse tree, 658, 664, 701
 parsing, 658, 658, 664
 chart, 697
 partial derivative, 580
 partial order, *see* planning, partial-order
 partial plan, *see* plan, partial
 partition, 231, 795
 part of, 233
 part of speech, 658
 Partridge, D., 29, 889
 parts, 252
 Pascal's wager, 432, 493
 Pascal, B., 12, 14, 431
 Pascaline, 15
 Pasero, R., 328, 866
PASSIVE-RL-AGENT, 602, 605, 607
 passive learning, 599
 passives, 687
 Paterson, M., 889
 path, 60, 85
PATH-COST, 72
PATH-COST-FUNCTION, 60
 path cost, 60, 61, 85
PATHFINDER, 457, 458, 465
 pathmax, 98
 Patil, R., 332, 702, 720, 865, 869
 Patrick, B., 117, 889
 pattern matching, 330
 Paul, R. P., 811, 889
 paying, 255
 payoff function, 124, 418
PCFG, *see* grammar, context-free
PDP, *see* parallel distributed processing
 Peano, G., 212, 316, 889
 Pearl, J., 25, 97, 99, 116–117, 143, 435, 437, 465, 467, 497, 596, 868, 873, 879, 889, 895
PEDESTAL, 390
 Pednault, E. P., 390, 889
PEGASUS, 26
 Peirce, C. S., 212, 316, 323, 685, 889
 pen, 713
 Penberthy, J., 390, 861, 889
 Peng, J., 623, 889
 Pengi, 411
 Penrose, R., 825, 889
 Pentland, A., 36, 883
 Peot, M., 411, 465, 889, 895
PERCEPT, 48
 percept, 39
 perception, 30
 perceptron, 19, 571, 573, 593, 594
 convergence theorem, 20
 learning rule, 576, 593
 representational power, 21, 596
 percept sequence, 33
 Pereira, F., 304, 306, 685, 688, 693, 694, 720, 874, 889, 890, 900
 Pereira, L., 306, 900
PERFORMANCE-ELEMENT, 608, 609
PERFORMANCE-FN, 48
 performance element, 525, 526, 558, 562, 766
 performance measure, 32, 40, 47, 50, 416, 472
 Perrault, C., 720, 866
 perspective, 767
 perspective projection, 726, 735
PERT, 367–369
 Peters, S., 685, 869
 Peterson, C., 595, 890
 Petrie, T., 770, 861
 phenomenology, 828
 Philips, A., 116, 886
 Phillips, M., 26, 903
 Phillips, S., 87, 878
 Philo, 179
 philosophy, 3, 8–10, 817–841
 European, 8
 phone (speech), 757
 phoneme, 585
 phone number, 246
 phonetic alphabet, 758
 photogrammetry, 768
 photometry, 729
 photosensitive spot, 749
 phrase, 655
 phrase structure, 655, 685
 physicalism, 819, 839
 Picasso, P., 834
 pick (choice point), 856
 Pickwick, Mr., 831
 piecewise continuity, 740
 pigeons, 13
 Pillsbury, 696
PIM, *see* Parallel Inference Machine
 ping-pong, 29, 598
 pinhole camera, 725
 Pinker, S., 687, 890
 Pisa, tower of, 526
 pit, bottomless, 153
 Pitt, L., 560, 862
 Pitts, W., 16, 19, 563, 570, 594, 885
 pixel, 727
 place, 236
 Place, U., 839, 890
 plan, 347
 canned, 407
 complete, 349
 conditional, 410, 806
 consistent, 349, 349
 fully instantiated, 346
 initial, 347
 linear, 363
 noninterleaved, 363
 nonlinear, 363
 parameterized, 398
 partial, 345
 representation, 346
PLAN-ERS1, 390
PLANEX, 411, 787
 Plankalkul, 14
PLANNER, 23, 330, 402
 planning, 42, 140, 211, 342, 539
 abstraction, 389
 adaptive, 389
 ADL formalism, 390
 assembly, 792
 blocks world, 19
 bounded-error, 796
 case-based, 389
 conditional, 392, 393–398, 407, 412, 415, 500
 contingency, 392
 and DDNs, 519
 deferred, 393
 and execution, 403–406
 fine-motion, 802, 809
 formalization, 25
 hierarchical, 371–380, 389
 hierarchical decomposition, 374
 history, 363
 as logical inference, 341
 menu, 249–252
 multi-agent, 390
 partial-order, 337, 346, 355–356, 390, 407
 progression, 345, 365

- reactive, 411, 411
 regression, 345, 356, 363
 route, 18
 and scheduling, 369
 search spaces, 345–346
 situation calculus, 341–342
 situation space, 345
 speech acts, 654
 symbolic, 788
 total order, 346
 under uncertainty, 795, 843
 planning agent, *see* agent, planning
 planning variable, 400
 plan recognition, **654**, 718
 plasticity, 564
 Plato, 8, 178, 827, 838
 Plotkin, G. D., 646, 890
 ply, 124
 Pnueli, A., 259, 890
 Podelski, A., 329, 859
 poetry, 4, 682
 Pohl, I., 87, 116, 890
 poker hands, 433
 Poland, 240
 Polguere, A., 687, 885
 policy, 411, 498, **500**, 517, 519, 806
 optimal, 500
POLICY-ITERATION, **506**, 608
 policy iteration, **505**, 505–506, 520,
 603, 843
 policy loss, 505
 Polifroni, J., 26, 903
 Pollack, M., 720, 866
 Pollard, C., 686, 890
 Pólya, G., 94, 890
 polytree, **448**, 464
POMDP, *see* Markov decision
 problem
 Pomerleau, D. A., 26, 586, 587, 750,
 877, 890
POMDP (partially observable
 Markov decision problem),
 520
 Pooh, Winnie the, 218
POP, 355, 356, 356, 357, 358, 362,
 364–367, 369, 374, 384,
 391, 398, 404, 406, 412
POP-DUNC, 381, 384, 385, 390,
 401
POPLOG, 330
 Popper, K. R., 432, 559, 836, 890
 Port-Royal Logic, 471, 493
 Portuguese, 627
 pose, **734**, 752
 positive example, 534
 positivism, logical, 10
 possibility theory, **466**, 466
 possible threat, **354**, **357**, 357
 possible world, **260**, 795
 Post, E. L., 179, 890
 posterior probability, *see*
 probability, conditional
POSTSCRIPT, 706, 707
 Prade, H., 466, 869
 Pradhan, M., 444, 456, 890
 pragmatic interpretation, *see*
