Chapter 1

Introduction

> κακοι μαρτυρεσ ανθρωποισιν οφθαλμοι και ωτα βαρβαρουσ ψυχασ εχοντων.
>
> Eyes and ears are poor witnesses for men if their souls do not understand the language.
>
> Heraclitus

This dissertation investigates the utility of abstraction for agents living in complex, dynamic environments. Abstraction is a historically broad concept, applied through the centuries to problems in virtually every intellectual discipline. Since the early 1960's, several well-defined notions of abstraction, hierarchy, and levels have proven to be valuable in philosophy, psychology, and artificial intelligence (AI) research.

Many of the outstanding problems in the cognitive sciences, and particularly in AI, concern the complexity and dynamism of the environments in which cognitive agents must function. While AI has made great strides in the study of problem-solving methods, the integration of these methods into intelligent behavior in our ever-changing world presents one of the field's current challenges. This dissertation examines the role that certain forms of abstraction can play in meeting this challenge.

Recent work in AI planning systems suggests that the "classical" planning paradigm, based on the plan/execute model of action and on its concomitant assumptions of a static world and an omniscient reasoner, is inadequate for agents in complex and dynamic environments. A rejoining trend has developed, both in AI planning systems and in robotic control theory, toward reactive systems based on distributed control structures. For reasons of efficiency and modularity these systems are often partitioned into layers of abstraction.

Two general ideas of abstraction have hitherto played an important role in AI research. The first involves the division of systems into "levels of detail," exemplified in AI planning systems by Sacerdoti's ABSTRIPS [Sacerdoti 1974]. This type of simplification abstraction has received considerable recent attention and is now fairly well understood; rigorous formalizations have been provided that demonstrate the utility of simplification abstraction in reducing the size of certain problem-solving search spaces (e.g., [Knoblock 1991b]). A second form of abstraction has been used in robotic control theory and in research on large-scale cognitive models. In this form of abstraction, cognitive processes are divided into levels, providing flexibility of behavior through the use of modularity and parallelism. This kind of abstraction is exemplified by work on blackboard architectures such as HEARSAY-II [Erman et al. 1980], in which novel control structures can be used to generate, for example, opportunistic problem-solving behavior.

This dissertation presents, formalizes and demonstrates a species of abstraction called supervenience. The concept of supervenience captures the notions of abstraction that are most important for systems that must integrate high-level reasoning with real-time action. Simplification abstraction is a special case of supervenience, so the search-reduction benefits of ABSTRIPS-style systems are usually, but not always, available in supervenient planning systems. The generality of supervenience allows, however, for uses of abstraction that are not possible in any ABSTRIPS-style system. For example, a supervenient planning system can make use of different knowledge representation systems and different reasoning mechanisms at each level of the abstraction hierarchy. Although similar capabilities may be found in existing blackboard systems and robotic control architectures, the principles that underlie this form of abstraction have not been sufficiently explored until now.

The central idea of supervenience is that representations at lower levels of abstraction are epistemologically "closer to the world" than those at higher levels, and that the representations at higher levels therefore <u>depend</u> on those at lower levels. The higher levels may contain representations that are simplifications of low-level, sensory reports, but they may just as well contain representations that are complex, structurally rich aggregates that have no unified representation at lower levels. In contrast to ABSTRIPS-style systems, in which higher levels must be simplifications of the lower levels, levels of supervenience may be dissimilar in various ways so long as the proper dependence relation holds. The thesis is that it is this <u>dependence</u>, and not the more restrictive notion of simplification, that allows for the flexible integration of cognition and action.

The conception of abstractness as epistemological "distance from the world" is not new. Rudolph Carnap, in his 1937 <u>Foundations of Logic and Mathematics</u>, writes:

> We find among the concepts of physics—and likewise among those of the whole of empirical science—differences of abstractness. Some are more elementary than others, in the sense that we can apply them in concrete cases on the basis of observations in a more direct way than others. The others are more abstract; in order to find out whether they hold in a certain case, we have to carry out a more complex procedure, which, however, also finally rests on observations. Between quite elementary concepts and those of high abstraction there are many intermediate levels. [Carnap 1960, 150]

Carnap's conception of abstraction is similar to supervenience. In the following pages the concept of supervenience is discussed, formalized, and applied to an implemented dynamic-world planning system.

Part I of the dissertation (Chapters 2 and 3) is largely historical. Chapter 2 describes recent developments in AI planning systems and discusses the emergence of dynamic-world planning as a critical area within the field. Previous approaches to dynamic-world planning are surveyed, and the relations between dynamic-world planning and other branches of planning research are sketched. Chapter 2 concludes with descriptions of specific behavioral problems that present difficulties for the current generation of dynamic-world planning systems.

Chapter 3 surveys the use of abstraction in AI planning systems. Special attention is paid to two forms of abstraction: reduced partition abstraction (the predominant form of simplification abstraction—used for example in ABSTRIPS-style systems), and partitioned control abstraction (used in robotic control systems and blackboard architectures). These forms of abstraction are discussed in relation to one another, and in relation to the dynamic-world planning systems mentioned in Chapter 2.

Part II of the dissertation (Chapters 4–6) presents the theory of supervenience. Chapter 4 traces the concept of supervenience from its roots in the philosophical literature to its application in AI planning systems. Chapter 5 describes supervenience formally, in the context of nonmonotonic reasoning systems. Chapter 6 uses the formal constructions of Chapter 5 to compare supervenience to the form of abstraction used in ABSTRIPS-style systems. The claim is made that supervenience is a generalization of reduced partition abstraction—a generalization that is particularly important for agents living in complex, dynamic environments.

Part III (Chapters 7–9) describes a dynamic-world planning system based on the concept of supervenience. Chapter 7 presents the general computational architecture of the system, called the supervenience architecture. The chapter also includes a comparison of the supervenience architecture to the *subsumption architecture,* a well-known architecture for robotic control systems.

