C. Green, "Theorem-Proving by Resolution as a Basis for Question-Answering Systems," in B. Meltzer and D. Michie, eds., *Machine Intelligence*, Vol. 4, Halsted, Wiley, New York, 1969, pp. 183–205.

K. J. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task*, Academic Press, Cambridge, Mass., 1989a.

K. J. Hammond, "Opportunistic Memory," *Proceedings of the 11th IJCAI*, Detroit, Mich., Morgan-Kaufmann, San Mateo, Calif., 1989, pp. 504–510.

B. Hayes-Roth and co-workers, "Modelling Planning as an Incremental, Opportunistic Process," *Proceedings of the Sixth IJCAI*, Tokyo, Japan, Morgan-Kaufmann, San Mateo, Calif., 1979, pp. 375–383.

G. G. Hendrix, "Modelling Simultaneous Actions and Continuous Processes," *Artif. Intell.* **4**, 145–180 (1973).

G. Houghton and Stephen Isard, "Why to Speak, What to Say and How to Say It: Modelling Language Production in Discourse," in P. Morris, ed., *Modelling Cognition*, John Wiley & Sons, Inc., Chichester, 1987.

S. A. Hutchinson and A. C. Kak, "Spar: A Planner that Satisfies Operational and Geometric Goals in Uncertain Environments," *AI Magazine* 30–61 (Spring 1990).

D. Joslin and J. Roach, "A Theoretical Analysis of Conjunctive-Goal Problems," *Artif. Intell.* **41**, 97–106 (Nov. 1989).

D. Kibler and B. Porter, "Episodic Learning," *Proceedings of the Third National Conference on Artificial Intelligence*, Washington, D.C., AAAI, Menlo Park, Calif., 1983, pp. 191–196.

J. McCarthy and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in B. Meltzer and D. Michie, eds., *Machine Intelligence*, Vol. 4, Halsted, John Wiley & Sons, Inc., New York, 1969, pp. 463–501.

D. A. McDermott, "A Temporal Logic for Reasoning about Processes and Plans," *Cog. Sci.* **6**, 101–155 (1982).

A. Newell and H. A. Simon, "GPS, a Program that Simulates Human Thought," in E. A. Feigenbaum and J. Feldman, eds., *Computers and Thought*, McGraw-Hill, New York, 1963, pp. 279–293.

R. Power, "The Organization of Purposeful Dialogues," *Linguistics* **17**, 107–152 (1979).

E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artif. Intell.* **5**, 115–135 (1974).

E. D. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier/North-Holland, New York, 1977.

R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals, and Understanding*, Erlbaum, Hillsdale, N.J., 1977.

L. Siklossy and J. Dreussi, "An Efficient Robot Planner Which Generates Its Own Procedures," *Proceedings of the Third IJCAI*, Stanford, Calif., Morgan-Kaufmann, San Mateo, Calif., 1973, pp. 423–430.

M. Stefik, "Planning with Constraints (MOLGEN: Part I)," *Artif. Intell.* **16**, 111–139 (May 1981).

G. J. Sussman, *A Computer Model of Skill Acquisition*, American Elsevier, New York, 1975.

A. Tate, *Project Planning Using a Hierarchic Non-Linear Planner*, Report No. 25, Artificial Intelligence Dept., University of Edinburgh, Aug. 1976.

A. Tate, "Generating Project Networks," *Proceedings of the Fifth IJCAI*, Cambridge, Mass., Morgan-Kaufmann, San Mateo, Calif., 1977, pp. 888–893.

A. Tate and A. M. Whiter, "Planning with Multiple Resource Constraints and an Application to a Naval Planning Problem," *Proceedings of the First Conference on AI Applications*, Denver, Colo., 1984, pp. 410–416.

S. A. Vere, "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-5**, 246–267 (May 1983).

S. A. Vere, "Splicing Plans to Achieve Misordered Goals," *Proceedings of the Ninth IJCAI*, Los Angeles, Calif., Morgan-Kaufmann, San Mateo, Calif., 1985, pp. 1016–1021.

S. A. Vere and T. W. Bickmore, "A Basic Agent," *Comp. Intell.*, 41–60 (May 1990).

D. H. D. Warren, "Generating Conditional Plans and Programs," *Proceedings of the AISB Conference*, University of Edinburgh, 1976, pp. 344–354.

R. B. Wesson, "Planning in the World of the Air Traffic Controller," *Proceedings of the Fifth IJCAI*, Cambridge, Mass., Morgan-Kaufmann, San Mateo, Calif., 1977, pp. 473–479.

R. Wilensky, *Planning and Understanding*, Addison-Wesley, Reading, Mass., 1983.

D. E. Wilkins, "Domain-Independent Planning: Representation and Plan Generation," *Artif. Intell.* **22**, 269–301 (April 1984).

D. E. Wilkins, *Recovering from Execution Errors in SIPE*, Technical Note 346, AI Center, SRI International, Menlo Park, Calif., Jan. 1985.

General References

L. Daniel, "Planning and Operations Research," in T. O'Shea and M. Eisenstadt, eds. *Artificial Intelligence: Tools, Techniques, and Applications*, Harper & Row, New York, 1984, pp. 423–452. Reviews STRIPS, NOAH, and NONLIN in detail, and discusses their limitations.

M. Drummond and A. Tate, *AI Planning: a Tutorial and Review*, Technical Report AIAI-TR-30, AI Applications Institute, Edinburgh, Scotland, Jan. 1989. A recent survey that compares 15 major planners.

M. Georgeff, "Planning," in *Annual Review of Computer Science*, Vol. 2, 1987, pp. 359–400. A survey of planning from a theoretical perspective.

E. D. Sacerdoti, "Problem Solving Tactics," *Proceedings of the Sixth IJCAI*, Tokyo, Japan, Morgan-Kaufmann, San Mateo, Calif., 1979, pp. 1077–1085. Discusses methods for improving the efficiency of planners.

W. Swartout, and co-workers, "Summary Report," *Proceedings of Knowledge-Based Planning Workshop*, DARPA, Austin, Tex., Dec. 1987, A1-A23. Summary of a workshop to identify and explore new directions for planning research.

D. E. Wilkins, *Practical Planning*, Morgan-Kaufmann, San Mateo, Calif., 1988. An in-depth exposition of the SIPE planner.

S. VERE
Lockheed AI Center

PLANNING, REACTIVE

Planning, in AI, refers to a body of techniques used to automate the process of selecting and carrying out actions to bring an environment to some desired state. In *Classical Planning* [eg, STRIPS (Fikes and Nilsson, 1971), NONLIN (Tate, 1975), DEVISER (Vere, 1983), TWEAK (Chapman, 1987), an application domain is encoded in terms of a set of primitive actions and their preconditions and effects characterized as state predicates. A planning problem consists of this domain description plus an initial

Reprinted from Encyclopedia of Artificial Intelligence, Second Edition,
Copyright © 1992 by John Wiley & Sons, Inc.

and final (or goal) state. Planning in this framework consists of searching through the space of action orderings to find one which takes the initial state to the goal state. This set of ordered actions is called the plan. The 'blocks world' is the archetypical domain for classical planning: a world consisting of a number of blocks which the agent (usually thought of as a robot) can stack upon each other. The agent is the only entity active in the environment, and the configurations of the blocks are precisely known at all times.

Many real-world application domains, particularly robotic applications, do not have the static characteristics of blocks world. Schoppers (1988) describes an example domain that contrasts strongly with the blocks world: his 'baby world.' This domain is similar to the blocks world, except it is inhabited by a "mischievous baby who will flatten block towers, snatch blocks out of the robot's hand, and even throw blocks at the robot." The crucial new ingredients in this problem domain are that the agent (1) cannot be certain of the effects of its actions, (2) cannot make the assumption that the world remains static and unchanging except when it carries out an action, and (3) cannot assume that it knows everything about the world. This is exactly the problem faced by a robotic machine operating in the "real-world," the same uncertain (see 1, 3) and dynamic (see 2) environment that humans inhabit in everyday life.