 interpretation, pragmatic
 pragmatics, 658
 Prata, A., 595, 870
 Pratt, V. R., 259, 890
 Prawitz, D., 291, 293, 890
 pre-editing, 692
 precompilation, 270
 precondition, 344
 disjunctive, 383
 universally quantified, 383
PRECONDITIONS, 402
 predecessor, 80
 predicate, 165, 674
 predicate calculus, *see* logic,
 first-order
 predicate symbol, 187, 211
PREDICT, 723
 predicting the future, 531, 553
 prediction-estimation process, **515**
 prediction phase, 509
PREDICTOR, 698–700, **702**, 702
 predictor, 698
 preference, 418, 473, 474, 483
 lexicographic, 496
 monotonic, 476
 preference independence, 483
 preferentially independent, 496
 premise, 167
 prenex form, 292
 preposition, 664
 president, 240
 Presley, E., 240
PRESS, 330, 331
 Price Waterhouse, 369
 Prieditis, A., 103, 116, 118, 887, 890
 primary colors, 730
 Princeton, 17, 704
Principia Mathematica, 17
PRINT, 856
 Prinz, D., 142, 890
 Prior, A. N., 258, 890
 prioritized sweeping, 607, 623
 priority queue, 623
 prior knowledge, **625**, 627, 636, 645
 prior probability, *see* probability,
 prior
 prismatic motion, 781
 Prisoner's Dilemma, 847
 prisoners, three, 435
 probabilistic model, 425
 probabilistic network, *see* belief
 network
 adaptive, 590
 probabilistic projection, 515
 probabilistic sensor models, 520
 probability, 12, 23, 25, 127
 alternatives to, 458
 assessment, 430
 axioms of, 422–423
 conditional, 418, **421**, 421–422,
 431, 440
 conjunctive, 440
 distribution, 139, **421**, 445
 history, 433
 judgments, 443
 of sun rising, 430
 prior, **420**, 420–421, 431
 theory, 417, 493
 probably approximately correct,
 553, 556, 560
PROBLEM, 60
 problem, 60, 85
 airport-siting, 496
 assembly sequencing, 70
 bandit, 623
 contingency, 59, 123
 cryptarithmic, 65, 91
 datatype, 60
 8-queens, 64, 83
 8-puzzle, 101, 103, 115
 8-queens, 86, 89
 exploration, 59
 halting, 277, 824
 inherently hard, 852–853
 intrinsically difficult, 106
 map-coloring, 91, 105
 missionaries and cannibals, 67
 monkey and bananas, 366
 multiple-state, 58, 66
 real-world, 63
 relaxed, 103
 robot navigation, 69
 route-finding, 93
 shoes-and-sock, 347
 shopping, 340
 single-state, 58, 60
 toy, 63
 travelling salesperson, 69
 VLSI layout, 69, 114
 problem-solving agent, *see* agent,
 problem-solving
 problem formulation, 56, 57
 problem generator, 526, 562
 problem solving, 22
 complexity, 342
 vs. planning, 338
 procedural attachment, 323
PROCESS, 311, **311**

- process, 237
 production, 40, 854
 production system, 297, 314
 product rule, 421
PROGRAM, 48
 programming language, 14, 160
 program synthesis, 411
 progression planner, *see* planning
 projection, probabilistic, 514
 Prolog, 197, 297, 304, 328, 363,
 644, 688
 Concurrent, 329
 parallel, 308
 Prolog Technology Theorem Prover,
 311, 330
 promotion, 353
 pronunciation, 585, 762
 proof, 160, 173, 266
 as search, 268
 by contradiction, 280
 checker, 312
 procedure, 11
 theory, 160, 165
 property, 185
PROPOSITIONAL-KB-AGENT, 177
 propositional attitude, 244
 propositional logic, *see* logic,
 propositional
 proprioception, 782
PROSPECTOR, 23, 466
 prosthetic limbs, 777
 Protagoras, 685
 protected link, 353
 protection, 350
 protection interval, 347
 protein structure, 641
 Provan, G. M., 444, 456, 890
 pruning, 123, **130**, 346, 542
 in contingency problems, 136
 in EBL, 631
 Psaltis, D., 595, 859, 870, 877
 pseudo-code, 855
 pseudo-experience, 607, 616
PSPACE, 811, 853
 psychological models, 566
 psychological reasoning, 261
 psychology, 3, 12-14
 experimental, 6, 13
 information-processing, 13
 psychophysics, 769
 public key encryption, 313
 Puget, J.-E., 646, 893
 Pullum, G. K., 162, 656, 686, 872,
 890
PUMA, 781, 782
 punishment, 528
 Purdom, P., 116, 891
 Putnam, H., 50, 286, 292, 432, 839,
 867, 891
 Pylyshyn, Z. W., 839, 891
- Q**
 $Q(a, i)$ (value of action in state), 612
 Q-learning, 599, 623, 624
Q-LEARNING-AGENT, 614
 Q-value, 612
 QA3, 212, 363
 QALY, **480**, 494
QLISP, 330
 qualia, 821, 836, 840
 qualification problem, 207, 213,
 415, 830
 qualitative physics, 233, **260**
 qualitative probabilistic network,
 465, 482
 quantified term, 676, 677
 quantifier, 165, **189**
 existential, 191-192
 nested, 192-193
 scope, 673, 686
 universal, 189-191
 quantization factor, 758
 quantum theory, 826
 quasi-logical form, **676**, 686
 query (logical), 201
 query variable, 445
 question, 711
 queue, 72
QUEUEING-FN, 72, 73
 Quevedo, T., 142, 810
 quiescence, **129**, 142
 Quillian, M. R., 257, 331, 891
 Quine, W., 179, 213, 232, 258, 891
 Quinlan, E., 720, 891
 Quinlan, J., 540, 559, 561, 616, 642,
 644, 646, 891
 Quirk, R., 685, 891
- R**
 $R(i)$ (Reward), 603
 R1, 24, 316
 Rabiner, L. R., 688, 766, 770, 891
 racing cars, 846
 Raibert, M. H., 778, 891
 Raiffa, H., 479, 484, 494, 879
 ramification problem, 207