Chapter 8 details the components of the Abstraction Partitioned Evaluator (APE), a single instance of the supervenience architecture. While the supervenience architecture is based on the use of levels, it does not mandate the use of any particular set of levels. A specific set of levels is used in APE; this set of levels is described and supported through evidence from philosophy, psychology, and AI research. Chapter 8 also details the programming constructs provided by APE, and discusses the use of these constructs for critical operations in dynamic-world planning. The chapter concludes with a discussion of the potential for parallelism in future versions of APE, and in the supervenience architecture in general.

Chapter 9 describes HomeBot, a dynamic-world planner built using APE. HomeBot runs in a simulated household environment, and will be used to demonstrate important features of the supervenience architecture. Several examples of HomeBot's behavior are presented in detail, including solutions to some of the difficult behavioral problems mentioned in Chapter 2. The chapter concludes with an assessment of HomeBot's performance, and a discussion of how it may be improved.

Part IV (Chapter 10) summarizes the main points of the dissertation, and discusses directions for future work. The dissertation covers a wide range of topics, from philosophy and psychology to theoretical and practical AI. The synthesis has important implications for cognitive science and AI practice, and many of the component questions may also deserve further exploration.[1]

---

[1]Earlier work on the project described in this dissertation was reported in [Spector and Hendler 1990a, 1990b, 1991a, 1991b].

Part I

Planning, Reaction, and Abstraction

Chapter 2

Planning and Reaction

2.1 Static-World Planning

Planning has been an active subfield of AI for over 30 years, and the literature on planning systems is large and varied. In Part I of this dissertation I explore two threads of work in the literature: the use of abstraction in planning systems, and the development of integrated planning/acting technologies. The contribution of this dissertation is a generalized concept of abstraction that can be used to attack some of the outstanding problems in the integration of planning and action. The history of the planning field is sketched in order to provide context for the discussion, but it is outside the scope of this work to chronicle the entire history of AI planning, or even to touch upon all of the issues of interest to planning researchers. Several summaries of work in the field, as well as reprints of seminal papers, can be found in [Allen, Hendler, and Tate 1990].

As in many areas of AI, early research in planning systems used simplified problem domains in order to more directly address the issues that were perceived to be at the core of the field. In the case of planning, this resulted in (at least) the following simplifying assumptions:

• The robot is the only agent active in the environment; there are no competitors, collaborators, or natural forces that might interfere with the state of the world either during the planning process or during execution.

• The robot's primitive actions will succeed, and will modify the state of the world according to their specifications, on every application.

• The robot is capable of performing a single action at a time.

• Planning time doesn't matter (except that it is best for it to be minimized).

• Execution time doesn't matter (except that it is best for it to be minimized).

I will call these assumptions the <u>static world assumptions</u>, and I will call planning in the context of such assumptions <u>static-world planning</u>. These terms are slightly misleading, since it is not only the dynamism (vs. stasis) of the world that is in question. Nonetheless, the terms have some history (e.g., [Hendler and Sanborn 1988], [Sanborn 1989]), and their meanings are reasonably clear. Other terms used in the literature for static-world planning are "classical planning" and "traditional planning"; these terms convey little useful information and are likely to age poorly as fewer new planners make the static-world assumptions.

Planning is difficult even in the context of the static-world assumptions, and many problems of static-world planning remain unsolved after years of research. Much of the early work was based on a formalization of action known as the <u>situation calculus</u> [McCarthy 1968], in which actions are modeled as functions on <u>situation variables</u> in a logical calculus. A "situation" logically encapsulates the "state of the world," and actions defined in the situation calculus can be considered as state-transformation operators. This conception invites the development of planning systems that use state-space search techniques in order to discover action sequences that transform the state of the world to conform with a set of goals.

An early and influential planning system that was based on the state-transformation model was STRIPS [Fikes and Nilsson 1971]. STRIPS simplified the representation of actions in the situation calculus by specifying state-transformations as <u>operators</u> composed of three lists of formulae: the <u>preconditions</u> that must be satisfied for the operator to be applicable, the add-list formulae that the operator would cause to be true, and the delete-list formulae that the operator would cause to be false. STRIPS also assumed that formulae not mentioned in an operator's add-list or delete-list would be unaffected by the application of that operator. This assumption, which has come to be known as the STRIPS assumption, can be seen as one approach to a deep problem in the theory of action called the frame problem. The frame problem is the problem of determining what aspects of the world remain unaffected by the performance of some particular action. The STRIPS assumption is a pragmatic solution in many simple domains, but in complex domains it becomes impossible to explicitly specify all effects of each action.

The frame problem, and associated problems such as the precondition qualification problem and the ramification problem (see [Georgeff 1987]), arise even in the context of the static-world assumptions, and solutions to these problems are still the subjects of considerable controversy (see, e.g., [Pylyshyn 1987], [Kyburg, Loui, and Carlson 1990]). Further, even when the STRIPS assumption (or any other "cheap" solution to the frame problem) is employed, the design and implementation of efficient planning systems remains difficult.

