27 AI: THE 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.

In Chapter 2, we suggested that it would be helpful to view the AI task as that of designing rational agents—that is, agents whose actions maximize their expected utility given their percept histories. We showed that the design problem depends on the percepts and actions available to the agent, the utility function 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, decision-theoretic agents. Moreover, the components of these designs can have a number of different instantiations—for example, logical or probabilistic reasoning, and atomic, factored, or structured representations of states. The intervening chapters presented the principles by which these components operate.

For all the agent designs and components, there has been tremendous progress both in our scientific understanding and in our technological capabilities. In this chapter, we stand back from the details and ask, *"Will all this progress lead to a general-purpose intelligent agent that can perform well in a wide variety of environments?"* Section 27.1 looks at the components of an intelligent agent to assess what's known and what's missing. Section 27.2 does the same for the overall agent architecture. Section 27.3 asks whether designing rational agents is the right goal in the first place. (The answer is, "Not really, but it's OK for now.") Finally, Section 27.4 examines the consequences of success in our endeavors.

27.1 AGENT COMPONENTS

Chapter 2 presented several agent designs and their components. To focus our discussion here, we will look at the utility-based agent, which we show again in Figure 27.1. When endowed with a learning component (Figure 2.15), this is the most general of our agent designs. Let's see where the state of the art stands for each of the components.

Interaction with the environment through sensors and actuators: For much of the history of AI, this has been a glaring weak point. With a few honorable exceptions, AI systems were built in such a way that humans had to supply the inputs and interpret the outputs,

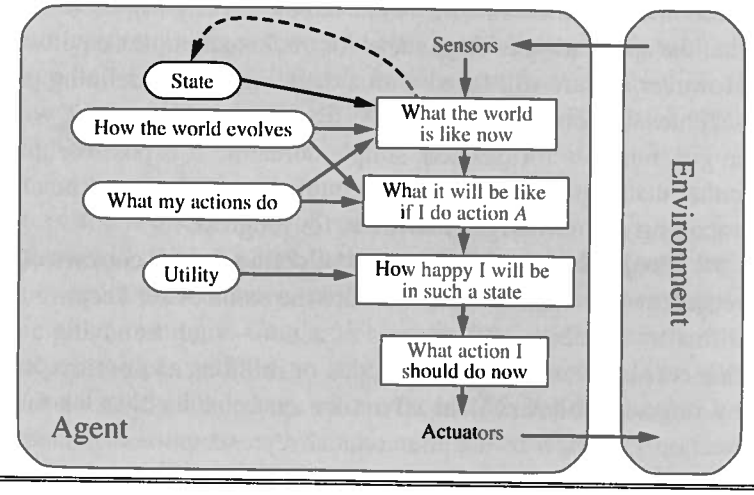

Figure 27.1 A model-based, utility-based agent, as first presented in Figure 2.14.

while robotic systems focused on low-level tasks in which high-level reasoning and planning were largely absent. This was due in part to the great expense and engineering effort required to get real robots to work at all. The situation has changed rapidly in recent years with the availability of ready-made programmable robots. These, in turn, have benefited from small, cheap, high-resolution CCD cameras and compact, reliable motor drives. MEMS (micro-electromechanical systems) technology has supplied miniaturized accelerometers, gyroscopes, and actuators for an artificial flying insect (Floreano *et al.*, 2009). It may also be possible to combine millions of MEMS devices to produce powerful macroscopic actuators.

Thus, we see that AI systems are at the cusp of moving from primarily software-only systems to embedded robotic systems. The state of robotics today is roughly comparable to the state of personal computers in about 1980: at that time researchers and hobbyists could experiment with PCs, but it would take another decade before they became commonplace.