 Ramsay, A., 330, 861
 Ramsey, F. P., 432, 493, 891
 randomization, 34
 random restart, 119
 random variable, 420, 441
 Boolean, 420
 Rao, B., 27, 520, 877, 880
 Raphael, B., 19, 115, 116, 646, 810,
 863, 874, 891
- rapid prototyping, 304
RAPTS, 330
 rational action, 518
 rational agent, *see* agent, rational
 rationalism, 827
 rationality, 4, 847
 calculative, 845
 limited, 8
 perfect, 8, 845
 rational speaker, 716
 Ratner, D., 87, 897
 rats, 13
 Rau, L., 696, 877
 Rayner, M., 632, 894
 ray tracing, 729
RBDTL, 635, 636, 646
 RBL, *see* relevance-based learning
 reactive planner, *see* planning,
 reactive
 reading signs, 254
 Reagan, R., 706
 real-time AI, *see* artificial
 intelligence, real-time
 real-world problems, *see* problem,
 real-world
 reasoning, 18, 163
 about location, 206
 default, 326, 458-460, 466
 goal-directed, 140
 intercausal, 461
 metalevel, 330
 uncertain, 25, 658, 682
 with uncertainty, 760
 Reboh, R., 330, 894
 recognition hypothesis, 755
 recognizable set, 795, 795
RECOMMENDATION, 57
 record player, 812
 recurrent, *see* neural network,
 recurrent
 recursive definitions, 643
 Reddy, R., 770, 870, 883
 Redfield, S., 595, 890
 reductio ad absurdum, 280
 reduction, 12, 184, 853
 redundancy, 791
 redundant step, 405
 reference class, **430**, 432
 referential opacity, 260
 referential transparency, 244
 refinement operator, 345
 reflectance, 729, 743
 reflectance map, 745
REFLEX-AGENT-WITH-STATE, **43**, 49
REFLEX-LEARNING-ELEMENT, 529,
 530
REFLEX-PERFORMANCE-ELEMENT,
 529, 530, 530

- refutation, 280, 283, 293, 311
 refutation completeness, 286
 regression, nonlinear, 572
 regression planner, *see* planning
 regret, 479
 Reichardt, J., 810, 891
 Reichenbach, H., 432, 892
 Reif, J., 796, 811, 864, 892
 reification, 230, 230, 319, 435
 reinforcement, 528, 528, 598
 reinforcement learning, 528, 598, 829
 Reingold, E., 116, 862
 Reiter, R., 213, 259, 459, 466, 892
 Reitman, W., 145, 892
RELATED-TO?, 322
 relation, 185
 relative clause, 710
 relaxed problem, *see* problem
 relevance, 169, 628, 646
 relevance-based learning, 628, 634, 645
RELS-IN, 321
RELS-OUT, 321, 322
REMAINDER, 57
 removal from a KB, 298
REMOVE-FRONT, 72, 73
 Remus, H., 145, 882, 892
 renaming, 273
 Renyi, A., 892
 Renyi, A., 431
 repeatability, 783
 repeated states, 82
 replanning, 367, 371, 392, 401–403, 407, 412
REPLANNING-AGENT, 402, 402
 representation, *see* knowledge representation
 representation languages, 157
 representation theorem, 480
REPRODUCTION, 620
REQUEST, 490
 Rescher, N., 261, 892
RESET-TRAIL, 307
 Resnik, P., 688, 892
 resolution, 18, 21, 172, 172, 265, 277, 290, 297, 648
 algorithm, 277
 completeness proof, 286
 generalized, 278
 input, 285
 inverse, 639, 639–641, 646
 linear, 285
 strategies, 284–286
 resolution closure, 287
RESOLVE-THREAT, 382
RESOLVE-THREATS, 356, 356, 357, 358, 374, 375, 382, 382, 384, 385, 385
 resolvent, 279, 639
 resource (in planning), 386–389, 391
 response, 13
REST, 273, 275, 303, 338, 395, 402
 restaurant, *see* learning, restaurant problem
 restrictive language (in planning), 342
Result (situation calculus), 342
 rete, 314
 retina, 724
RETRACT, 321, 325, 326, 412
 retraction, 325
REWORLD, 601
 reward, 528, 598, 603
 reward-to-go, 601
 reward function, 503
 rewrite rule, 310, 333, 655, 854
REWRITES-FOR, 699, 702
 Rice, T., 494, 886
 Rich, E., 5, 592
 Rieger, C., 23, 892
 Riesbeck, C. K., 23, 328, 865, 894
 right thing, doing the, 4, 8
 Ringle, M., 840, 592
 risk aversion, 478
 risk neutrality, 478
 risk seeking, 478
 Rissanen, J., 560, 892
 Ristad, E., 690
 Ritchie, G. D., 647, 892
 Rivest, R., 560, 595, 853, 862, 866, 892
RMS, *see* root mean square
 Roach, J., 329, 892
 roadmap, 800
 Roberts, D. D., 331, 892
 Roberts, L., 769, 892
 Roberts, M., 143, 862
 Robin, C., 218
 Robinson, A., 292
 Robinson, G., 293, 902
 Robinson, J., 18, 277, 286, 293, 328, 592
 Robo-SoAR, 788
 robot, 773, 773, 809
 architecture, 786–790
 autonomous, 773
 cleaning, 812
 holonomic, 778
 mobile, 775, 812
 navigation, 69
 nonholonomic, 778
 robot game, 811
 robotics, 6, 363, 512
 behavior-based, 789
 father of, 810
 laws of, 814
 Robot Institute of America, 773
 robot reply, 832
 Rochester, N., 14, 17, 18
 Rock, I., 770, 892
 rollup (of a DBN), 515
 Romania, 56
 root mean square error, 505
 Rorty, R., 840, 892
 Rosenberg, C., 585, 895
 Rosenblatt, E., 19, 576, 594, 769, 572, 892
 Rosenblitt, D., 364, 884
 Rosenbloom, P., 26, 316, 645, 843, 881, 887, 898
 Rosenbluth, A., 117, 885
 Rosenbluth, M., 117, 885
 Rosenfeld, E., 28, 860
 Rosenholtz, R., 743, 769, 883
 Rosenschein, J., 180, 892
 Rosenschein, S. J., 180, 788, 878, 892
 Rosenthal, D. M., 840, 892
 Ross, K., 720, 899
 Ross, S. M., 433, 892
 rotary motion, 781
 rotation, 734
 Rothwell, C., 769, 893
 Roukos, S., 687, 562