Recent advances in the design of planning algorithms and of their supporting knowledge representation systems have improved the performance of STRIPS-style systems, but significant complexity problems remain. Whereas STRIPS represents plans as linear (totally-ordered) sequences of actions, researchers have argued that the use of so called non-linear (partially-ordered) representations can reduce the size of the planning search-space (e.g. in NOAH [Sacerdoti 1975], [Sacerdoti 1977] and in NONLIN [Tate 1976]). The use of non-linear plan representations is an example of a least-commitment search strategy; Stefik's MOLGEN system generalized this strategy and pioneered a constraint posting framework for nonlinear planning [Stefik

1981]. Chapman's TWEAK system further refined the constraint posting formulation [Chapman 1987]. Some researchers have recast planning problems as more traditional search problems (e.g., [Korf 1987]), and have thereby applied advanced search strategies to the improvement of planning algorithms. Unfortunately, analytical work has shown that these improvements have a limit; nonlinear planners are condemned to exponential worst-case complexity [Chapman 1987], and it has been shown that even in the knowledge-impoverished blocks-world domain planning problems are generally at least NP-hard [Bylander 1991], [Chenoweth 1991], [Gupta and Nau 1991]. Advanced planning architectures such as SIPE [Wilkins 1988] and O-Plan [Currie and Tate 1991] allow for the use of search heuristics to combat this complexity. Recent studies have related the complexity and decidability of planning problems to the nature of the operators allowed in a given system [Erol et al. 1992].

The complexities of planning are largely due to the generality of the problems that planning systems are expected to solve. Within the applied AI community significant progress has been made in the design of <u>domain dependent</u> planning systems—systems custom-built for particular applications. As with many AI technologies, special purpose systems can be engineered to achieve improved performance at the expense of flexibility and generality. Most work in the academic AI planning community, however, focuses on <u>domain independent</u> techniques. This is because domain dependent systems, though useful, contribute little to the understanding of the larger issues in the field. Note however that the distinction between domain dependence and domain independence is not crisp; for example, recent work on the complexity of blocks world planning shows that the details of <u>how</u> the blocks world domain is encoded into operators may have a significant impact on the complexity of planning problems [Bylander 1991]. Certain "domain independent" techniques may be <u>applicable</u> in a wide range of domains but <u>appropriate</u> only in a handful, and the appropriateness of a technique may depend not only on the domain but also on the manner in which it is encoded. It is nonetheless useful to distinguish systems developed for a single application (e.g., for controlling a particular factory) from those intended as portable technologies. It is likely that useful planning systems in the near future will all be domain dependent in some sense, but it is only by studying the principles of planning in general that the flexibility and generality of such “domain dependent” planning systems will be improved.

In the wake of discouraging complexity results, several groups of researchers have been advancing fundamentally different approaches to the production of intelligent behavior. Work in <u>case-based</u> <u>planning</u> is based on the intuition that it may be more efficient to retrieve and to modify old plans than it is to plan from scratch. This work is also supported by evidence from cognitive psychology that indicates that human planning is largely memory based (see, e.g., [Schank 1982]). Case-based planning research involves, aside from the work on planning algorithms *per se,* the development of several underlying technologies. These include systems for knowledge representation and indexing, plan matching, plan retrieval, and plan modification. While it is certain that case-based techniques will eventually form an important part of advanced planning systems, case-based retrieval does not by itself obviate the need for more traditional planning. Unless case-bases contain plans for <u>every</u> eventuality, systems will still require algorithms for plan-modification, and perhaps also for “from-scratch” planning. Some current case-based planning systems are designed to be directly integrated into more traditional planning systems; for example Kambhampati’s PRIAR system retrieves plans, reinterprets them in the context of new problems, and then passes them to a hierarchical nonlinear planner for refitting [Kambhampati 1990], [Kambhampati and Hendler 1992].

Another approach to coping with the computational complexity of planning is to reduce the need for planning by endowing a system with some sort of "situated intelligence" [Suchman 1987]. This work is based upon the idea that it is often possible to generate reasonable behavioral sequences by applying "situated" control rules at each moment. Such rules may be used to choose the next action to perform without long-term planning, and the resulting systems can generate surprisingly "intelligent" behavioral sequences. This approach has the further advantage that it works equally well when the static-world assumptions listed above are violated, although just how well it works is the subject of considerable controversy (see Section 2.2).

The rejection of the static-world assumptions has generated a large body of work that I will call <u>dynamic-world planning</u>. Dynamic-world planning focuses on the problems that don't arise in the simplified domains of static-world planning systems: uncertainty in the world, the possibility of failure, the presence of other agents or of unpredictable natural forces, etc. Such violations of the static-world assumptions are commonplace in many interesting real-world domains.

The phrase "reactive planning" has also been used to describe dynamic-world planning, but the term "reactive" generally indicates that behavior is being generated <u>in the absence</u> of goal-driven deliberation. Dynamic-world planning, by contrast, incorporates the progress made in static-world planning, domain-dependent planning, case-based planning, and other fields, in order to tackle unrestricted "robot behavior" problems in dynamic domains.

Dynamic-world planning research plays an important role in planning research as a whole. The intractability results of static-world planning are discouraging, but it might be the case that large, static-world planning problems rarely occur in actual domains. Similarly, the difficulties of producing "real-time" reactive behavior should be put in perspective—we do not yet know enough about the performance requirements that constitute “sufficient” reactivity. Research in dynamic-world planning studies the generation of intelligent behavior in the large, without restricting attention to the special cases of static domains or knowledge-free reactions. It thereby provides a framework in which to evaluate the contributions of other areas of planning research.

2.2 Generating Planned Activity

Although the field of AI planning has always been directed toward the control of robots in the real world, the static-world assumptions have guided researchers to design systems that could not be extended to handle the complexity, the precariousness, or the dynamism of real-world domains. A principal feature of early planning systems that derives from the static-world assumptions is the plan-execute cycle.[2] The idea of the plan-execute cycle is that the planner produces a complete plan and then hands it to an execution module that “does it.” If the execution module encounters problems, or if new goals are posted, then the planner may be re-invoked. I will refer to planning in the context of this sort of plan-execute cycle as planning in the total-planning framework. More specifically, the total-planning framework entails the following assumptions:

• Planning problems are specified as state-transformation problems, with initial and final state descriptions provided to the reasoner.

• Solutions consist of complete sequences of primitive actions, directly executable by the robot, that are guaranteed to transform initial states to final states.