Classical planning becomes too brittle in these application domains (Kaelbling, 1986; Chapman, 1987; Brooks, 1986; Agre and Chapman, 1987) for two main reasons: because the world can change while planning is in progress, partial plans may be, thus, rendered useless; and, because of the uncertain effects of actions, "correct" plans actually may fail to achieve their goal. Chapman (1987) summarizes the state-of-the-art in classical planning. He shows that classical planning, in the general case, is undecidable, and even in its simpler forms can be computationally intractable. Furthermore, Arbib (1981) (see also SCHEMA THEORY) has maintained for some time that behavior should not be produced as the result of symbolic reasoning from axioms, but rather as the result of cooperation and competition between concurrent, active agents called *schema instances*. His schema theory is a bridge between Cognitive Science, Brain Theory, and Artificial Intelligence, and implementations of it have been made by Lyons (1989), Ankin (1989), and Draper and coworkers (1989), among others.

Work in building planning systems that can operate well in application domains such as Schoppers' "baby-world" has shown up recently in a number of workshops and conferences, eg, the 1986 Timberline Lodge workshop, the 1988 Rochester Planning Workshop, the 1990 and 1991 AAAI Spring Symposia, and the 1990 Workshop on Innovative Approaches to Planning, Scheduling, and Control. This work has been grouped together under the name *Reactive Planning*. It has been noted that reactive planning is dangerously near a contradiction in terms, because reaction is usually considered acting *without* planning. The term reactive planning is used here to refer to techniques developed to cope with the selection and execution of actions to achieve objectives in an uncertain and dynamic environment. In one sense, the goal of this field is to combine the advantages of a reactive system, robustness, and time–critical response, with the advantages of a planning system, look–ahead and global optimality. However, it can be argued that such integration may lead to a redefinition of the meaning of both planning and reaction.

In the next section, the characteristics of the reactive planning domain are described via a running example of an automated tour guide. Subsequently, a selection of the key reactive planning techniques and a description of how they can be used to address the tour guide example is presented. Finally, there is a summary of the state of the field, and a discussion of what problems remain open.

CHARACTERISTICS OF THE REACTIVE PLANNING DOMAIN

This section introduces the characteristics of the reactive planning problem domain via a running example: an automated New York City tour guide. Basically, the duty of a tour guide is to take the tour to a set of landmarks, explain each one, and answer any questions. However, in practice, there is more than this to being a good tour guide. A tour guide must fulfill one of the most stringent aspects of a reactive planning problem—he/she must be prepared to act and interact *at any time*.

Reactivity

Continual vigilance is essential to a reactive system. For example, a tour guide is continuously on duty once the tour starts. The audience can ask questions about almost anything at anytime, and the guide should be prepared to respond. There are also a host of other tasks, such as making sure everyone is back on the bus before going to the next location, working around the changing weather, considering the preferences of the audience, guarding the general safety of the tourists, etc. The classical planning approach involves the *a priori* complete or incremental generation of a plan, which is then sent to a plan executer to be carried out. A plan executer can do nothing without a plan to execute. A reactive system needs to be able to respond to the environment without necessary recourse to plans.

Timely Activity

Coordination with externally imposed time constraints is unavoidable in a reactive domain. For example, the tour guide must deal with the fact that the timeliness of actions is important. Specific opening hours of parks, museums, zoos, etc, must be considered if the tour is to remain on time, and in the rescheduling of events if there are any disturbances to the tour. The tour can only stop for a limited time at each location, and the next location needs to be given to the bus driver in advance. Time constraints are involved in reactive planning in three ways: (1) the agent must carry out actions in a timely fashion, (2) the agent must consider the effects of time in choosing which actions are appropriate, and (3) the strategy for choosing the next action must also abide by time constraints. As an example of this third constraint, note that

the strategy for deciding where to go next can only occupy some fraction of the time allocated to visit a location.

Uncertainty

Dealing with uncertainty is a fact of life for a reactive system. A tour guide has *a priori* information at his or her disposal. Nonetheless, making a tour plan in the sense of the classical planning approach, *a priori* production of set of actions and some ordering information, is out of the question because the state of the environment and the actions of other entities during plan execution time (ie, the duration of the tour) are uncertain. Weather forecasts tend to be imperfect, and the preferences and humors of tour group members are notoriously fickle, not to mention the traffic patterns of New York City! For example, a good tour guide will visit outdoor locations, such as Central Park, when it is dry. Indoor activities will be scheduled for times when it rains. In New York City, this will occasionally demand hasty improvisations. Improvising the completion of a tour will require that the guide think about timely coordination with entities such as museums, parks, and restaurants, as well as maintain an on-going interaction with the audience. In such a case, it is better to keep the audience dry and interested than to spend time deriving an optimal perturbed tour!

Improvisation and Interaction

Effective behavior in a reactive planning domain is the result of the close interaction of agent and environment. For example, special events, such as parades or street theater, can be both a nuisance (for routes might have to be changed) and an opportunity (to expose tourists to those events) for a tour guide. A good tour guide should take advantage of unexpected opportunities when possible. This may mean adding new locations to the tour or selecting different paths between locations. With a human tour guide, the description and emphasis of the talks delivered at each location would also change. As another example, a good guide asks the tourists for their preferences before and during the tour, and may adapt the tour accordingly. Thus, a good tour is not a sequence of actions carried out by the guide on a passive audience, but rather a continual interaction between guide and audience. It is less successful to carry out a predetermined tour, hear the complaints afterwards, and try to convince the complainers to take another tour which will incorporate their wishes (ie, a "recovery from error" paradigm).

Actually building an "automated" tour guide would require the application of many other areas of AI in addition to reactive planning: speech-understanding, computer vision, tutorial-systems, cooperative problem-solving, user-modeling, path-planning, redundant kinematics, etc. In this example, the reactive planning component concerns the selection and execution of actions appropriate to the situation.

TECHNIQUES FOR REACTIVE PLANNING

This section discusses the techniques that have been developed for dealing with reactive planning problems. The first work in this area dates from about 1986 (though the inspiration could indeed be traced back to the STRIPS Triangle Table work from the 1970s); and the field is still far from solved. The techniques developed in this field can be effectively classified into three groups, based on their objectives:

1. *Architectures for Reactive Machines.* A reactive machine is a system in which the choice of the next action is based on hardwired response to sensory input and built-in goals. It differs from a classical plan executer in that it is always ready to carry out actions; it is not waiting for a plan to be loaded. Work in this area is in developing architectures, representations, and algorithms that can be used in building reactive machines that exhibit intelligent and robust behavior. This was one of the first areas of the field to be explored. It produced the unexpected results that it is possible to build a reactive agent that produces behavior that an observer would classify as intelligent, despite the lack of classical planning and reasoning in the agent.

2. *Design of Reactive Machines.* One of the first criticisms of reactive machines was that their robustness and intelligence really depended on the skills of the programmer who built them. This second area of reactive planning responded by designing tools which automatically build reactive systems to fulfill a set of desired criteria. The machine generation stage is assumed to be off-line; a specification of the machine and its environment is written and then input to the generation tool, which produces a detailed description of the reactive machine.

3. *Planning for Reaction.* Another major criticism of reactive machines is that they were "myopic"—unable to rise above local responses to the environment. This third area of reactive planning responded by trying to integrate the concept of planning-ahead with that of reaction to the environment.