 Roussel, P., 328, 866, 893
 route finding, 68
 Rouveirrol, C., 646, 893
 Rowe, N. C., 29, 893
 RSA (Rivest, Shamir, and Adelman), 313
 Rubik's cube, 86, 103, 735
 Ruján, P., 595, 883
 rule
 causal, 212
 condition-action, 40, 145, 612
 diagnostic, 212
 dotted, 697
 if-then, 40, 167
 implication, 167
 of thumb, 94
 situation-action, 40, 498
 uncertain, 461
RULE-ACTION, 41, 43
 rule-based systems, 458
 with uncertainty, 460–462
RULE-LHS, 666
RULE-MATCH, 41, 43
RULE-RHS, 666
 rule memory, 314
RULES, 666, 699

- rule schema, 671
 Rumelhart, D. E., 24, 595, 893
RUN-ENVIRONMENT, 47, 48
RUN-EVAL-ENVIRONMENT, 47, 48,
 48
RUN-NETWORK, 581
RUNNING-AVERAGE, 602, 605
 runtime variable, 400
 Rusconi, E., 466, 893
 Russell, B., 10, 16, 17, 291, 292,
 825, 901
 Russell, J., 494, 893
 Russell, S. J., 27, 50, 117, 143, 309,
 330, 520, 596, 646, 843,
 844, 846, 867, 874, 877,
 880, 893, 898
 Russian, 21
 Ruzzo, W. L., 720, 873
 Ryder, J. L., 145, 893
- S**
- S-set, 550
 sabotage, 661
 Sacerdoti, E. D., 330, 363, 389, 893,
 894
 Sachs, J., 162, 894
 Sacks, E., 260, 894
 Saenz, R., 720, 899
 safety clearance, 797
 Sag, I., 686, 720, 872, 878, 890
 Sagalowicz, D., 330, 894
 Sager, N., 685, 594
 SAINT, 19
 St. Petersburg paradox, 476, 494
 Salton, G., 688, 894
 SAM, 297, 313, 330
 Samad, T., 595, 874
 Sammut, C., 539, 894
 sample complexity, *see* complexity,
 sample
 sampling rate, 758
 Sampson, G., 688, 872
 Samuel, A., 17, 18, 138, 143, 559,
 600, 617, 622, 823, 829, 894
 Samuelsson, C., 632, 894
 Sanna, R., 594, 872, 889
 Sanskrit, 256, 331, 685
 Sapir-Whorf hypothesis, 162
 Saraswat, V. A., 329, 894
 SAT, *see* satisfiability problem
 satisfiability, 164
 satisfiability problem, 182
 saturation, 287
 Satyanarayana, S., 595, 859
 Savage, L. J., 12, 424, 432, 493, 894
 sawing-the-lady-in-halftrick, 817
 say, 663
 Sayre, K., 818, 894
- scaled orthographic projection, 726
SCANNER, 699, 700, 702, 702
 scanner (in chart parsing), 698
 scene, 725
 scent marking, 653
 Schabes, Y., 687, 894
 Schaeffer, J., 138, 142, 144, 884,
 594
 Schalkoff, R. J., 5, 894
 Schank, R. C., 23, 894
 scheduling, 367, 369
 job shop, 369
 space missions, 369
 Schemes, R., 596, 597
 schema, 234
SCHEME, 329, 856
 Scherl, R., 259, 894
 Schmolze, J., 332, 894
 Schoenberg, L., 594, 859, 887
 Schoenfinkel, M., 329, 894
 Schofield, P., 86, 894
 Schoppers, M. J., 411, 894
 Schröder, E., 292, 894
 Schubert, L., 258, 686, 877
 Schwartz, J. H., 595, 878
 Schwartz, W., 432, 873
 Schweikard, A., 794, 901
 Schwuttke, U., 26, 594
 scientific discovery, 559
SCISOR, 696
 script, 234
 Scriven, M., 839, 895
 scruffy, 25
 SEAN, 330
SEARCH, 57
 search, 22, 42, 56, 85
 A*, 96-101
 completeness, 100
 complexity, 100
 optimality, 99
 alpha-beta, 129-133, 141, 143,
 844
 B*, 143
 backtracking, 84, 309, 771
 bidirectional, 80
 blind, 73
 constraint satisfaction, 83-84
 current-best-hypothesis, 546
 cutting off, 129
 general, 85
 greedy, 93, 115, 118
 heuristic, 73, 115
 hill-climbing, 111, 595
IDA*, 106
 informed, 73, 92
 iterative deepening, 79, 85, 87,
 129, 312
 iterative deepening A*, 106
- iterative improvement, 115
 minimax, 124-126, 130, 139, 141
 quiescence, 129
 random-restart, 595
 simulated annealing, 113
SMA*, 107
 uninformed, 73
- search cost, 61
 search node, *see* node, search
 search strategy, 70, 73
 search tree, 71
 Searle, J. R., 565, 685, 819,
 831-835, 837, 839, 840, 895
- segment (of discourse), 717
 segmentation (of an image), 734
 segmentation (of speech), 757
 Sejnowski, T., 585, 594, 595, 617,
 575, 895, 899
- SELECT-NONPRIMITIVE**, 374
SELECT-SUB-GOAL, 374, 382-384,
 385, 391
- SELECT-SUBGOAL**, 355, 356, 356
- SELECTION**, 620
- Self, M., 865
- Selfridge, M., 687, 862
- Selfridge, O. G., 17, 594, 895
- Selfridge, P., 325, 868
- Sells, P., 686, 895
- Selman, B., 117, 184, 331, 466, 878,
 879, 895
- SEM-NET-NODE**, 321
- semantic interpretation, 672-678,
 685, 689, 720
- semantic network, 298, 316
 partitioned, 323
- SEMANTICS**, 662, 663
- semantics, 28, 157, 165, 672
 causal, 821
 compositional, 672
 intersective, 708
 logical, 178
 preference, 688
- semidecidability, 277
- semidynamic, *see* environment,
 semidynamic
- Seneff, S., 26, 903
- sensing action, *see* action, sensing
- sensitivity analysis, 447, 492
- sensor, 31, 724
 abstract, 795
 cross-beam, 785
 depth, 785
 force, 784
 lane position, 513
 parallel-beam, 785
 structured light, 785
 tactile, 724, 784
- sensor failure, 512

- sensor fusion, 512
 sensor model, 510, 510–513
 stationary, 510
 sensory stimuli, 724
 sentence
 analytic, 164
 atomic, 167, 189, 193, 211, 279
 complex, 167, 211
 in a KB, 151, 178
 in a language, 655
 logical, 186
 necessarily true, 163
 as physical configuration, 158
 quantified, 211
 valid, 163
 separability, *see* utility function,
 separable
 sequence prediction, 90