The initial and final states need not be specified completely; that is, the state descriptions need not specify the values of all domain predicates in either the initial or

[2] Brooks describes traditional AI systems as operating within a sense-model-plan-act (SMPA) framework [Brooks 1991, 570]. The plan-execute cycle is the segment of the SMPA framework that is highlighted in planning research, problems of sensation and model building have generally been relegated to other fields.

goal situations. It is implicit in the total-planning framework, however, that the problem specification includes sufficient information to produce complete plans that are composed only of primitive actions. Solutions may be specified in various ways; for example, nonlinear planners may produce partially ordered networks of primitive actions. The essential feature of total-planning systems is that the planner is expected to solve the entire problem first, before handing the solution to a “dumb” execution module.

Some of the work in dynamic-world planning fits into the total-planning framework. The plan-execute cycle allows for mechanisms that detect plan failures, triggering “replanning” procedures in the next “plan” phase (e.g., [Hayes 1975]). Methods have been developed for scheduling failure detection tasks, and for tailoring replanning for specific plan failures (e.g., [Doyle et al. 1986]). The case-based techniques of Kambhampati’s PRIAR system can be viewed as a refinement of some of these methods. Replanning systems are attractive because they build on the established techniques of static-world planning, but they have major drawbacks in dynamic environments. One problem is that replanning is triggered only after failure has been detected; in some cases this is not sufficient (see Section 2.3). A more significant problem is that replanning, while more efficient than planning from scratch, is still a time-consuming process. The reactivity of a system based on the plan-execute cycle is limited by the speed of replanning algorithms, and current research suggests that replanning delays will be debilitating in complex dynamic environments.

A technique suggested by Schoppers involves the construction of <u>universal</u> <u>plans</u> that specify conditional action sequences guaranteed to achieve their goals regardless of changes in the world [Schoppers 1987]. The conditionals in a universal plan require sensation and in some sense determine the “plan” at execution-time;

considered in this light the approach falls outside of the total-planning framework. But Schoppers contends that the real work is in constructing the universal plan, not in arbitrating actions at execution time. A universal plan provides a complete solution to a given problem in terms of primitive actions, and it is fully constructed before execution begins; in this sense the universal plans approach is a type of total-planning. Serious doubts have been raised, however, about the tractability of universal plan construction and about the practicality of storing universal plans. According to some researchers, the universal plans approach is problematic at best (see the debate in AI Magazine, Winter 1989).

Enhancements to total-planning systems may improve performance and reactivity, but it has become widely recognized that the assumptions of the total-planning framework, while convenient for their relation to standard AI techniques, are at odds with the requirements of actual robots in the real world. The weaknesses of the framework take several forms. The following weaknesses are independent of the static-world assumptions:

• Some information becomes available only at execution-time, while total-planning systems rely on having all relevant information at plan-time.

• The acquisition of information at execution-time (sensation and perceptual interpretation) is itself behavior that must be planned, and information necessary for the planning of later perceptual activities may depend on the results of earlier perceptual activities.

• Some actions are inherently iterative, with the world providing bounds on the iteration (for example, hammering a nail).

When the static-world assumptions do not hold, the total-planning framework manifests the following additional weaknesses:

• Even when the initial situation is entirely known to the planner at plan-time, descriptions of the world in the initial situation may not be valid at execution-time.

• Total-planning is computationally expensive, and some problems of reasoning about action may even be undecidable; time wasted planning for expectations that may not materialize, or for contingencies that may not occur, may be debilitating.

• If a domain is essentially fluid and unpredictable then the notion of a correct total-plan is not even meaningful; the world must be reasonably stable for precomputed sequences of actions to achieve their goals, and any attribution of "correctness" to plans in lieu of such stability is groundless.

• The world imposes real-time constraints that are sometimes severe, exacerbating the above-listed difficulties.

The rejection of the total-planning framework, along with the rejection of the static world assumptions, leads to a framework that I will call <u>generating planned activity</u>. The focus on the <u>plan</u> in the total-planning framework is dropped in favor of a focus on the behavior of the system in its environment. In this framework plans are considered useful insofar as they enable an agent to behave intelligently, but the goal of generating correct plans is secondary to the goal of producing appropriate behavioral sequences. The <u>situated</u> nature of cognitive agents is particularly emphasized [Suchman 1987].

In 1987 Agre and Chapman made a strong pitch for a re-orientation of planning research, away from the total-planning framework and toward the framework of generating planned activity.[3] They proclaimed 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*. . . . Life is fired at you point blank: when the rock you step on pivots unexpectedly, you have only milliseconds to react. Proving theorems is out of the question. [Agre and Chapman 1987]

---

[3]Agre and Chapman use different terminology; they use "capital-P Planning" for the total-planning framework, and "lower-case-p planning" for their proposed alternative.

Agre and Chapman developed a system called Pengi that played a simple but dynamic video game called Pengo, and their work introduced several important concepts into the vocabulary of dynamic-world planning. For example, Pengi used indexical-functional entities (such as “the-block-I’m-pushing”) to represent the agent’s environment, instead of traditional predicate and variable forms. This has proven to be a valuable technique for dynamic-world planning, and has been studied and extended by other researchers (e.g., [Schoppers and Shu 1990]).

While Pengi embodied a “planless” model of activity, Agre and Chapman did not deny that plans would have a role to play in more complex systems (see, e.g., [Chapman and Agre 1986]). Nonetheless, their work has served as a touchstone for research based on the more radical assumption that plans are unnecessary and that intelligent behavior is best engineered through the composition of simple, planless, behavioral units. Brooks, a vocal proponent of this view, advocates “intelligence without reason” and builds “artificial insects” that generate complex and appropriate behavioral sequences in dynamic environments, despite their lack of plans [Brooks 1991].