Architectures for Reactive Machines

Chapman finished his TWEAK (Chapman, 1987) paper by suggesting that improvisation might be a better paradigm than planning. Agre and Chapman (1987) carried out the first step on the road to building systems that can exhibit intelligent behavior in uncertain and dynamic domains. They argued that "before and beneath any activity of plan following, life is a continual improvisation, a matter of deciding what to do *now* based on how the world is *now*." They built a program, *Pengi,* based on these concepts. Pengi played a video arcade game called Pengo. In a typical Pengo game, the computer appeared to hunt down targets, build traps, escape ambushes, take advantage of opportunities, and act in a timely fashion. However, the intelligent behavior of Pengi was the result of the interaction of relatively simple opportunistic strategies with a complex, structured environment. Two key ideas developed for Pengi were the concept of *routines* and the concept of *indexical-functional* representation.

Routines. A routine is a pattern of interaction between agent and environment. A routine need not be explicitly represented by the agent. It can simply be a property of

the regularity in the interaction between the environment and the agent. For example, tour groups exhibit certain behavioral regularity: they may listen to the guide, they ask questions, they wander off, etc. The tour guide will typically explain some landmark, accepting and responding to questions as they arise. This highly interactive pattern is an example of a routine. Internally, the tour guide may simply have rules that say "answer any question asked" and "describe current landmark". The routine is created by the interleaved effects of these rules driven off the behavior of the environment (tour group). Recognizing and exploiting regularity in the environment allows for the construction of simpler and more robust agents that can effectively cope with uncertainty.

Indexical-Functional Representation. In an indexical-functional representation [called *deictic representation* in the more systematic exposition in Agre (in press)], properties of the immediate environment are only represented in terms of the impact they have on the objectives of the agent. For instance, a tour guide may have to deal with many people and many tours in any day. A classical planning approach would involve uniquely naming each individual encountered, eg, PERSON-23. This introduces combinatorics, since the space of persons must now be searched for appropriate instances. A Pengi-like tour guide would not uniquely name every person it encountered, but would only represent people in terms of their impact on its objectives, eg, person-now-asking-question, or person-in-danger-of-being-lost. The perceptual system directly matches the environment with such indexical-functional entities, and no search is required. Pengi used a similar mechanism to measure directly from the environment clues as to which actions to take next; these are called indexical-functional aspects. A Pengi tour guide might have such aspects as tour-becoming-bored (and hence should be livened up), or street-theater-in-view (consider diverting to look at it), etc.

Behaviors and Subsumption. Brooks (1986) noted that the standard view of intelligent robot-control architectures as pipe-lined collections of functional modules caused problems with robustness, buildability, and testability. He suggested a novel architecture, called the subsumption architecture, that emphasized building intelligent control from layers of task–achieving behaviors (see Figure 1) much in the spirit of Braitenberg (1984). He built a number of robot systems (Brooks 1986, 1987, 1989) that exhibited behavior similar to Pengi: they appeared to act in a robust intelligent way in complex environments. And like Pengi, they consisted internally of simple opportunistic control rules.

Brooks built his systems as networks of augmented fine state machine modules. Each module can accept input, has internal state, and can produce output. In addition, the output of a module can be inhibited or the input suppressed and replaced. (Brooks associates a time interval with the inhibition and suppression, which will be omitted in the simple examples here). For example, a particularly simple tour guide could be described by two connected modules: Local-Wander, which produces random heading signals, connected to a Move which when given a heading will move in that direction. A subsumption architecture is a hierarchy of such behaviors. High-level behaviors subsume the behaviors of lower levels but add in some extra behavior where appropriate. If we consider the "wander-move" behavior as a level zero subsumption architecture, we can build a level one architecture that "directs" the lower level when appropriate, but otherwise remains silent (see Fig. 2).

Level 1 consists of a counter (Up Counter) that indexes through the list of landmarks, suppresses the local wander behavior to force the guide to each location, and then triggers a speech at that location. While the guide is at a location, the local wander behavior becomes unsuppressed and the guide wanders around the landmark. This is a more useful tour guide, but it is still rather inflexible. We can add a third level that allows for reorganization of the schedule should the weather turn nasty, eg, Figure 3. If bad weather is seen by the Weather module, it inhibits the counter output and replaces it with the index to a landmark that is known to be indoors.

Goals and Beliefs. Nilsson (1988) has put forward an action network structure for teleoactive agents. These agents take changing environmental and goal conditions into account before acting. A strong motivation for this work was the dissatisfaction with conventional action control methods, where higher level units surrender control when they activate a lower-level unit and have to suspend until control is explicitly returned. This scheme can cause low level units to continue to operate in situations for which they are no longer relevant, since changes in situation go unnoticed by the suspended higher levels. To overcome this, Nilsson proposed a new scheme for organizing behavior based on networks of combinatorial circuits in

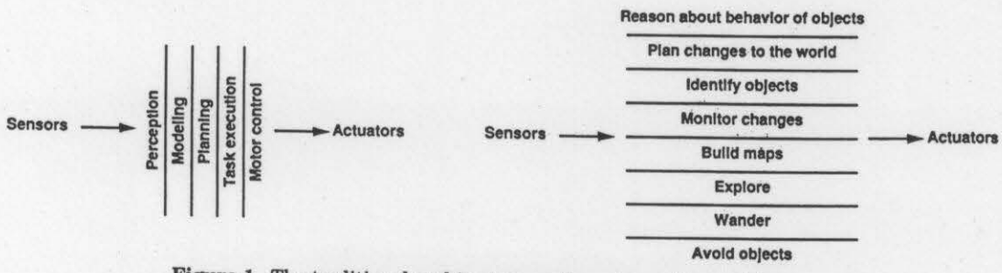

Figure 1. The traditional architecture vs the subsumption architecture.

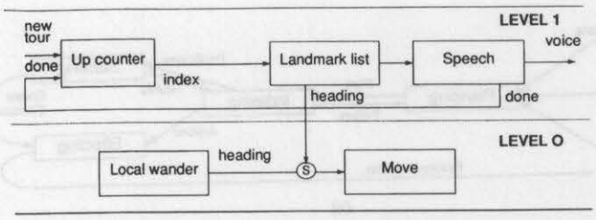

Figure 2. Levels 0 and 1 of a Subsumption tour guide.

which higher level routines enable lower level ones but do *not* surrender control. In addition, he developed a language, ACTNET, that allows a user to specify these control networks as networks of combinatorial circuits. The goals and beliefs of an agent are explicitly represented in this network. The planning component of a network is 'built-in' in the sense that goals are hard-wired to fire particular actions.

The basic unit in Nilsson's network is an action unit (see Fig. 4). An action unit is a logical gate that decides whether to fire a particular action based on the goal request G, the absence of goal establishment (the purpose P) and the preconditions P_i of the action. An action unit continually monitors its input, and fires the action as soon as it is applicable and executable. Goals, as well as preconditions, can be dynamic in nature ("talk so loud that all the tourists can hear it" is dependent on the background noise level). Networks are constructed by tying together the signals representing the goals, preconditions, etc, in an appropriate fashion. Higher level networks can enable lower level ones by turning on their purpose input.

As an example of network, consider the case of a tour guide trying to show the Empire State Building (ESB) to the group (see Fig. 5). This goal fires two primitive goals, ie, at_ESB and present_ESB_speech via action-unit Show_ESB. These establish the goals for the units move_to_ESB and present_ESB_speech. Since present_ESB_speech has as its precondition at_ESB, it has to wait until the other action unit move_to_ESB has established that precondition.