 sequent, 291
 sequential decision, *see* decision,
 sequential
 sequential decision problems, 520
 Sergot, M., 259, 880
 Sestoft, P., 645, 878
 set of support, 285, 310
 sets, 199
 Settle, L., 313, 330, 874
 Shachter, R., 465, 480, 494, 520,
 895, 898
 shading, 735, 743–745
 Shafer, G., 466, 467, 895
 Shaham, R. W., 840, 862
 Shakey, 19, 360, 363, 365, 411, 787,
 810
 Shalla, L., 293, 902
 Shankar, N., 313, 826, 895
 Shannon, C., 16, 17, 122, 142, 540,
 559, 895
 shape, 734
 shape from shading, 769
 shape from texture, 769
 Shapiro, E., 329, 330, 646, 895, 897
 Shapiro, S. C., 28, 94, 323
 Sharp, D., 596, 866
 Sharpies, M., 29, 895
 Shavlik, J., 560, 869, 896
 Shaw, J., 94, 143, 292, 329, 888
 sheep shearing, 775
 Shenoy, P., 466, 896
 Shepard, E. H., 886
 Shieber, S. M., 328, 686, 688, 890,
 896
 shipping lanes, 793
 Shirayagi, K., 145, 896
 shoes and socks, 364
 Shoham, Y., 213, 258, 259, 466,
 876, 896
 shopping, 227, 247–255, 371, 392
 shopping list, 249
 short-term memory, 316, 570
 shortest path, 87, 119
 Shortliffe, E. H., 22, 466, 864, 896
 SHRDLU, 23, 330
 Shrobe, H., 329, 870
 SHUNYATA, 826
 Shwe, M., 465, 896
 Sidner, C., 719, 720, 874
 Siekmann, J., 294, 896
 Sietsma, J., 595, 896
 SIGART, 28
 sigmoid function, 569, 580, 583
 signal processing, 758–759
 sign function, 569
 Siklossy, L., 389, 896
 silhouette curve, 800, 800
 silhouette method, 800
 similarity network, 457
 Simmons, R., 778, 896
 Simon, H. A., 6, 10, 17, 28, 50, 86,
 94, 137, 143, 292, 313, 330,
 881, 888, 896
SIMPLE-COMMUNICATING-AGENT,
 663
SIMPLE-PLANNING-AGENT, 338
SIMPLE-POLICY-AGENT, 501, 521
SIMPLE-PROBLEM-SOLVING-AGENT,
 57
SIMPLE-REFLEX-AGENT, 41
SIMPLIFY, 311
SIMULATED-ANNEALING, 113
 simulated annealing, 111, 113, 117,
 119
 simulation of intelligence, 834
 simulation of world, 821
 simultaneity, 227
 Singh, S., 623, 861
 single-state problem, *see* problem,
 single-state
 singly connected network, 447
 sins, seven deadly, 95
 SIPE, 371, 387, 389, 390
 SIR, 19
 Siskind, J. M., 687, 896
SITUATED-PLANNING-AGENT, 408
 situated agents, 830
 situated automaton, 788
 situated language, *see* language,
 situated
 situatedness, 26, 403
 situation, 204, 472
 situation-action mapping, 828
 situation calculus, 204, 212, 213,
 216, 227, 234, 259, 361,
 363, 368, 390, 400, 659
 skeleton, **798**
SKELETON-AGENT, 38
 skeletonization, 796, 798, 809
 Skinner, B. E., 15, 50, 896
 Skolem, T., 212, 292, 896
 Skolem constant, 292, 679
 Skolem function, 282, 292, 400
 skolemization, **281**, 292
 Slagle, J. R., 19, 142–144, 896, 897
 slant, 734
 Slate, D. J., 87, 897
 Slater, E., 142, 897
 sliding-block puzzle, 63
 Sloman, A., 330, 839, 861, 897
 Slovic, P., 4, 878
 SMA*, 107
 Smallwood, R., 520, 897
 Smith, D. E., 295, 309, 330, 411,
 872, 889, 897
 Smith, D. R., 330, 897
 Smith, J., 494, 888
 Smith, R., 559, 864
 Smith, S., 369, 871
 smoothing, 730
 SNARC, 16
 SNEPS, 298, 323
 SNLP, 364
 snow, 162
 SOAR, 298, 645, 788, 790, 843, 844
 Société de Linguistique, 653
 Socrates, 6, 8, 9
 Socratic reasoning, 312
 Soderland, S., 364, 897
 softbot, **36**, 412, 773
 software agent, 36
 Solomonoff, R., 17, 559, 897
 solution, **56**, 60, 85, 349, 358
 in planning, 342, **349**
SOLUTION?, 356, 358, 374
 soma, 564
 Sompolsky, H., 595, 860
 sonar, 501, 512, 784
 Sondik, E., 520, 897
 sonnet, 830
 soul, 838
 sound (inference), 159, 178
 sour grapes, 32, 526
 space complexity, *see* complexity
 spacecraft assembly, 390
 SPANAM, 692
 Sparck Jones, K., 688, 874
 sparse system, 441
 spatial reasoning, 260
 spatial substance, *see* substance
 speaker, 652
 speaker identification, 759
 specialization, 546, 547
 specificity preference, 459
 spectrophotometry, 730
 specularly reflected light, 729

- SPEECH-PART, 663
 speech act, 652, 652–654, 684, 720
 indirect, 652
 interpretation, 663
 speech recognition, 25, 26, 657,
757, 757–767
 speech synthesis, 657
 speech understanding, 757
 speedup learning, 527
 spelling correction, 704, 720, 722
 Spiegelhalter, D. J., 465, 560, 596,
 881, 885, 897
 spies, 163
 SPIKE, 370, 390
 spin glass, 595
 Spirites, P., 596, 897
 Sproull, R. F., x, 495, 870
 SPSS, 370
 SQRT, 34
 square root, 34
 SRI, 19, 343, 360, 363, 494, 810
 Srivastava, M., 313, 897
 SSD, *see* sum of squared differences
 stability
 dynamic, 778
 static, 778
 Stader, J., 390, 859
 staged search, 117
 STAHL, 647
 Stamper, R., 466, 899
 standardizing apart, **271**, 294
 Stanford University, 17, 22, 457
 Stanhope Demonstrator, 292
 stapler, 728
 START, 405–408, 666
 start symbol, 854
 STATE, 72, 601
 state
 representation, 339
 state (in a chart), 697
 state (in the world), 56
 state (process), 237
 STATE-DESCRIPTION, 337, 338, 395,
 402, 408
 state evolution model, 514
 States, D., 877
 state set space, 60
 state space, 60, 85, 790
 state variable, 508