Brooks’s work has also provided valuable contributions to dynamic-world planning. For example, his subsumption architecture provides a link between the planning literature and control theory that will be examined further in Section 3.3. Taken in its most radical form, however, his anti-plan stance leads to problems as severe as those of the replanning framework. I will call models of activity based on the anti-plan stance models of pure reaction, since advocates of such systems envision agents that behave intelligently through interaction with the world that is not mediated by “deliberation” or “plan-formation,” as these words are traditionally used. Pure reaction can be seen as the complementary extreme of the replanning framework; while replanning makes a minimal concession to the dynamism of the world in exploiting the plan-execute cycle, pure reaction makes a minimal concession to the complexity of reasoning by allowing the composition of reactive units. Just as a replanning system is ill-equipped to play a video game, so is a purely reactive system ill-equipped to prove theorems or, for example, to plan a vacation itinerary.

While debates about the ultimate power of purely reactive systems are still being waged, many researchers are developing new frameworks for dynamic-world planning that lie somewhere between replanning and pure reactivity. Maes has developed a system that is similar in spirit to purely reactive systems, but that allows for run-time computation of action selection strategies and "full-fledged goals" [Maes 1990]. Several researchers have developed systems that "compile" reactive systems from higher level specifications, either before or during execution (e.g., [Rosenschein and Kaelbling 1986], [Nilsson 1991]). Correspondingly, work in deliberative planning has begun to focus on algorithms that can be interrupted or influenced by changes in the world (e.g., the anytime algorithms of [Dean and Boddy 1988]). The bulk of work in dynamic-world planning has been concerned with developing techniques to span the gulf between simple reactivity and expensive, powerful, deliberative reasoning. The approaches are numerous and varied; several conference sections and entire workshops have been held on the topic (e.g., [Georgeff and Lansky 1986], [Sycara 1990]). The phrase "generating planned activity" was chosen to cover the large number of the proposed approaches that share the aim of modeling goal-directed, knowledge-based behavior in complex dynamic domains.

Chrisman has developed a framework in which proposed dynamic-world planning architectures may be classified according to the features of the domains in which they are most suited to operate. Environments are classified along three dimensions: the complexity of world states, the predictability of future events, and the density of "critical choice points" [Chrisman et al. 1991]. In a similar vein, Kinney and Georgeff have developed an experimental framework in which competing strategies for dynamic-world planning may be compared [Kinney and Georgeff 1991]. In this work a small set of similar agent architectures, all built in PRS [Georgeff and Ingrand 1989], are compared with respect to their performance in the Tileworld testbed [Pollack and Ringuette 1990]. Although both the range of agents and the complexity of the domain are quite limited,[4] this work allows for meaningful comparisons that may help to impose some order on the large and heterogeneous set of proposed architectures.

Although comparative work such as [Chrisman et al. 1991] and [Kinney and Georgeff 1991] shows that the literature of dynamic-world planning has reached a level of maturity at which evaluations can be made, certain issues have yet to be adequately addressed in the dynamic-world planning literature to date. One of the key outstanding problems involves the necessity for <u>integration</u> of reactive and deliberative components of dynamic-world planning systems. Various combinations of deliberation and reaction suffice in artificial domains that are slightly more complex than Blocks World, but real domains impose more stringent requirements. In certain complex domains intelligent deliberation requires consideration of information acquired from execution-time monitoring. Similarly, in certain domains intelligent reactivity relies on access to symbolic knowledge computed by deliberative processes. I will argue that such requirements are the norm, rather than the exception, even in moderately complex domains such as household cleaning. The generation of appropriate behavioral sequences in such domains requires the smooth integration of reactive and deliberative capabilities, with information flowing both from reactive to deliberative components, and from deliberative to reactive components. In the next section I will describe simple problems that reveal the need for such a capability.

[4]See Section 9.1 for further discussion of domains and domain simulators.

2.3 Problems of Integrated Behavior

The recent shift of interest in the planning community to architectures for generating planned activity does not imply that the problems of static-world planning have been solved. On the contrary, many of the problems that have concerned investigators from the earliest days of planning research are still unsolved, and much of the work in dynamic-world planning is orthogonal to the concerns of static-world planning. To the extent that a dynamic-world planner must engage in static-world planning, the problems of complexity, search control, reasoning about action (for example, the frame problem), reasoning about optimality, etc., are still difficult. New problems arise concerning decisions about where, when, and how to allocate resources for static-world planning procedures within the overall activity of reactive agents, and additional problems arise due to the unique requirements imposed by complex dynamic domains.

On the other hand, <u>some</u> of the static-world planning problems <u>are</u> changed by the move to the framework of generating planned activity. In this section I will examine the effects of the change of framework on static-world planning problems, and I will also discuss new problems that have not yet been adequately addressed by dynamic-world planning systems.

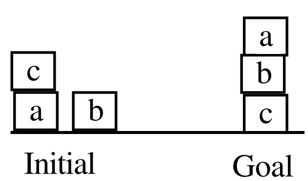

Figure 1. Sussman's Anomaly.

Consider the classic static-world planning problem, known as Sussman's

Anomaly, depicted in Figure 1. In the total-planning framework this example gives rise to "optimality" problems when subgoals are assumed to be independent. The goal is specified as the conjunction of (on **a b**) and (on **b c**), and early planning systems would attempt to completely solve one conjunct before beginning work on the other. If (on **a b**) is solved first (by putting **c** on the table on then putting **a** on **b**) then the new (on **a b**) stack would have to be immediately dismantled in order to solve (on **b c**). (Blocks may be picked up only one at a time.) On the other hand, if (on **b c**) is solved first (by putting **b** on **c**) then once again the new stack would have to be dismantled in order to achieve the other conjunct.