In later work Nilsson (in press) generalizes the notion of action networks to structures that reactively execute sequences of goal-seeking actions, and develops a language for programming such structures. Whereas action units achieve their goals single-handedly, teleo-reactive

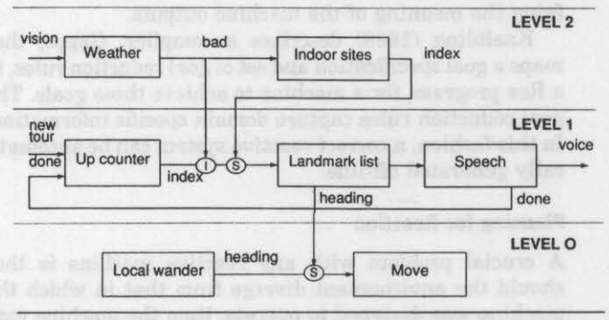

Figure 3. Levels 0, 1, and 2 of a Subsumption tour guide.

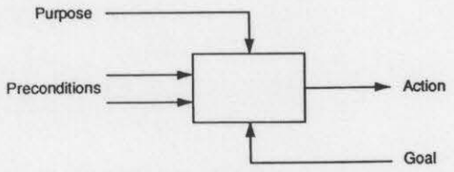

Figure 4. Nilsson's Action Unit.

structures eventually achieve their effects by conditionally enlisting other teleo-reactive structures or actions to achieve necessary subgoals.

Monitoring Change. In dynamic worlds, change proceeds regardless of the agent's state. Therefore, keeping track of the state of affairs is a nontrivial task in such domains. Hendler's DR [Hendler (1989), Sanborn and Hendler (1988)] is an approach to managing observations and actions in dynamic domains. In DR, monitors are employed to keep track of the environmental situation. A monitor is an independent information gathering component of the overall reasoning system that maintains the world model. The duty of a monitor is to report significant events.

Monitoring is coupled to reaction via constraints reporting. Every action has a set of constraints on its execution state. Constraints are divided into inhibitory constraints that prevent an applicable action from executing, and enabling constraints that allow an applicable action to execute. Action selection is based on the active constraints. If an enabling constraint is found on an action, then that action is performed. If an inhibitory constraint is found on an action, then the action is suspended and another applicable action is pursued until the constraint is removed.

Design of Reactive Machines

Reactive machines produce intelligent, but improvised, behavior, and were a major step forward in addressing the problem of acting in uncertain and dynamic environments. They respond quickly and robustly, and can be surprisingly intelligent given their lack of internal models. However, much of the power of a Brooks-style

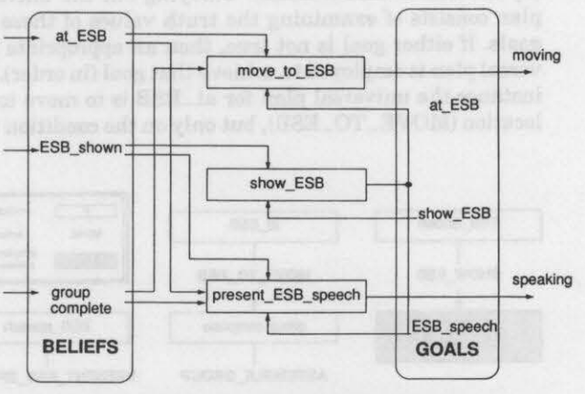

Figure 5. Part of a Tour Guide Action Network.

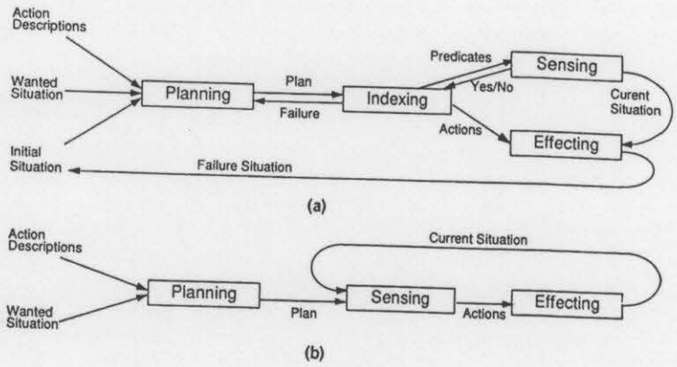

Figure 6. *(a)* The Classical vs *(b)* the Universal Plan approach.

robot, for example, comes from the skill with which the behaviors have been designed and composed together in the subsumption architecture. A major portion of work in reactive planning tries to develop techniques to design formally correct machines, and provide tools for automatically generating a reactive machine to suit a set of design criteria.

Universal Plans: The Autogeneration of Reactive Machines. The Universal Plan (Schoppers, 1987) is an approach to building reactive machines as highly conditional plans. It is a plan that can achieve a goal given any possible initial state of the environment. In this approach, actions are selected through the classifications of the actual situation encountered at execution time, as opposed to the classical approach of selecting actions based on a single initial situation at planning time. In a universal plan, the failure of an action to achieve its predicted effects does not require replanning, only the selection of a new initial point from which to execute the plan. (See Fig. 6.) Therefore, the task of a planner has changed from generating a sequence of actions, to anticipating all possible situations and the concomitant appropriate responses. At execution time, a sensing module recognizes the actual situation and selects the appropriate action to perform.

For example, a (simplified) universal plan for a tour guide showing the Empire State Building (ESB) is shown in Figure 7. The goal, to have the landmark shown to the tour, initiates the two subgoals at_ESB and ESB_speech (by the action SHOW_ESB). Carrying out the universal plan consists of examining the truth values of these two goals. If either goal is not true, then an appropriate universal plan is employed to achieve that goal (in order). For instance the universal plan for at_ESB is to move to the location (MOVE_TO_ESB), but only on the condition that

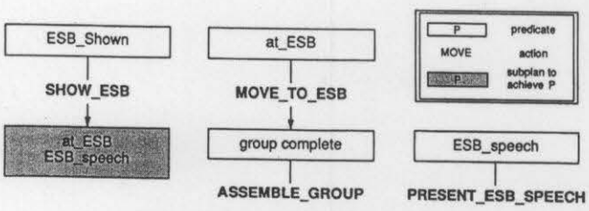

Figure 7. A Universal Tour Guide.

the group is already complete. If this condition does not hold, then the group must first be assembled. Notice that failure of the action MOVE_TO_ESB, for any reason, results in the predicate at_ESB remaining false. This allows the action to be attempted again without replanning. Once the group is at the location, the guide can then pursue the other goal of actually presenting the history of the building.

Situated Automata: Formally Correct Reactive Machines. The *situated automata* model of Rosenschien and Kaelbling (1986, 1987, 1988) has influenced many workers in the field of reactive planning. This model addresses the issues of real-time performance and provably correct behavior. A program in this model is a finite machine with internal state that transforms inputs to outputs within a small time period. It therefore provides a good base on which to build reactive systems. A Lisp-like language, called Rex, was developed to specify such machines. The output of a Rex program is a description of a digital circuit for a machine that meets the Rex specification.

A key characteristic of the situated automata model is the correctness with which the machine's potential behavior is addressed by considering the *knowledge* that the machine contains, where knowledge is analyzed in terms of the relationship between a machine and its environment. A machine is defined to know a proposition ϕ in machine state s, if in all states s, ϕ is satisfied. Given a Rex program, a background theory describing facts about the environment, and a description of the epistemic meaning of the machine inputs, the situated automata model provides a methodology for attacking the problem of verifying the meaning of the machine outputs.

Kaelbling (1988) describes a compiler, Gapps, that maps a goal specification and set of goal reduction rules, to a Rex program, for a machine to achieve these goals. The goal reduction rules capture domain specific information. In this fashion, a correct reactive system can be automatically generated off-line.