 static, *see* environment, static
 static (variable), 856
 static universe, 383
 stationarity (for rewards), 507
 stationarity (in PAC learning), 553
 statistical mechanics, 24, 595
 STATLOG, 560
 Steel, S., 411, 860
 Steele, G. L., 329, 330, 868, 897
 steering vehicles, 780
 Stefik, M. J., 390, 897
 Stein, P., 143, 879
 step function, 569
 stepper motor, 783
 STEPS, 372, 374
 stereogram, random dot, 768
 stereopsis, binocular, 735
 Sterling, L., 330, 897
 Sternberg, M., 641, 879, 887
 Stevens, K., 769, 897
 Stickel, M. E., 258, 293, 311, 330,
 720, 876, 897
 stimulus, 13
 Stob, M., 559, 889
 stochastic dominance, *see*
 dominance, stochastic, 493
 stochastic simulation, **453**, 455, 464,
 515
 Stockman, G., 143, 897
 Stoic school, 179, 258, 291
 Stokes, I., 390, 859
 Stolcke, A., 687, 897
 Stone, H., 116, 897
 Stone, J., 116, 897
 Stone, P., 559, 877
 Stonebraker, M., 219, 897
 STORE, 299, 300, 302, 334
 Story, W., 86, 878
 stotting, 659
 Strachey, C. S., 16, 142, 897, 899
 straight-line distance, 93
 Strat, T., 466, 893
 strategy (in a game), 124
 strawberries, enjoy, 823
 string (in a language), 655
 string (in logic), 245
 STRIPS, 343, 360, 363, 365, 389,
 390, 400, 401, 411, 412,
 645, 787
 STRIPSlanguage, 343–345, 367
 Strohm, G., 390, 871
 strong AI, **29**, 831–834, 839
 structure (composite object), 234
 Stubblefield, W. A., 5, 883
 Stuckey, P., 329, 877
 STUDENT, 19
 stuff, 242
 Stutz, J., 865
 subcat, *see* subcategorization list
 subcategorization, 670, 669–671
 list, 670
 subevent, 235
 subject-aux inversion, 711
 subjective case, 668
 subjectivism, 13, **430**
 subplans
 interaction between, 341
 Subramanian, D., 646, 846, 893, 898
 SUBS, 321, 322
 SUBSEQUENCE, 666
 SUBSET?, **322**, 322
 SUBST, 265, 273
 substance, 228, 241–243
 spatial, 242
 temporal, 242
 substitutability (of lotteries), 474
 substitution, 201, 265, 270
 subsumption (in description logic),
 323
 subsumption (in resolution), 286
 subsumption architecture, 411
 SUCCESSOR, 76
 successor-state axiom, *see* axiom,
 successor-state
 successor function, 60
 SUCCESSORS, 107, 110, 132
 Suermondt, H. J., 465, 876
 Sugihara, K., 749, 769, 898
 SUMMATION, 851, 852
 summer's day, 831
 sum of squared differences, 735
 Sundararajan, R., 329, 892
 Sun Microsystems, x
 sun rising, 430
 Superman, 161, 244
 SUPERS, 321, 322
 supervised learning, 528, 829
 SUPPORT-EXCEPT, 451, 452, 452
 sure thing, 478
 surface patches, 746
 surface structure, 686
 Sussman, G. J., 330, 363, 868, 898
 Sussman anomaly, 365
 Sutherland, G., 22, 864
 Sutton, R., 603, 623, 861, 898
 Svartvik, J., 685, 891
 Swade, D. D., 15, 898
 Swedish, 29
 syllogism, 6, 291
 Symantec, 694
 symbol
 dynamic, 660
 static, 660
 symbolic differentiation, 333
 symbolic integration, 552
 synapse, 564
 synchronous, 209
 syntactic ambiguity, 720
 syntactic sugar, 200
 syntactic theory (of knowledge), 245
 syntax, 23, 165
 syntax, logical, 157, 178
 synthesis, 313
 deductive, 313
 synthesis of algorithms, 313

- system gain, 507, 519
 systems reply, 832
SYSTRAN, 692
Szeliski, R., 736
- T**
T(cat,i) (liquid event), 237
T-SCHED, 390
TABLE-DRIVEN-AGENT, 38, 39
 table tennis, *see* ping-pong
 tabu search, 117
TACITUS, 258
 tactile sensor, *see* sensor, tactile
Tadepalli, P., 646, 898
TAG (Tree-Adjoining Grammar),
 687
Tait, P., 86, 898
Taki, K., 329, 888, 898
Talos, 810
Tambe, M., 316, 645, 859, 898
Tanimoto, S., 29, 898
Tarjan, R. E., 853, 898
Tarski's world, 213
Tarski, A., 11, 212, 685, 898
Tarzan, 68, 707
Tash, J. K., 520, 898
 taskability, 790
 task network, 363
Tate, A., 25, 363, 364, 369, 389,
 390, 859, 861, 866
Tatman, J., 520, 898
TAUM-METEO, 692, 720
 tautology, 164
Tawney, G., 142, 866
 taxi
 in Athens, 435
 automated, 526
 environment, 39
 taxonomic hierarchy, 23, 228, 230
 taxonomic information, 252
 taxonomy, 230, 257
Taylor, C., 560, 885
Taylor, R., 795, 810, 883, 898
TD, *see* temporal-difference
TD-UPDATE, 604, 605
 teacher, 528, 598
TEAM, 694, 720
TEIRESIAS, 330
 telepathic communication, 660
 telephone number, 87, 246
 telepresence, 776
 television, 652
TELL, 152, 153, 214, 245, 273, 298,
 299, 320, 323, 660
Teller, A., 117, 885
Teller, E., 117, 885
Temperley, H., 594, 866
 temporal-difference equation, 604
- temporal constraints, 388
 temporal differencing, 622
 temporal logic, *see* logic, temporal
 temporal substance, *see* substance
Tenenberg, J., 391, 899
Tennant, H., 720, 899
 term (in IR), 695
 term (in logic), 186, 188
TERMINAL-TEST, 126
TERMINAL?, 601–602, 605, 608, 614
 terminal state, 124, 598
 terminal symbol, 655, 854
 terminal test, 124
 termination condition, 803
 terminological logic, *see* logic,
 terminological
 term rewriting, 293
Tesauro, G., 139, 615, 617, 899
Tesla, N., 810
 test set, 538, 538
 texel, 743
 text, 715
 text categorization, 695–696