One total-planning solution to the Sussman Anomaly is to reason about action sequences as partial orders, allowing the primitive actions that achieve one subgoal to be interleaved with those that achieve another [Sacerdoti 1975]. While this sort of reasoning is necessary in many situations (for example, to avoid painting oneself into a corner [Sacerdoti 1975]), the character of the problem changes, and simpler solutions are sometimes available, in the framework of generating planned activity. In generating planned activity it is assumed, as part of the problem-solving framework, that there are capabilities for monitoring the world, and for reassessing the situation at run-time. This attention to the situation in which the problem-solver finds itself may in many cases take the place of extensive precomputation. In the case of the Sussman Anomaly, if the system decides initially to work on (on **a b**), then once **c** is removed from **a** the opportunity arises to notice that it is better to work on (on **b c**) for a few steps, and then to continue working on (on **a b**). This would generate the optimal behavioral sequence without the necessity of representing and reasoning about the partial order of plan-steps. In other words, if there is a decision procedure capable of noticing that (on **a b**) is the goal that should be worked on initially, and capable of noticing that (on **b c**) becomes more attractive once **c** is removed from **a**, then a simple “hill climbing” search will produce optimal behavior in cases such as the Sussman Anomaly. The power (and complexity) of nonlinear, partial-order planning could be reserved for more difficult cases.

McDermott suggested such an approach as early as 1978 and implemented it in his NASL system [McDermott 1978]. NASL was ahead of its time in several respects; for example, it suffered from the fact that certain technical problems (for example, the development of an inferential theory of time and events) had yet to be solved. More significantly, the value of NASL’s rejection of the total-planning framework was not recognized by the larger AI community until sobering complexity results had been proven, and until the gulf between blocks world and the real world had been more fully explored—events that took another decade to unfold.

The framework of generating planned activity also leads to a new set of problems, many of which are concerned with the complexities of control and communication within large dynamic systems. For example, Gat discusses the <u>Wesson Oil problem</u>:

> A more difficult problem arises when a high-priority task attempts to initiate an activity whose resources are currently in use by a low-priority task, a problem I call the Wesson Oil problem. The name derives from a television commercial for Wesson Oil in which a housewife is frying chicken (in Wesson Oil, of course) when one of her children suddenly falls down and has to be taken to the hospital. However, before going to the hospital the housewife turns off the stove. When she returns an hour later she resumes frying the chicken (which turns out crispy despite the fact that it has been soaking in the oil for an hour). [Gat 1991, 40]

Gat notes that the action of turning off the stove can not be accounted for by previous systems (Firby’s RAP system [Firby 1989] in particular), because the use of “clean-up procedures” requires a more complex control model than that usually used in purely reactive architectures. This is one example of a case in which the simple <u>combination</u> of reactive and deliberative processes provides insufficient <u>integration</u>. It is not enough for a reactive system to suspend execution of one plan (cooking) while running a higher priority plan (going to the hospital); it is also necessary to allow for reasoning procedures that plan for appropriate handling of the suspension and of the subsequent resumption of the lower priority plan.

The Wesson Oil problem points to a need for deliberative mediation of reactive processes, but it does not reveal the extent of the interdependence between reaction and deliberation. While it may be reasonable to suppose that a screaming child triggers a pre-existing high-priority reaction behavior, it is easy to devise problems for which no such reaction rules may be reasonably presumed to exist. The Open Window problem and the Ice Cube problem are two examples; they will be explained below, and a solution to the Ice Cube problem in the HomeBot system will be discussed in Section 9.3.3.

In the Open Window problem we suppose that a household robot notices an open window after having heard a prediction of rain on the radio. The combination of rain and open windows often leads to undesirable conditions (for example, a wet rug), and we may suppose that the robot has goals to prevent such conditions. But an open window is not always a problem, and hence the robot must have, at minimum, some means of turning reactions to open windows on and off. The conditional rule, "close if rain predicted, leave open otherwise," might initially appear to be adequate, but it won't suffice in the context of complicating circumstances. Windows covered by awnings need not be closed for simple rain showers, but should be closed for severe thunder storms or for hurricanes. In case of severe storms, "storm windows" and/or shutters should be closed as well. The robot might also consider the veracity of the radio announcer: wasn't that the announcer who predicted a Martian invasion last week? Perhaps the weather report was part of a documentary about Seattle, or a segment of a "classic broadcast" from the early days of radio. Circumstances can be envisioned that would complicate any "open window reaction rule" to an arbitrary degree, and a rule that checked for all such complications would be computationally expensive if constructible at all. The rule might require arbitrary access to knowledge and to reasoning resources, and might even call for action in the world (for example, a confirming telephone call to the weather bureau).

In the Ice Cube problem, we imagine a household-cleaning robot in the midst of a routine household chore. An ice cube is on the floor in the middle of a room, and at some point it is seen by the robot. As in the Open Window problem, the assumption of a built-in "ice cube reaction rule" is problematic, and we assume that no such rule exists. Hence we can expect no immediate response to the sighting of the cube. Assuming that the robot has sufficient causal knowledge to infer that the ice cube will melt over time, however, and assuming that it can also infer that puddles may be dangerous (since people tend to slip in them), it would be appropriate for the robot to eventually make such inferences, and to alter its behavior accordingly.[5] The robot should then suspend work on its current task, take care of the potential safety hazard, and then resume the original task with as little replanning as possible.

The complications raised by the Wesson Oil problem exist in the Ice Cube problem as well; for example, if the original task involved carrying a tray of fine china, the process of suspending the original task could involve significant planning. New difficulties are introduced, however, by the Open Window and Ice Cube problems: dynamic deliberation is required to infer that a reaction is even necessary.

Neither replanning systems nor purely reactive systems are capable of producing appropriate behavior in such cases. Replanning systems are driven by failures, but in the Open Window and Ice Cube problems no failures occur. The change in behavior after seeing the open window or the ice cube must be triggered by inferences, not by simple failure conditions that may be detected in the world. Purely reactive systems

[5]Of course, the ice cube may be hazardous before it melts as well; in any event we may presume that reasoning is required to infer the existence of a present or future hazard.

preclude, by definition, the use of reasoning processes that could infer the necessary causal connections.