Planning for Reaction

A crucial problem with any reactive machine is that should the environment diverge from that in which the machine was designed to operate, then the machine may produce inappropriate behavior. The best a reactive ma-

chine can do in the face of such a change is to degrade gracefully; that is, its responses may not be as efficient or appropriate in the new environment, however, these responses will not result in self-destruction. To address this problem completely, it is necessary to integrate the concept of the reactive machine with the concept of *a priori* planning. Note that these two concepts have some complementary characteristics: a reactive machine is not able to look ahead and plan, while an on-line, time-constrained planner cannot react quickly enough to deal with unexpected hazards. In this section we look at some approaches to integrating a planning component with a reactive machine.

Time Constraints on Planning. Interaction with a dynamic world forces time-constraints onto both planning and reaction. Reactive systems can usually cope with these constraints, however, it is more difficult for planners. Dean and Boddy (1987) introduced the class of ANYTIME algorithms to deal with time-dependent planning. An algorithm has the ANYTIME property if it can handle preemption, will return a sensible answer at any time when terminated, and, most importantly, will return answers which improve monotonically over time. The design of ANYTIME algorithms plays a key role in many approaches to reactive planning.

Integrating Planning and Reaction in a Hierarchy. Hendler (1990) notes the conflicting needs of reactors and planners: the reactors need input that closely matches the external world and need to respond on a very short timescale, the planners need to abstract away from the details of the external world and may need to work for some time on a problem. The solution suggested is a hierarchy as shown in Figure 8, based on the DR model discussed earlier. A scheduler chooses planner or reaction agents on a time permitting basis.

To investigate this scheduling approach, Spector and Hendler (1988) propose a model in which planning and reaction interact across five different levels: sensory/motor, spatial, temporal, causal, and conventional (ie, general world knowledge). This multilevel reasoning is realized in a parallel, blackboard-based planner called APE (Abstraction Partitioned Evaluator). As an example of

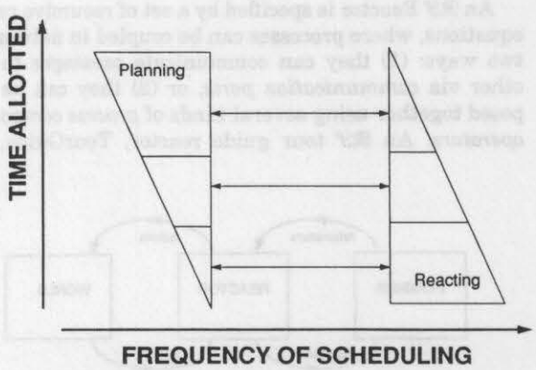

Figure 8. Hendler's Planning-Reaction hierarchy.

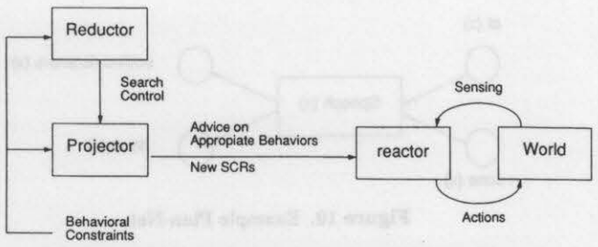

Figure 9. ERE Reductor-Projector-Reactor architecture.

how APE would reason, suppose that the tour guide has a plan for conducting a tour that includes visiting the Empire State Building. Upon arrival however, the tour guide sees an excessive line for entrance to the Empire State Building. This would first be reported as a problem at the spatial level. Replanning at that level is then immediately attempted (eg, find another, shorter line), but if that fails, then the problem is passed to the next highest level for solution and so on. During higher level reasoning, the tour guide would be using more of its resources on those levels, and would be less able to react to change in the environment, thus the propagation of information to higher levels only occurs when time (and the environment) permits.

Planning to Guide Reaction. Bresina and Drummond (1990) introduce an alternative approach to systems that produce intelligent action under time constraints. Their domain of application is photoelectric telescope scheduling problems. They suggest an architecture, called ERE, that combines the ability to react to the current environment with the ability to plan ahead. Their system consists of three independent and concurrent components (see Fig. 9): a *reactor*, based on Drummonds plan-net formalism, that reacts to the current environment; a *projector*, that determines the future effect of possible next actions, and advises the reactor on which ones best satisfy the systems objectives; and a *reductor* that advises the projector about which possible next actions are the appropriate ones to explore given the systems objectives (ie, the Reductor provides search control to the Projector).

The ERE architecture evolved from Drummond's work on the plan-net formalism (Drummond (1986)). This formalism is based on net theory (Peterson, 1981), and was designed to provide the ability to represent iterated actions and the sensory effects of actions. For example, the fact that a tour guide can deliver a speech about some landmark only if he/she is at that landmark and he has not delivered the speech before on that tour is represented by the plan–net of Figure 10. Circles represent conditions that must exist. Boxes represent events. When arcs join conditions and events, this denotes that the conditions enable the events. When arcs join events to conditions, this denotes that the events cause the conditions. There is much more to plan-nets than this condition-event theory, but this is sufficient for our examples. The net in Figure 10 demands that the tour guide be at a landmark x, at which a speech has not previously been given, $\neg$done(x), before the action-event speech can occur. Once the action

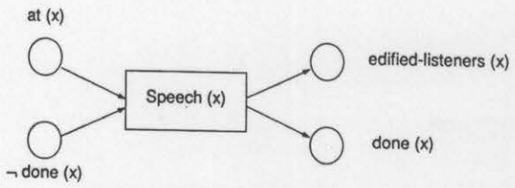

Figure 10. Example Plan-Net.

occurs, a physical change to the world occurs because the speech has happened, and the internal representation of the tour guide is updated to include done(x).

An ERE Reactor component for a tour guide that presents landmarks at random is given by the set of one instance of each of the three nets in Figure 11 for each landmark location x ∈ Landmarks. The first net is simply the speech net of Figure 10. The second net 'motivates' the tour guide to visit new landmarks. The third net models the fact that a tour will get wet if it visits an outdoor landmark while it is raining. In any given situation s, the Reactor derives the set of actions that can be carried out by the tour guide. These are simply the set of actions that are enabled in the plan-net by the situation s, enabled(s). One of these is chosen nondeterministically and carried out.

This Reactor is capable of exhibiting all tour behaviors. However, it does so in a myopic fashion, since its only criterion for selecting actions is that they be enabled. The objective of the Projector component is to use a description of the Reactor's plan-net to determine, in this case, which tours are acceptable given the current environment and a set of desired behavior constraints.

For example, consider an automated tour guide which has explicit instructions not to get the tour wet. For all worlds in which it doesn't rain, the Reactor is capable of fulfilling this constraint. However, should the world change and it starts to rain, then the Reactor will not obey this constraint on its own. This constraint, expressed in ERE's behavioral constraint language, would be (prevent tour-is-wet tour-start-time tour-end-time). Using estimates of the probability of conditions and the utility of actions, the Projector component projects forward in time the effects of action sequences that could be selected from the Reactor's plan-net (Drummond and Bresina (1990) describe a theory of temporal projection based on these plan-nets). When it finds the first sequence that obeys the behavioral constraint, it compiles a set of *Situated Control Rules* (SCRs) and sends them to the Reactor. These are if-then rules that offer the Reactor advice on the best subset of *enabled*(s) to carry out to keep in line with the behavioral constraint. In our example, the SCRs would advise the next acceptable place to go, ie, that subset of *enabled*(s) where *indoors*(s) is true. Once the Projector has found one good sequence, it continues by inspecting other less likely conditions and again updating the Reactor with SCRs when available. The nature of Projector-Reactor interaction is as follows: Without any planning, the Reactor should be able to give some sort of a tour; with some planning, the Reactor should be able to give, say, a single acceptable tour; and with lots of planning, the Reactor should be able to give many different acceptable tours (one tour being selected over another depending on various environmental circumstances).