 text interpretation, 694
 texture, 735, 742, 742–743
 texture gradient, 743, 769
Thag, 626
 thee and thou, 664
THEN-PART, 395
THEO, 788, 790, 843, 844
 theorem, 198
 theorem prover, 5, 297, 310–313
 as assistants, 312
 theorem proving, 363, 411
 mathematical, 20, 29
 theory of information value, 609,
 830, 844
 theory resolution, 293
 thermostat, 820
 Theseus, 559
 thingification, 230
Thomason, R., 685, 899
Thomré, J., 27, 879
Thompson, C., 720, 899
Thompson, K., 137, 144, 866
Thompson, S., 719, 720, 883
Thorndike, E. L., 13
Thorne, J., 686, 899
Thorpe, C., 587, 877
 thought, *see* reasoning
 laws of, 7
 threat, 353
 threat resolution, 353, 382
 3-CNF, 182
 3-D information, 734
 3SAT, 84, 182
 Tic-Tac-Toe, 123, 124, 145, 147
 tiling algorithm, 573
- tilt, 734
 Timberline workshop, 364
 time, 258
 time complexity, *see* complexity
 time expressions, 722
 time interval, 238, 259, 262
 time machine, 365
 time slice, 515
Tinsley, M., 138
TMS, *see* truth maintenance system
Todd, B., 466, 899
Todd, P. M., 595, 886
 tokenization, 703
Tomasi, C., 737, 899
 tomato, 762
 tongue, 762
Torrance, S., 29, 895
TOSCA, 369
 total cost, 61
 total order, *see* planning, total order
 touch, *see* sensor, tactile
Touretzky, D. S., 331, 596, 899
 towers of Hanoi, 793
 toy problem, *see* problem, toy
 trace, *see* gap
 tractability of inference, 324
 trading, 263
 trail, 306
 training
 curve, 580
 sequence, 600
 set, 534, 538
 on test set, 538
 transfer motion, 793
 transfer path, 793
 transition function, 606
 transition model, 499, 502
 transitivity (of preferences), 473,
 474
 transitivity of implication, 278
 transit motion, 793
 transit path, 793
 translation, 29
 travelling salesperson problem, 69,
 69, 116, 119
 tree model, 132
Treloar, N., 142, 894
 triangle table, 401
 trie, 704
 trigram, 761
 trihedral solid, 746
 tropism, 201
 truth, 163, 178, 189
 truth-functionality, 460, 464
 truth-preserving (inference), 159
 truth maintenance, 326
 truth maintenance system, 326, 332,
 460, 839

- assumption-based, 327, 332
 justification-based, 326
 truth table, 168, 179, 180
 TSP, *see* travelling salesperson problem
 Tuck, M., 788, 881
 tuple, 187
 Turing, A., 5, 11, 14, 16, 28, 122, 142, 292, 465, 691, 823, 824, 826, 827, 831, 836, 839, 840, 899
 Turing award, 466, 853
 Turing machine, 11, 560, 827
 Turing Test, 5, 7, 28, 37, 652, 823, 825, 830
 total, 6
 Turk, 141
 turning radius, 780
 Tversky, A., 4, 443, 479, 878, 899
 TWEAK, 363
 Tweety, 320
 2001: A Space Odyssey, 465
 TYPE, 663
 type, *see* category
 type (in planning), 383
 type predicate, 308
 typical instance, 232
- U**
U (utility), 472
 UCPOP, 390
 Ueda, K., 329, 899
 Ulam, S., 143, 879
 Ullman, J. D., 328, 853, 859, 899
 Ullman, S., 753, 769, 877, 899
 uncertainty, 23, 25, 66, 229, 415, 462, 830, 843
 in games, 123
 model, 805
 reasoning with, *see* reasoning
 rule-based approach to, 458
 summarizing, 417
 unconditional probability, *see* probability, prior
 unconsciousness, 564
 undecidability, 11, 29
 understanding, 716
 understanding problem, 654
 ungrammatical sentences, 669
 unicorn, 181
 unification, 270, 270–271, 290, 314, 328, 383
 algorithm, 302–303
 in a DCG, 686
 and equality, 284
 unifier, 270
 most general (MGU), 271, 294
 uniform-cost search, 85
 uniform convergence theory, 560
 uniform cost search, 75
 uniform prior, 589
 UNIFY, 270, 271, 273, 275, 284, 303, 303, 307, 358, 385
 UNIFY-LISTS, 303, 303, 303
 UNIFY-VAR, 303, 303, 306
 Unimation, 810
 uninformed search, 73
 unique main subaction, 378, 389
 unit (of a neural network), 567
 unit clause, 285
 unit preference, 285
 Unit Resolution, 172
 units function, 231
 Universal Elimination, 266
 universal field, AI as a, 4
 universal plan, 411
 UNIX, 706
 unsatisfiability, 164
 unsupervised learning, 528
 UOSAT-II, 370, 390
 UPDATE, 601, 602
 UPDATE-ACTIVE-MODEL, 608
 UPDATE-FN, 48
 UPDATE-MEMORY, 38
 UPDATE-STATE, 42, 43, 57
 upward solution, 376, 389, 391
 URP, 494
 Urquhart, A., 261, 892
 Uskov, A., 144, 859
 Utgoff, P. E., 552, 559, 886
 utilitarianism, 10
 act, 847
 rule, 847
 UTILITY, 126
 utility, 23, 44, 123, 418
 expected, 50, 135, 419, 471, 472, 476, 516
 independence, 484
 maximum expected, 419
 of money, 476–478
 node, 485
 normalized, 478
 ordinal, 476
 theory, 418, 473–475, 493
 multiattribute, 493
 utility function, 45, 50, 124, 471, 474, 499, 615
 additive, 502
 multiplicative, 484
 separable, 502
 utility scale, 478–480
 utterance, 652
 UWL, 411
- V**
 vacuum, *see* agent, vacuum
 vacuum tube, 14, 16
 vacuum world, 32, 51, 87, 90, 412, 624
 vagueness, 459, 681
 Valiant, L., 560, 899
 valid conclusions, 165
 VALIDITY, 182
 validity, 163, 178
 valid sentences, 169
 VALUE, 111–113, 126
 value, backed-up, 124
 VALUE-DETERMINATION, 506
 VALUE-ITERATION, 503, 504, 504, 506, 608
