The first online planner with execution monitoring was **PLANEX** (Pikes *et al.*, 1972), which worked with the STRIPS planner to control the robot Shakcy. The NASL planner (McDermott, 1978a) treated a planning problem simply as a specification for carrying out a complex action, so that execution and planning were completely unified. **SIPE** (System for Interactive Planning and Execution monitoring) (Wilkins, 1988, 1990) was the first planner to deal systematically with the problem of replanning. It has been used in demonstration projects in several domains including planning operations on the flight deck of an aircraft carrier, job-shop scheduling for an Australian beer factory, and planning the construction of multistory buildings (Kartam and Levitt, 1990).

In the mid-1980s, pessimism about the slow run times of planning systems led to the proposal of reflex agents called **reactive planning** systems (Brooks, 1986, Agre and Chapman, 1987). **PENGI** (Agre and Chapman, 1987) could play a (fully observable) video game by using Boolean circuits combined with a "visual" representation of current goals and the agent's internal state. "Universal plans" (Schoppers, 1987, 1989) were developed as a lookup-table method for reactive planning, but turned out to be a rediscovery of the idea of policies that had long been used in Markov decision processes (see Chapter 17). A universal plan (or a policy) contains a mapping from any state to the action that should be taken in that state. Koenig (2001) surveys online planning techniques, under the name **Agent-Centered Search**.

Multiagent planning has leaped in popularity in recent years, although it does have a long history. Konolige (1982) formalizes multiagent planning in first-order logic, while Pednault (1986) gives a **STIPS-style** description. The notion of joint intention, which is essential if agents are to execute a joint plan, comes from work on communicative acts (Cohen and Levesque, 1990; Cohen *et al.*, 1990). Rountier and Brafman (2001) show how to adapt partial-order planning to a multiactor setting. Brafman and Domshlak (2008) devise a **multiactor planning algorithm** whose complexity grows only linearly with the number of actors, provided that the degree of coupling (measured partly by the **tree width** of the graph of interactions among agents) is bounded. Petrik and Zilberstein (2009) show that an approach based on bilinear programming outperforms the cover-set approach we outlined in the chapter.

We have barely skimmed the surface of work on negotiation in multiagent planning. Durfee and Lesser (1989) discuss how tasks can be shared out among agents by negotiation. Kraus *et al.* (1991) describe a system for playing Diplomacy, a board game requiring negotiation, coalition formation, and dishonesty. Stone (2000) shows how agents can cooperate as teammates in the competitive, dynamic, partially observable environment of robotic soccer. In a later article, Stone (2003) analyzes two competitive multiagent environments—RoboCup and TAC, the auction-based Trading Agents Competition—and finds that the computational intractability of our current theoretically well-founded approaches has led to many multiagent systems being designed by **ad hoc** methods.

In his highly influential **Society of Mind** theory, Marvin Minsky (1986, 2007) proposes that human minds are constructed from an ensemble of agents. Livnat and Pippenger (2006) prove that, for the problem of optimal path-finding, and given a limitation on the total amount of compiling resources, the best architecture for an agent is an ensemble of subagents, each of which tries to optimize its own objective, and all of which are in conflict with one another.
The boid model on page 429 is due to Reynolds (1987), who won an Academy Award for its application to swarms of penguins in *Batman Returns*. The NERD game and the methods for learning strategies are described by Bryant and Miikkulainen (2007).

Recent book on multiagent systems include those by Weiss (2000a), Young (2004), Vlassis (2008), and Shoham and Leyton-Brown (2009). There is an annual conference on autonomous agents and multiagent systems (AAMAS).

**EXERCISES**

11.1 The goals we have considered so far all ask the planner to make the world satisfy the goal at just one time step. Not all goals can be expressed this way: you do not achieve the goal of suspending a chandelier above the ground by throwing it in the air. More seriously, you wouldn't want your spacecraft life-support system to supply oxygen one day but not the next. A *maintenance goal* is achieved when the agent's plan causes a condition to hold continuously from a given state onward. Describe how to extend the formalism of this chapter to support maintenance goals.

11.2 You have a number of trucks with which to deliver a set of packages. Each package starts at some location on a grid map, and has a destination somewhere else. Each truck is directly controlled by moving forward and turning. Construct a hierarchy of high-level actions for this problem. What knowledge about the solution does your hierarchy encode?

11.3 Suppose that a high-level action has exactly one implementation as a sequence of primitive actions. Give an algorithm for computing its preconditions and effects, given the complete refinement hierarchy and schemas for the primitive actions.

11.4 Suppose that the optimistic reachable set of a high-level plan is a superset of the goal set; can anything be concluded about whether the plan achieves the goal? What if the pessimistic reachable set doesn't intersect the goal set? Explain.

11.5 Write an algorithm that takes an initial state (specified by a set of propositional literals) and a sequence of HLAs (each defined by preconditions and angelic specifications of optimistic and pessimistic reachable sets) and computes optimistic and pessimistic descriptions of the reachable set of the sequence.

11.6 In Figure 11.2 we showed how to describe actions in a scheduling problem by using separate fields for DURATION, USE, and CONSUME. Now suppose we wanted to combine scheduling with deterministic planning, which requires deterministic and conditional effects. Consider each of the three fields and explain if they should remain separate fields, or if they should become effects of the action. Give an example for each of the three.

11.7 Some of the operations in standard programming languages can be modeled as actions that change the state of the world. For example, the assignment operation changes the contents of a memory location, and the punt operation changes the state of the output stream. A program consisting of these operations can also be considered as a plan, whose goal is given