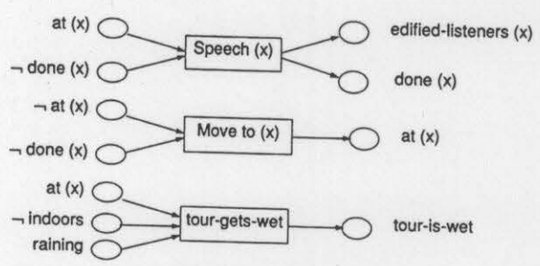

Figure 11. ERE Tour Guide.

Planner and Reactor as a Coupled System. Lyons and Hendriks and co-workers (1990, 1991) address the problem of planning for reaction by considering a Planner and a Reactor as two elements of a concurrent, cooperating system. The Planner interacts with, and is influenced by, the Reactor, in the same fashion that the Reactor interacts with, and is influenced by, the World. This architecture is shown in Figure 12. This reactor component is based on a special purpose model of computation developed for representing highly conditional robot plans. The model, called $\mathcal{RS}$, is an extension of the *Robot Schemas* model of Lyons and Arbib (1989) and inherits the philosophy of Arbib's schema theory (Arbib, 1992). It sees behavior as the result of the cooperation and competition of a set of interacting schema instances. The model is process-based, and process–algebra methods (Hoare, 1985) are used to analyze the process network behaviors. Processes can be defined in terms of networks of other processes, grounding out with a set of atomic, pre-defined, processes. This provides a powerful mechanism for specifying flexible, hierarchical structures. They make the point that to analyze the behavior of reactive machines it is very necessary to know in what sort of environment the machine will be situated. Thus, in reactive machine analysis, the $\mathcal{RS}$ model is used to represent both the plan (or controller) and the environment in which the plan will be carried out (Lyons, 1990).

An $\mathcal{RS}$ Reactor is specified by a set of recursive process equations, where processes can be coupled in networks in two ways: (1) they can communicate messages to each other via *communication ports*, or (2) they can be composed together using several kinds of *process composition operators*. An $\mathcal{RS}$ tour guide reactor, TourGuide, that

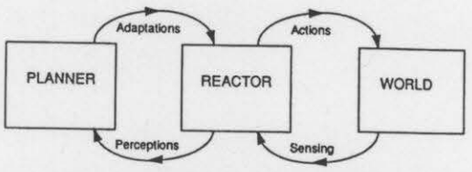

Figure 12. Planner-Reactor-World system.

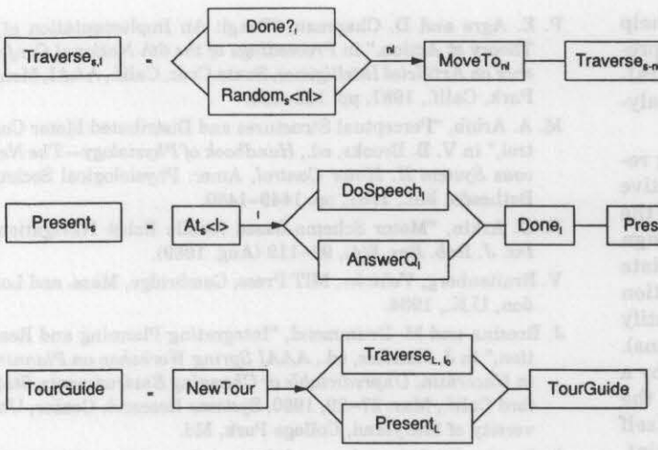

Figure 13. $\mathcal{RS}$ Tour Guide Reactor.

tours landmarks in some arbitrary order is illustrated in Figure 13, and defined algebraically as follows:

$$\text{Traverse}_{S,l} = (\text{ Done?}_l \text{ , Random}_S\langle\text{nl}\rangle \text{ }) : (\text{ MoveTo}_{nl} \text{ ; Traverse}_{S-nl,nl} \text{ })$$

$$\text{Present}_S = \text{At}_S\langle\text{l}\rangle : (\text{ (Dospeech}_l\text{,AnswerQ) ; Done}_l \text{ ; Present}_{S-l} \text{ })$$

$$\text{TourGuide} = \text{NewTour:(Traverse}_{L,l_0}\text{,Present}_L\text{);TourGuide}$$

The Traverse network consists of the concurrent network that selects the next landmark to visit, nl, from the set of landmarks, S, while monitoring for when the current landmark has been completed; it then signals a move to the new position, after which it recurses. The Present network consists of a monitor process that reports when the tour guide is first at landmark l and then concurrently interleaves delivering the landmark speech, Dospeech$_l$ and question answering, AnswerQ. When these are finished it signals the landmark is done, and recurses. (The interaction between Done? and Done is a typical application of $\mathcal{RS}$ message passing.) The reactor network, TourGuide, recursively activates a concurrent combination of Traverse and Present for each new tour. Recursion of Traverse and Present is bounded using the conditional composition operator (denoted ':'). (Conditional composition is special kind of sequential composition in which the second process is only created if and when the first process terminates successfully; if it aborts, then the composition aborts. If Random and At abort when given an empty argument, then the above reactor recursion is bounded).

The Planner "tunes" the Reactor so that the concurrent composition of Reactor and the world model, World, continue to produce appropriate behavior. The full system is then a concurrent composition of the Planner and the Reactor-World system: (Planner,(Reactor,World)). By considering a Reactor to be a set of concurrent situation-labeled actions, Lyons and Hendriks formally define what it means for the Planner to *improve* the behavior of the Reactor over time. The Planner cycles continuously, making changes to the Reactor in accordance with the definition of Reactor improvement. The Planner does not have access to the complete internals of the Reactor; its observations are restricted to the output of *perception processes* that it can embed into the Reactor. Thus, the Planner can ignore the bulk of the sensing carried out by the Reactor, and concentrate on a few important pieces of information. Perception processes are used to (1) determine when some part of the Reactor is in danger of failing, (2) reflect ongoing resource usage, and (3) determine whether certain goals or subgoals have been met.

For example, if during a tour, clouds start moving in, a perception process may signal this to the planner. From this information, the Planner deduces the possibility for rain and therefore the possibility of the tour getting wet if it is at an outdoor location. Therefore, it reasons to adapt the Reactor by changing the Traverse network to restrict the selection of landmarks to only indoor ones when it is raining, while retaining the full choice otherwise. After sending down this adaptation to the Reactor, the latter has become self-sufficient to deal with rain or sun shine, and the Planner doesn't need to worry about weather changes again. In this fashion, the Planner can use the Reactor to focus its reasoning, and in turn the Reactor is guided by the Planner to improve its behavior.

DISCUSSION AND REMAINING PROBLEMS

Reactive planning is still a young field. It has established alternatives to classical planning when operating in an uncertain and dynamic environment; however, no single reactive-planning paradigm has emerged yet. What has emerged is a number of themes that will be important to this field.

1. Reactive machines can behave intelligently and robustly. They work best when they can be designed to exploit an environment. This results in what Agre and Chapman call *routines*, patterns of behavior that are goal directed, but which are not represented explicitly in the agent, rather they are brought about as a consequence of the interaction between agent and environment. The terms *situated activity* and *emergent behavior* refer to similar phenomena.