Keeping track of the state of the world: This is one of the core capabilities required for an intelligent agent. It requires both perception and updating of internal representations. Chapter 4 showed how to keep track of atomic state representations; Chapter 7 described how to do it for factored (propositional) state representations; Chapter 12 extended this to first-order logic; and Chapter 15 described **filtering** algorithms for probabilistic reasoning in uncertain environments. Current filtering and perception algorithms can be combined to do a reasonable job of reporting low-level predicates such as "the cup is on the table." Detecting higher-level actions, such as "Dr. Russell is having a cup of tea with Dr. Norvig while discussing plans for next week," is more difficult. Currently it can be done (see Figure 24.25 on page 961) only with the help of annotated examples.

Another problem is that, although the approximate filtering algorithms from Chapter 15 can handle quite large environments, they are still dealing with a factored representation—they have random variables, but do not represent objects and relations explicitly. Section 14.6 explained how probability and first-order logic can be combined to solve this problem, and

Section 14.6.3 showed how we can handle uncertainty about the identity of objects. We expect that the application of these ideas for tracking complex environments will yield huge benefits. However, we are still faced with a daunting task of defining general, reusable representation schemes for complex domains. As discussed in Chapter 12, we don't yet know how to do that in general; only for isolated, simple domains. It is possible that a new focus on probabilistic rather than logical representation coupled with aggressive machine learning (rather than hand-encoding of knowledge) will allow for progress.

Projecting, evaluating, and selecting future courses of action: The basic knowledge-representation requirements here are the same as for keeping track of the world; the primary difficulty is coping with courses of action—such as having a conversation or a cup of tea—that consist eventually of thousands or millions of primitive steps for a real agent. It is only by imposing **hierarchical structure** on behavior that we humans cope at all. We saw in Section 11.2 how to use hierarchical representations to handle problems of this scale; furthermore, work in **hierarchical reinforcement learning** has succeeded in combining some of these ideas with the techniques for decision making under uncertainty described in Chapter 17. As yet, algorithms for the partially observable case (POMDPs) are using the same atomic state representation we used for the search algorithms of Chapter 3. There is clearly a great deal of work to do here, but the technical foundations are largely in place. Section 27.2 discusses the question of how the search for effective long-range plans might be controlled.

Utility as an expression of preferences: In principle, basing rational decisions on the maximization of expected utility is completely general and avoids many of the problems of purely goal-based approaches, such as conflicting goals and uncertain attainment. As yet, however, there has been very little work on constructing *realistic* utility functions—imagine, for example, the complex web of interacting preferences that must be understood by an agent operating as an office assistant for a human being. It has proven very difficult to decompose preferences over complex states in the same way that Bayes nets decompose beliefs over complex states. One reason may be that preferences over states are really *compiled* from preferences over state histories, which are described by **reward functions** (see Chapter 17). Even if the reward function is simple, the corresponding utility function may be very complex. This suggests that we take seriously the task of knowledge engineering for reward functions as a way of conveying to our agents what it is that we want them to do.

Learning: Chapters 18 to 21 described how learning in an agent can be formulated as inductive learning (supervised, unsupervised, or reinforcement-based) of the functions that constitute the various components of the agent. Very powerful logical and statistical techniques have been developed that can cope with quite large problems, reaching or exceeding human capabilities in many tasks—as long as we are dealing with a predefined vocabulary of features and concepts. On the other hand, machine learning has made very little progress on the important problem of constructing new representations at levels of abstraction higher than the input vocabulary. In computer vision, for example, learning complex concepts such as *Classroom* and *Cafeteria* would be made unnecessarily difficult if the agent were forced to work from pixels as the input representation; instead, the agent needs to be able to form intermediate concepts first, such as *Desk* and *Tray*, without explicit human supervision. Similar considerations apply to learning behavior: *HavingACupOfTea* is a very important high-level step in many plans, but how does it get into an action library that initially contains much simpler actions such as *RaiseArm* and *Swallow*? Perhaps this will incorporate some of the ideas of **deep belief networks**—Bayesian networks that have multiple layers of hidden variables, as in the work of Hinton *et al.* (2006), Hawkins and Blakeslee (2004), and Bengio and LeCun (2007).