Problems like the Open Window problem and the Ice Cube problem make it clear that reasoning and reaction must be integrated, and not just combined, in systems that are to produce appropriate behavior in complex, dynamic environments.[6] Approaches to such integration generally involve concurrency, and often divide computational components into "levels" in order to organize the complex systems that result (e.g., see [Hayes-Roth and Hayes-Roth 1979], [Gat 1991]). The notions of "level" that have been used in this context form a large and heterogeneous set, and the principles that underlie the choice of such levels have never been sufficiently articulated. In Chapter 3, I will explore the various uses of level-related concepts in planning and in control systems, and in Part II, I will develop a theory of levels tailored to problems of integrated behavior such as the Open Window and Ice Cube problems.

---

[6]The "banana peel problem" also demonstrated the need for integration; see, e.g., [Spector and Hendler 1991b].

Chapter 3

Abstraction in Planning

3.1 The Abstraction Kaleidoscope

Abstraction has played an important role in the design of planning systems at least since ABSTRIPS [Sacerdoti 1974], but the particular role that it has played has been the subject of considerable confusion. In fact, the term “abstraction” has been used for a wide array of concepts, some of which are related to one another as variations on a theme, and some of which are more distant relatives. Some uses of abstraction address issues of static-world planning, and some address issues that arise only in more general, dynamic problem-solving contexts. Similarly, some uses of abstraction are useful primarily within the total-planning framework, while some address issues that arise while generating planned activity. In the present section a survey is made of this kaleidoscope of topics as they pertain to AI planning systems. The two topics that are of particular interest for the purposes of this dissertation—reduced partition abstraction and partitioned control abstraction—are taken up in greater detail in the remainder of the chapter.

In 1960 Bunge wrote an article called “Levels: A Semantical Preliminary” in which he surveyed meanings of the word “level” and attempted to impose order on the tumult that he found. His disparaging assessment of the use of the term is still somewhat appropriate:

> As used in contemporary science and ontology, the term *level* is highly ambiguous. Most authors do not care for a definition or even for a distinct characterization of this word; as a result, one and the same name—‘theory of

> levels'—is applied to a variety of doctrines having different referents. Thus, whereas a neo-platonist has in mind links in the Chain of Being, a mechanist may refer just to degrees of complexity, and a biologist either to integrated wholes or to stages in evolution. No wonder that they should often misunderstand each other, if they speak of different things while designating them with one and the same word. [Bunge 1960, 396]

In 1968, at a symposium on "Hierarchical Structure in Nature and Artifact," a group of researchers from a wide range of disciplines followed up on some of the questions that had been raised by Bunge and by others. The proceedings of the symposium [Whyte, Wilson, and Wilson 1969] make for interesting reading on the breadth and the history of concepts of hierarchy. The term "hierarchy" is traced to Pseudo-Dionysius, Plato, and Aristotle [Whyte 1969, 8], and applied to problems from several fields of modern science. Although Bunge's complaints of ambiguity might still apply, several authors (including Bunge) provide new definitions and attempt to clarify the relations between various uses of level-related terms.

A similar multiplicity of concepts can be found in the literature of AI. The concept of hierarchy is ubiquitous, and various notions of abstraction and of levels have been applied to a range of AI problems. In recent years there has been a trend toward greater precision in the delineation of different subspecies of abstraction. Several of these subspecies can be seen as echoes of definitions provided at the 1968 symposium.

Wilkins provides an analysis of the various guises that "abstraction," "levels," and "hierarchy" have assumed in the planning literature [Wilkins 1988]. He uses the phrase "level of abstraction" to refer to levels that partition the planning domain by the <u>granularity</u> (fineness of detail) of the domain descriptions. He distinguishes levels of abstraction from <u>planning</u> <u>levels</u>, which are defined by the planning <u>process</u> and not by the structure of the domain knowledge. He describes a planning level as follows:

> Most planning systems have some central iterative loop that performs some computation on the plan during each iteration. This may involve applying schemas, axioms, or operators to each element of the existing plan to produce a more detailed plan. To the extent that such an iteration takes one well-defined plan and produces another well-defined plan, we call it a planning level. [Wilkins 1988, 47]

Abstraction levels and planning levels are clearly distinct concepts, but in some systems they may be combined, in which case coordination problems may arise [Wilkins 1988, 48–57]. For our purposes it is sufficient to note that there is an important distinction between "levels" of domain description and "levels" in the progress of some problem-solving procedure.

The "hierarchy" in so-called "hierarchical nonlinear planners" is usually just the subgoaling structure induced by the iterated application of planning operators; that is, hierarchical nonlinear planners generate hierarchies of "planning levels." Although Wilkins uses the term "level" in this context, such planning levels do not represent distinct, separable domains of reasoning. "Planning levels" refer to a notion of hierarchical decomposition, and not to a notion of partitioning, even though the term "level" might suggest otherwise.

Wilkins's "abstraction levels" do, however, refer to a partitioning of the domain of reasoning. I will refer to all notions of abstraction, hierarchy, etc., which imply such a partitioning as instances of <u>partitioning abstraction</u>.[7] The use of planning levels does not imply the use of partitioning abstraction, and neither is partitioning abstraction limited to Wilkins's "abstraction levels"; the use of partitioning abstraction does not imply that <u>granularity</u> is the criterion on which the partitioning is

[7] This definition of partitioning abstraction should not be confused with the <u>partitioned abstraction hierarchies</u> of [Knoblock, Tenenberg, and Yang 1991]. Knoblock's partitioned abstraction hierarchies are instances of partitioning abstraction, but whereas I use "partitioning abstraction" to refer to <u>any</u> system in which domain knowledge is partitioned into levels of abstraction, Knoblock uses "partitioned abstraction hierarchy" to refer only to ABSTRIPS-style systems in which criticalities (and hence partitions) are assigned to predicates independently of the operators in which they occur.

based. The concept of partitioning abstraction is particularly important when considering the application of multilevel architectures to problem-solvers that reason with abstraction. If the operative notion of abstraction is a type of partitioning abstraction, then it often makes sense to map each partition to a specific level of the multilevel architecture.