2. A number of concepts have been developed to help build reactive machines: indexical-functional representation, behaviors and subsumption; decentralized, concurrent decision making; and formal analysis of agent and world as a coupled system.
3. The main critique on the approach of achieving reactive behavior by the *a priori* design of reactive machines (as voiced by Ginsberg, 1989) is that of the enormous computational complexity, both at design time (as the machine should have an appropriate action for every possible situation) and execution time (as the sensing system must be able to identify all aspects of the world that distinguish situations). The latter, however, can be partially remedied by a decision-tree approach to the determination of the actual situation in which the agent finds itself (Schoppers, 1987). From an efficiency viewpoint, however, there is a problem in that large parts of the plan are irrelevant (as most situations are very unlikely to occur), yet all achieve equal attention during the design phase.
4. The weaknesses of reactive machines are clear: they can be very myopic and there are limits to their robustness. Some form of additional, longer-term reasoning is necessary. However, it is an open question as to how to integrate reactive machines and longer-term reasoning. The importance of having planning and reaction as concurrent, interacting activities has become evident.

There are still a lot of open questions in this area, and of course, much work in other fields of planning, nonmonotonic reasoning, robotics, etc, also impacts reactive planning. We list here just a few of the open areas:

1. There has been little work in integrating the reactive machine work with the extensive real-time computation field.
2. Resources play a key role in many reactive domains—the concept of 'making do' with what is available—but as yet there is no unique reactive viewpoint on resources.
3. There are a number of suggestions, but no definitive answers yet, on an appropriate way to view long-term 'planning' so as to integrate it with reaction.
4. Learning can play a key role in increasing the scope in which a reactive planning system can operate. There appears to be a synergy between reaction and learning (Sutton, 1990; Mitchel, 1990). For example, the learning component of Sutton's Dyna occupies a similar place in his architecture to the planning component in Bresina and Drummond's ERE.

BIBLIOGRAPHY

P. E. Agre, *The Dynamic Structure of Everyday Life*, Cambridge University Press, Cambridge, U.K., in press.

P. E. Agre and D. Chapman, "Pengi: An Implementation of a Theory of Action," in *Proceedings of the 6th National Conference on Artificial Intelligence*, Santa Cruz, Calif., AAAI, Menlo Park, Calif., 1987, pp. 123–154.

M. A. Arbib, "Perceptual Structures and Distributed Motor Control," in V. B. Brooks, ed., *Handbook of Physiology—The Nervous System II. Motor Control*, Amer. Physiological Society, Bethesda, Md., 1981, pp. 1449–1480.

R. C. Arkin, "Motor Schema-Based Mobile Robot Navigation," *Int. J. Rob. Res.* 8(4), 92–112 (Aug. 1989).

V. Braitenberg, *Vehicles*, MIT Press, Cambridge, Mass. and London, U.K., 1984.

J. Bresina and M. Drummond, "Integrating Planning and Reaction," in J. Hendler, ed., *AAAI Spring Workshop on Planning in Uncertain, Unpredictable or Changing Environments*, Stanford Calif., Mar. 27–29, 1990, Systems Research Center, University of Maryland, College Park, Md.

R. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE J. Rob. Aut.* RA-2(1): 14–23 (Mar. 1986).

R. Brooks, "A Hardware Retargetable Distributed Layered Architecture for Mobile Robot Control," in IEEE International Conference of Robotics and Automation, Raleigh, N.C., 1987.

R. Brooks, "A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network," *Neural Computation*, 1(1) (1989).

R. Brooks, "Intelligence without Representation," *Artific. Intell.* 47(1–3), 139–160 (Jan. 1991).

D. Chapman, "Planning for Conjunctive Goals," *Artific. Intell.* 32, 333–377 (1987).

T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," in *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Morgan Kaufman, San Mateo, Calif., 1987, pp. 49–54.

B. A. Draper, R. T. Collins, J. Brolio, A. R. Hanson, and E. M. Riseman, "The Schema System," *Int. J. Comp. Vis.* 2(3), 209–250 (1989).

M. Drummond, "A Representation of Action and Belief for Automatic Planning Systems," in *Workshop on Planning & Reasoning about Action*, Timberline, Oreg., 1986, pp. 267–289.

M. Drummond and J. Bresina, "Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction," in *Proceedings of the 9th National Conference on Artificial Intelligence*, July 29–Aug. 3, 1990, AAAI, Menlo Park, Calif., 1990.

R. E. Fikes and N. J. Nilsson, "Strips: A New Approach to the Application of Theorem Proving to Problem Solving," *Artific. Intell.* 2, 189–208 (1971).

M. Ginsberg, "Universal Planning: An (Almost) Universally Bad Idea," *AI Mag.*, 40–44 (Winter 1989).

J. Hendler, "Real-time Reaction for Planning Systems," in *AAAI Spring Symposium on Planning and Search*, March 1989, AAAI, Menlo Park, Calif., 1989, pp. 24–26.

J. Hendler, "Abstraction and Reaction," in J. Hendler, ed., *AAAI Spring Workshop on Planning in Uncertain, Unpredictable or Changing Environments*, Stanford, Calif., Mar. 27–29, 1990, Systems Research Center, University of Maryland, College Park, Md., 1990.

C. A. R. Hoare, *Communicating Sequential Processes*, International Series in Computer Science, Prentice-Hall, New York, 1985.

L. P. Kaelbling, "An Architecture for Intelligent Reactive Systems," in *Workshop on Planning & Reasoning about Action*, Timberline, Oreg., 1986, pp. 235–250.

L. P. Kaelbling, "Goals as Parallel Program Specifications," in

Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, Minn., 1988, AAAI, Menlo Park, Calif., 1988, pp. 60–65.

D. M. Lyons, "A Formal Model for Reactive Robot Plans," in *2nd International Conference on Computer Integrated Manufacturing,* RPI, Troy, N.Y., May 21–23, 1990.

D. M. Lyons and M. A. Arbib, "A Formal Model of Computation for Sensory-Based Robotics," *IEEE Trans. on Robotics Automation* 5(3), 280–293 (June 1989).

D. M. Lyons, A. J. Hendriks, and S. Mehta, "Achieving Robustness by Casting Planning as Adaptation of a Reactive System, in *IEEE International Conference on Robotics and Automation,* Apr. 7–12, 1991, IEEE, New York, 1991.

D. M. Lyons, R. Pelavin, A. Hendriks, and P. Benjamin, "*RS*: A Formal Model for Reactive Robot Plans," in J. Hendler, ed., *AAAI Spring Symposium on Planning in Uncertain and Changing Environments,* Stanford, Calif., Mar. 27–29, 1990, Systems Research Center, University of Maryland, College Park, Md., 1990.

T. Mitchell, "Becoming Increasingly Reactive," in *Proceedings of the 9th National Conference on Artificial Intelligence,* Boston, Mass., July 29–Aug. 3, 1990, AAAI, Menlo, Calif., 1990, pp. 1051–1058.

N. J. Nilsson, "Action Networks," in J. Weber, J. Tenenberg, and J. Allen, eds., *From Formal Systems to Practical Systems,* Dept. of Computer Science, University of Rochester, Rochester, N.Y., 1988, pp. 21–52.

N. J. Nilsson, "Toward Agent Programs with Circuit Semantics," unpublished.

J. L. Peterson, *Petri-Net Theory and the Modelling of Systems,* Prentice-Hall, New York, 1981.

S. Rosenschein, "Synthesizing Information-Tracking Automata from Environmental Descriptions," in R. Brachman, H. Levesque, and R. Reiter, eds., *Proceedings of 1st International Conference on Principles of Knowledge Rep and Reasoning,* May 1989, Toronto, Canada, Morgan Kaufman, San Mateo, Calif., pp. 386–393.

S. J. Rosenschein and L. P. Kaelbling, *The Synthesis of Digital Machines with Provable Epistemic Properties,* Technical Note 412, SRI International, Menlo Park, Calif., April 1987.