 value determination, 505, 603, 624
 value function, 476
 additive, 483
 value iteration, 502, 502–505, 520
 value node, *see* utility node
 value of computation, 844
 value of information, 487–490, 493, 497, 502
 value of perfect information, 488
 van Benthem, J., 261, 899
 van Doorn, A., 769, 879
 van Harmelen, E., 645, 899
 van Heijenoort, J., 293, 899
 vanishing point, 726
 VanLehn, K., 686, 899
 van Roy, P., 304, 307, 329, 899
 Vapnik, V., 560, 899
 variabilization (in EBL), 630
 variable (in a CSP), 83
 variable (in a grammar rule), 668
 variable (logical), 190
 VARIABLE?, 303
 variable binding, 357, 374
 Vasconcellos, M., 692, 900
 Vaucanson, J., 810
 Vecchi, M., 117, 879
 vector-space model, 695
 vector quantization, 759, 762, 763
 Veloso, M., 646, 900
 Vendler, Z., 259, 900
 Venn diagram, 422
 verb phrase, 655
 Vere, S. A., 389, 900
 verification, 313
 hardware, 226
 VERSION-SPACE-LEARNING, 549, 552
 VERSION-SPACE-UPDATE, 549, 549, 550
 version-space learning, 530
 version space, 549, 550
 version space collapse, 551
 vertex (of a chart), 697
 vertex (polyhedron), 746

- vervet monkeys, 651
 very long names, 218
 virtual reality, 776
 visibility graph, 799
 vision, 6, 7, 12, 19, 725–767
 Vitanyi, P. M. B., 560, 882
 Viterbi algorithm, 765, 772
 VLSI layout, 69, 114
 vocabulary words, 664
 vocal tract, 762
 von Linne, C., 258
 von Mises, R., 432, 900
 Von Neumann, J., 12, 14, 16, 142,
 493, 594, 847, 900
 von Winterfeldt, D., 493, 900
 Voorhees, E., 720, 900
 Voronoi diagram, 800
 Voyager, 26, 370
 VP, *see* verb phrase
 VPI, *see* value of perfect
 information
- W**
- $W_{j,i}$ (weight on a link), 568
 Waibel, A., 770, 900
 Walden, W., 143, 879
 Waldinger, R., 213, 293, 330, 363,
 894, 900
 Wall, R., 685, 869
 Wall Street Journal, 696
 Walter, G., 810
 Waltz, D., 19, 749, 900
 WAM, *see* Warren Abstract Machine
 Wand, M., 146, 900
 Wang, E., 260, 898
 Wang, H., 6, 179, 900
 Wanner, E., 162, 687, 900
 war games, 849
 Warmuth, M., 560, 862, 891
 WARPLAN, 363, 411
 WARPLAN-C, 411
 Warren, D. H. D., 304, 306, 328,
 363, 411, 685, 890, 900
 Warren Abstract Machine, 306, 328
 washing clothes, 722
 Washington, G., 240
 Wasserman, P. D., 596, 900
 Watkins, C., 623, 900
 Watson, C., 832, 900
 Watson, J., 13
 Watson, L., 329, 892
 wavelength, 730
 weak AI, 29, 839
 weak method, 22
 weather reports, 692
- Weaver, S., 777
 Weaver, W., 540, 559, 895
 Webber, B. L., 28, 688, 720, 874,
 878, 900
 Weber, J., 27, 520, 737, 877, 880
 Wefald, E. H., 50, 143, 844, 893
 Wegman, M., 328, 889
 weight (in a neural network), 567
 weighted linear function, 127
 Weinstein, S., 559, 889
 Weiss, I., 770, 859
 Weiss, S. M., 560, 901
 Weizenbaum, J., 20, 839, 849, 901
 Weld, D., 260, 364, 390, 411, 814,
 848, 861, 869, 870, 889,
 897, 901
 well-formed formula, T93, 193
 Wellman, M. P., 364, 465, 494, 520,
 843, 868, 874, 901
 Wells, M., 143, 879
 Werbos, P., 595, 889, 901
 Wermuth, N., 465, 881
 West, Col., 267
 Westinghouse, 369, 390
 wff, *see* well-formed formula
 Wheatstone, C., 768, 907
 White, T., 594
 Whitehead, A. N., 8, 16, 291, 292,
 629, 901
 Whiter, A., 389, 898
 Whittaker, W., 778, 896
 Whorf, B., 162, 901
 WHY, 721
 wide content, 821
 Widrow, B., 19, 594, 601, 622, 907
 Wiener, N., 142, 520, 594, 907
 Wigderson, A., 847, 885
 Wilber, B., 894
 Wilcox, B., 145, 892
 Wilensky, R., x, 23, 720, 826, 832,
 907
 Wilkins, D. E., 140, 389, 907
 Wilks, Y., 688, 907
 Williams, R., 496, 595, 623, 889,
 893
 Williamson, M., 411, 870
 Wilson, R. H., 794, 901
 Winker, S., 313, 330, 902
 Winograd, S., 19, 902
 Winograd, T., 19, 23, 330, 827, 898,
 902
 Winston, P. H., 5, 19, 548, 559, 902
 Wittgenstein, L., 158, 179, 232, 685,
 822, 902
- Woehr, J., 466, 902
 Wöhler, R., 834
 Wojciechowski, W., 313, 902
 Wojcik, A., 313, 902
 Wood, D., 116, 881
 Woods, W. A., x, 23, 331, 686, 693,
 720, 902
 word, 652
 Wordnet, 704, 720
 working memory, 314, 645
 world model, in disambiguation, 682
 world state, 56
 worst possible catastrophe, 478
 Wos, L., 293, 294, 313, 330, 902
 Wright, S., 464, 902
 Wrightson, G., 294, 896
 Wu, D., 688, 720, 902
 wumpus world, 153, 206, 216, 227,
 412, 652
 Wundt, W., 12
- X**
- XCON, 316
 Xerox, 692
 xor, *see* exclusive or
- Y**
- Yager, E., 788, 881
 Yakimovsky, Y., 494, 870
 Yang, Q., 389, 903
 Yannakakis, M., 847, 889
 Yap, R., 329, 877
 Yarowsky, D., 688, 903
 yield (in parsing), 658
 Yip, K., 260, 903
 Yoshikawa, T., 811, 903
 Young, D., 29, 895
 Younger, D., 720, 903
- Z**
- Z-3, 14
 Zadeh, L. A., 466, 903
 Zapp, A., 587, 750, 770, 869
 Zermelo, E., 142, 826, 903
 ZFC, 826
 Zhivotovsky, A., 144, 859
 Zilberstein, S., 844, 903
 Zimmermann, H.-J., 466, 903
 zip code, 572, 586
 Zisserman, A., 769, 871, 887, 893
 Zobrist, A., 145, 903
 Zue, V., 26, 903
 Zuse, K., 14, 142, 903
 Zytkow, J. M., 881