DEEP BELIEF NETWORKS

The vast majority of machine learning research today assumes a factored representation, learning a function $h : \mathbb{R}^n \rightarrow \mathbb{R}$ for regression and $h : \mathbb{R}^n \rightarrow \{0, 1\}$ for classification. Learning researchers will need to adapt their very successful techniques for factored representations to structured representations, particularly hierarchical representations. The work on inductive logic programming in Chapter 19 is a first step in this direction; the logical next step is to combine these ideas with the probabilistic languages of Section 14.6.

Unless we understand such issues, we are faced with the daunting task of constructing large commonsense knowledge bases by hand, an approach that has not fared well to date. There is great promise in using the Web as a source of natural language text, images, and videos to serve as a comprehensive knowledge base, but so far machine learning algorithms are limited in the amount of organized knowledge they can extract from these sources.

27.2 AGENT ARCHITECTURES

It is natural to ask, "Which of the agent architectures in Chapter 2 should an agent use?" The answer is, "All of them!" We have seen that reflex responses are needed for situations in which time is of the essence, whereas knowledge-based deliberation allows the agent to plan ahead. A complete agent must be able to do both, using a **hybrid architecture**. One important property of hybrid architectures is that the boundaries between different decision components are not fixed. For example, **compilation** continually converts declarative information at the deliberative level into more efficient representations, eventually reaching the reflex level—see Figure 27.2. (This is the purpose of explanation-based learning, as discussed in Chapter 19.) Agent 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. A less studied problem is the *reversal* of this process: when the environment changes, learned reflexes may no longer be appropriate and the agent must return to the deliberative level to produce new behaviors.

HYBRID ARCHITECTURE

Agents also need ways to control their own deliberations. They must be able to cease deliberating when action is demanded, and they must be able to use the time available for deliberation to execute the most profitable computations. For example, a taxi-driving agent that sees an accident ahead must decide in a split second either to brake or to take evasive action. 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

REAL-TIME AI

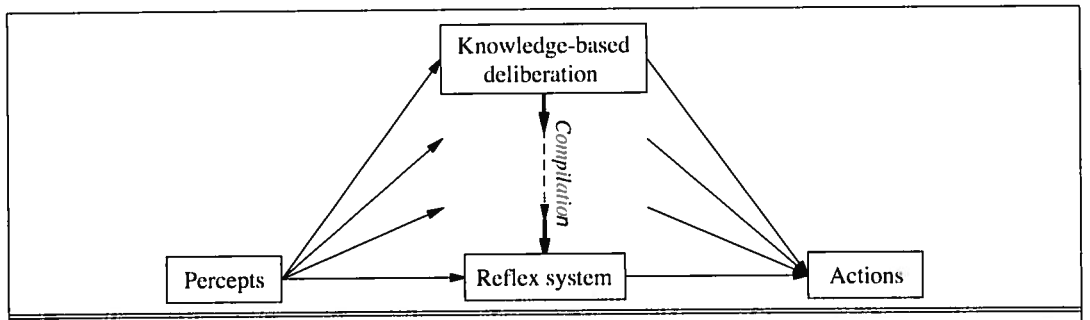

Figure 27.2 Compilation serves to convert deliberative decision making into more efficient, reflexive mechanisms.

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.

ANYTIME ALGORITHM

Clearly, there is a pressing need for *general* methods of controlling deliberation, rather than specific recipes for what to think about in each situation. The first useful idea is to employ **anytime algorithms** (Dean and Boddy, 1988; Horvitz, 1987). 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 assesses whether further computation is worthwhile. (See Section 3.5.4 for a brief description of metalevel decision making.) Example of an anytime algorithms include iterative deepening in game-tree search and MCMC in Bayesian networks.