J. Sanborn and J. Hendler, "A Model of Reaction for Planning in Dynamic Environments," *AI in Eng.* 3(2), 95–102 (1988).

M. J. Schoppers, "Universal Plans for Reactive Robots in Unpredictable Environments," in *Proceedings of the 10th International Joint Conference on Artificial Intelligence,* 1987, Morgan Kaufman, San Mateo, Calif., 1987, pp. 1039–1046.

M. Schoppers, *Representation and Automatic Synthesis of Reaction Plans,* Report, Dept. of Computer Science, University of Illinois, Urbana-Champaign, Ill., 1989.

L. Spector and J. Hendler, "An Abstraction-Partitioned Model for Reactive Planning," in Y. Wilks and P. McKevitt, eds., *Fifth Rocky Mountain Conference on AI,* Computing Research Laboratory, New Mexico State University, Las Cruces, N.M., June 1990.

M. Stefik, "Planning with Constraints (Molgen: Part 1); and Planning and Meta-Planning (Molgen: Part 2)." *Artific. Intell.* 6, 111–170 (1981).

R. Sutton, "First Results with Dyna," in J. Hendler, ed., *AAAI Spring Symposium on Planning in Uncertain and Changing Environments,* Stanford, Calif., Mar. 27–29, 1990, Systems Research Center, University of Maryland, College Park, Md., 1990.

A. Tate, "Generating Project Networks," in *Proceedings of the 5th International Joint Conference on Artificial Intelligence,* Cambridge, Mass., 1977, Morgan Kaufman, San Mateo, Calif., 1977, pp. 888–893.

S. Vere, "Planning in Time: Windows and Durations for Activities and Goals," *IEEE PAMI,* 5(3), 246–267 (1983).

D. E. Wilkins, "Domain-Independent Planning: Representation and Plan Generation," *Artific. Intell.* 22(3), 269–301, 1984.

D. M. Lyons
A. J. Hendriks
North American Philips Corp.

A synopsis of the diverse work in this field would not have been possible without the cooperation of other researchers. The authors thank the following people for their insightful comments and helpful discussions: Phil Agre, Michael Arbib, Marc Drummond, Jim Hendler, Nils Nilsson and Marcel Schoppers.

PLAUSIBLE REASONING. See Reasoning, plausible.

POLITICS

A system that simulates human ideological understanding of international political events, POLITICS was written in 1979 by Carbonell at Yale University and is a successor to MARGIE and a predecessor to BORIS (qv) (see J. Carbonell, "POLITICS," in R. C. Schank and C. K. Riesbeck, eds., *Inside Computer Understanding: Five Programs Plus Miniatures,* Lawrence Erlbaum, Hillsdale, N.J., 1981, pp. 259–307).

M. R. Taie
AT&T Bell Laboratories

POPLOG

POPLOG is an AI environment for teaching, research, and product development. It provides a vehicle for implementation of multiple languages based on a common virtual machine from which machine-code for target machines is generated, and which provides a "compiler toolkit" for further language development. It is designed to realize AI technology on standard hardware. Its hardware requirements are moderate; it needs no special-purpose architecture and has an executable image that can be below 1 MB. Yet unlike most LISP systems, it is capable of exploiting the full addressing capability of 32-bit machines. POPLOG is used to "rescue" AI applications that have "run out of steam" on a restricted vehicle, such as an expert systems shell.

The system was developed at Sussex University and incorporates fundamental work by Edinburgh University on languages for AI. It provides the following integrated capabilities:

1. POP-II. The system implementation language is derived from Burtstall and Popplestone's POP-2 (see R. M. Burstall, J. S. Collins, and R. J. Popplestone,

POP-2 Papers, Edinburgh University Press, 1968). (See POP-11.) It provides symbolic processing.

2. PROLOG. The POPLOG Virtual Machine was developed by Gibson and Hardy and is descended from the Elliott 4130 designed by Hoare and co-workers (which was the target machine for the first implementation of POP-2). As a result of a collaboration with Mellish, POP-11 and Prolog were implemented as intercallable languages, sharing one environment on the VAX computer.
3. Common LISP. The implementation by Williams was originally oriented toward providing a secure and convenient implementation of the language; it has subsequently been tuned to exploit the code-generating capabilities of the Virtual Machine.
4. ML. This strongly typed functional language has a stylistic relationship to POP-2 and PROLOG, and is based on the work of Darlington, who developed the use of recursion equations as a functional equivalent to the logic clauses embodied in PROLOG. The type inference system, due to Milner, supports polymorphism and thus obviates much of the strait-jacket that earlier type-systems (such as that of Pascal) placed on expressivity. The Poplog implementation is by Nichols and Duncan.
5. VED. An integrated editor compiles selected code incrementally and supports the online documentation. Operating in either native or EMACS mode, this operating system interface implements a range of system calls interactively and uniformly and allows access to significant Unix and VMS capabilities.
6. Compiler toolkit. Supports the implementation of new languages within the environment and allows the user to create efficient code by calling a standardized set of code-planting procedures. The planted code operates on a POPLOG virtual machine, which provides the capabilities required for POP-11, Common Lisp, and PROLOG.

The anticipated market role of X-windows has effected the recent evolution of POPLOG. The product has evolved to align itself with a C-based approach to X. While CLX is supported, it is not seen as primary, and the major emphasis has been in supporting OpenLook and Motif in addition to a native POPLOG widget set. This in turn has had an impact on the virtual machine architecture, in order to enhance compatibility with C. In order to support callback from C to the POPLOG languages, data-structures managed by the garbage collector can be orthogonally given a "fixed" attribute which causes them to be nonrelocatable, thus avoiding one of the notorious pitfalls of interworking. POPLOG pointers and C-pointers are now identical, since the "dope" information required for the AI languages is placed before the location pointed to. Moreover, a capability of describing structures in POP-11 that are fully compatible with C-structures has been provided. [See also C. Mellish and S. Hardy, "Integrating Prolog into the POPLOG Environment," in *Proceedings of the Eighth IJCAI*, Karlsruhe, FRG, 1983.

Robin Popplestone
University of Massachusetts

POP-11

This language is derived from Burstell and Popplestone's POP-2 (see POPLOG). It provides symbolic processing and is the system implementation language. The original POP-2, although owing most to the early work on LISP by McCarthy and co-workers, was influenced syntactically by Algol 60 and semantically by Strachey's CPL and Landin's ISWIM. It provided a uniform name-space encompassing both functions and data, closures by lambda-lifting, and (in 1971) continuations. It was the original implementation language of the Boyer-Moore theorem prover (see R. S. Boyer and Moore J. Strother, *A Computational Logic*, Academic Press, New York, 1979). POP-2 supported work on structure-sharing in implementations of Robinson's resolution and a pioneering study in the integration of vision and touch in robotics. Another aspect, due to Michie [D. Michie, Memo Functions and Machine Learning, *Nature* **218**, 19–22 (1968)] was memoization of functions, permitting flexible, user controlled trade-offs between space and time.

Technological developments underlying POP-2 included incremental compilation supporting interactive computing, a uniform space for atoms and lists under the control of a generalized relocating garbage collector, and a restricted form of lightweight process, which supported the use of POP-2 as a command language for the Multipop time sharing system. Modernized as POP-11 by Sloman and Hardy, the language has taken on board lexical variables (from Scheme) and the representation of numbers of Common LISP. The syntax was made more redundant as a means of catching errors earlier in the program development process, and an on-line help system provides hundreds of help files covering languge features and various aspects of AI, with hypertext-like cross-linking.

Robin Popplestone
University of Massachusetts

PREDICATE LOGIC. See Logic, predicate.

PREFERENCE SEMANTICS

Preference semantics (PS) had its philosophical origins in Wilks (1971, 1972) who argued that

1. To have a meaning is to have one from among a set of possible meanings.
2. Giving meaning is the process of choosing or preferring among those.