DECISION-THEORETIC METAREASONING

The second technique for controlling deliberation is **decision-theoretic metareasoning** (Russell and Wefald, 1989, 1991; Horvitz, 1989; Horvitz and Breese, 1996). This method applies the theory of information value (Chapter 16) to the selection of individual 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 to guarantee that the algorithms have the anytime property. Metareasoning is expensive, of course, and compilation methods can be applied so that the overhead is small compared to the costs of the computations being controlled. Metalevel reinforcement learning may provide another way to acquire effective policies for controlling deliberation: in essence, computations that lead to better decisions are reinforced, while those that turn out to have no effect are penalized. This approach avoids the myopia problems of the simple value-of-information calculation.

REFLECTIVE ARCHITECTURE

Metareasoning is one specific example of a **reflective architecture**—that is, an architecture that enables deliberation about the computational entities and actions occurring within the architecture itself. A theoretical foundation for reflective architectures can be built by defining a joint state space composed from the environment state and the computational state of the agent itself. Decision-making and learning algorithms can be designed that operate over this joint state space and thereby serve to implement and improve the agent's computational activities. Eventually, we expect task-specific algorithms such as alpha–beta search and backward chaining to disappear from AI systems, to be replaced by general methods that direct the agent's computations toward the efficient generation of high-quality decisions.

27.3 ARE WE GOING IN THE RIGHT DIRECTION?

The preceding section listed many advances and many opportunities for further progress. But where is this all leading? Dreyfus (1992) gives the analogy of trying to get to the moon by climbing a tree; one can report steady progress, all the way to the top of the tree. In this section, we consider whether AI's current path is more like a tree climb or a rocket trip.

In Chapter 1, we said that our goal was to build agents that *act rationally*. However, we also said that

> . . . achieving perfect rationality—always doing the right thing—is not feasible in complicated environments. The computational demands are just too high. For most of the book, however, we will adopt the working hypothesis that perfect rationality is a good starting point for analysis.

Now it is time to consider again what exactly the goal of AI is. We want to build agents, but with what specification in mind? Here are four possibilities:

PERFECT RATIONALITY

Perfect rationality. 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. We have seen that the calculations necessary to achieve perfect rationality in most environments are too time consuming, so perfect rationality is not a realistic goal.

CALCULATIVE RATIONALITY

Calculative rationality. This is the notion of rationality that we have used implicitly in designing logical and decision-theoretic agents, and most of theoretical AI research has focused on this property. 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, but in most environments, the right answer at the wrong time is of no value. In practice, AI system designers are forced to compromise on decision quality to obtain reasonable overall performance; unfortunately, the theoretical basis of calculative rationality does not provide a well-founded way to make such compromises.

BOUNDED RATIONALITY

Bounded rationality. Herbert Simon (1957) rejected the notion of perfect (or even approximately perfect) rationality and replaced it with bounded rationality, a descriptive theory of decision making by real agents. He wrote,

> The capacity of the human mind for formulating and solving complex problems is very small compared with the size of the problems whose solution is required for objectively rational behavior in the real world—or even for a reasonable approximation to such objective rationality.

He suggested that bounded rationality works primarily by **satisficing**—that is, deliberating only long enough to come up with an answer that is "good enough." Simon won the Nobel Prize in economics for this work and has written about it in depth (Simon, 1982). It appears to be a useful model of human behaviors in many cases. It is not a formal specification for intelligent agents, however, because the definition of "good enough" is not given by the theory. Furthermore, satisficing seems to be just one of a large range of methods used to cope with bounded resources.

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 four possibilities, bounded optimality seems to offer the best hope for a strong theoretical foundation for AI. It has the advantage of being possible to achieve: there is always at least one best program—something that perfect rationality lacks. Bounded optimal agents are actually useful in the real world, whereas calculatively rational agents usually are not, and satisficing agents might or might not be, depending on how ambitious they are.

The traditional approach in AI has been to start with calculative rationality and then make compromises to meet resource constraints. If the problems imposed by the constraints are minor, one would expect the final design to be similar to a BO agent design. But as the resource constraints become more critical—for example, as the environment becomes more complex—one would expect the two designs to diverge. In the theory of bounded optimality, these constraints can be handled in a principled fashion.

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 *et al.*, 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 gigabyte machine made a significant difference to the design of the BO program. One way to make sure this cannot happen is to be slightly more relaxed about the criteria for bounded optimality. By analogy with the notion of asymptotic complexity (Appendix A), we can define **asymptotic bounded optimality** (ABO) as follows (Russell and Subramanian, 1995). Suppose a program P is bounded optimal for a machine M in a class of environments $\mathbf{E}$, where the complexity of environments in $\mathbf{E}$ is unbounded. Then program P' is ABO for M in $\mathbf{E}$ if it can outperform P by running on a machine kM that is k times faster (or larger) than M. 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 will reflect optimizations that are specific to the particular environment. This is only to be expected, and it is similar to the way in which racing cars restricted by engine capacity 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 sum, the concept of bounded optimality is proposed as a formal task for AI 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.

27.4 WHAT IF AI DOES SUCCEED?

In David Lodge's *Small World* (1984), 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 be evoked by asking AI researchers, "*What if you succeed?*"

As Section 26.3 relates, there are ethical issues to consider. Intelligent computers are more powerful than dumb ones, but will that power be used for good or ill? Those who strive to develop AI have a responsibility to see that the impact of their work is a positive one. The scope of the impact will depend on the degree of success of AI. Even modest successes in AI have already changed the ways in which computer science is taught (Stein, 2002) and software development is practiced. AI has made possible new applications such as speech recognition systems, inventory control systems, surveillance systems, robots, and search engines.

We can expect that medium-level successes in AI would affect all kinds of people in their daily lives. So far, computerized communication networks, such as cell phones and the Internet, have had this kind of pervasive effect on society, but AI has not. AI has been at work behind the scenes—for example, in automatically approving or denying credit card transactions for every purchase made on the Web—but has not been visible to the average consumer. We can imagine that truly useful personal assistants for the office or the home would have a large positive impact on people's lives, although they might cause some economic dislocation in the short term. Automated assistants for driving could prevent accidents, saving tens of thousands of lives per year. A technological capability at this level might also be applied to the development of autonomous weapons, which many view as undesirable. Some of the biggest societal problems we face today—such as the harnessing of genomic information for treating disease, the efficient management of energy resources, and the verification of treaties concerning nuclear weapons—are being addressed with the help of AI technologies.

Finally, it seems likely that a large-scale success in AI—the creation of human-level intelligence and beyond—would change the lives of a majority of humankind. The very nature of our work and play would be altered, as would our view of intelligence, consciousness, and the future destiny of the human race. AI systems at this level of capability could threaten human autonomy, freedom, and even survival. For these reasons, we cannot divorce AI research from its ethical consequences (see Section 26.3).

Which way will the future go? Science fiction authors seem to favor dystopian futures over utopian ones, probably because they make for more interesting plots. But so far, AI seems to fit in with other revolutionary technologies (printing, plumbing, air travel, telephony) whose negative repercussions are outweighed by their positive aspects.

In conclusion, we see that AI has made great progress in its short history, but the final sentence of Alan Turing's (1950) essay on *Computing Machinery and Intelligence* is still valid today:

> *We can see only a short distance ahead,*
> *but we can see that much remains to be done.*

A MATHEMATICAL BACKGROUND

A.1 COMPLEXITY ANALYSIS AND O() NOTATION

BENCHMARKING

Computer scientists are often faced with the task of comparing algorithms to see how fast they run or how much memory they require. There are two approaches to this task. The first is **benchmarking**—running the algorithms on a computer and measuring speed in seconds and memory consumption in bytes. 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. The second approach relies on a mathematical **analysis of algorithms**, independently of the particular implementation and input, as discussed below.

ANALYSIS OF ALGORITHMS

A.1.1 Asymptotic analysis

We will consider algorithm analysis through 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) do
      sum ← sum + sequence[i]
  return sum
```

The first step in the analysis is to abstract over the input, in order 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.