Wilkins also notes that meta-levels in some planning systems (e.g., MOLGEN [Stefik 1981]) have been referred to as "layers of control" which "model hierarchical planning." Meta-levels are used to allow for reasoning about the reasoning processes themselves; this is yet another notion of "level," distinct both from levels of abstraction and from Wilkins's "planning levels." Meta-reasoning can be useful in planning systems, however, and it may be of particular utility in contexts that demand reactivity. For example, it is possible for a system to reason about the time that the system's own reasoning is taking, and to make reasoning/reaction trade-offs on that basis (see [Georgeff and Ingrand 1989], [Kraus, Nirkhe, and Perlis 1990] or [Wilensky 1983]). Several researchers have proposed multilevel architectures, similar in many respects to the architecture described in Part III of this dissertation, in which each level reasons about the lower levels in the hierarchy (e.g., [Hayes-Roth 1985], [Hayes-Roth 1990], [Kuokka 1990], [Schoppers and Linden 1990]).

The utility of meta-reasoning for dynamic-world planning is clear from the scenario, discussed in [McDermott 1978] and [Kraus, Nirkhe, and Perlis 1990], in which Nell is tied to railroad tracks as a train approaches. Dudley's job is to rescue Nell, and if he is to be successful he must complete all of his planning and execution tasks before the train runs her down. The only way that a system can perform reliably in such circumstances is for it to account explicitly for its own reasoning time, and for it to choose between reasoning and execution options on that basis. However, these issues are mostly independent of the methods used for base-level (as opposed to meta-level) reasoning and execution, and in particular there is little connection between the notion of levels in a meta-level system and that in a system with multiple levels of abstraction. The motivation for the use of abstraction-partitioned systems generally springs from the complexity of representations of the world, and from the vast amounts of planning knowledge required for solving real-world problems. These are problems for any planning system, whether or not it has meta-reasoning capabilities. For this reason it makes sense to study levels of abstraction and meta-levels independently. It might also be the case that good planning algorithms will obviate, to some extent, the need for meta-reasoning in the first place.

Tenenberg also provides a discussion of the range of uses of abstraction in planning systems [Tenenberg 1991]. Although he makes "no attempt to provide a universal, encompassing definition," he does "formalize previously vague notions of abstraction" (p. 215). He details two species of abstraction called inheritance abstraction and relaxed model abstraction. Both of these techniques are instances of the Map, Plan, Inverse-Map (MPI) model of abstract problem-solving. Tenenberg describes this model as follows:

> Typically, an abstraction is taken as a mapping between representations, from a *concrete* level to an *abstract* level (or to several abstract levels through repeated mappings). The abstract level typically does not contain certain concrete-level details which expand the state-space, but which are usually unnecessary to consider in order to obtain an approximate solution. The search strategy is to abstract the operators and initial state of a system, problem solve at the abstract level, and use the abstract solution to guide the search back at the original level. [Tenenberg 1991, 214]

As is implied by this definition, the types of abstraction considered by Tenenberg are all simplification-based; that is, the representations at the higher levels of abstraction omit details and are therefore simplifications of their lower-level counterparts. I will refer to all such models of abstraction as types of simplification abstraction. Note that Wilkins's "abstraction levels" are defined by decreasing "granularity"; the standard interpretation of this criterion would be as a species of simplification abstraction. Tenenberg's inheritance abstraction exploits the inheritance mechanisms commonly used in representing object and action taxonomies in order to focus search in planning algorithms. The higher-level representations are in this case generalizations of the lower level representations, while the lower level representations are specializations of those at the higher levels. In many cases the lower level representations induce a richer, more complex search-space because they include the specifications of the higher levels (via inheritance) plus the additional information acquired in specialization. In these cases the MPI framework can be used, allowing "general" plans to guide the search for "specialized" plans. Tenenberg provides an analysis of the conditions under which this will be effective [Tenenberg 1991, 231–249]. Note that inheritance abstraction is not necessarily a type of partitioning abstraction; although partitions could be defined through the inheritance hierarchy, such partitions are not part of the basic framework.

Relaxed model abstraction is a variant of simplification abstraction that is more typical in planning systems. The essential idea is that abstract (higher) levels are generated from concrete (lower) levels by the removal of information of various sorts. While inheritance abstraction can also be viewed as removing "specializing" information at higher levels, relaxed model abstraction makes no claims that the simplified representations are more "general" than their lower level counterparts in the sense used in object and action taxonomies. The higher levels are usually generated from the lower levels by simple syntactic reductions of the constraints specified in problem solving operators. The most common form of syntactic reduction is the elimination of preconditions. This form of abstraction has been the subject of considerable recent interest and research; it forms a "spectrum of abstraction hierarchies" [Knoblock, Tenenberg, and Yang 1991] within the kaleidoscope of abstraction here under consideration. A more thorough examination of this work is provided in Section 3.2.

Christensen [Christensen 1991] has developed a variant of simplification abstraction that uses a technique called predicate relaxation, rather than precondition elimination, to generate abstract problems from lower-level problems. Predicate relaxation "defines a new predicate P^1_{rel} from a predicate P in such a way that P^1_{rel} holds in all states in which P holds and in all states in which P can be achieved by the application of one operator" (p. 12). Relaxed predicates are computed by regressing predicates through operators, and are used during planning at higher levels of abstraction. This provides a semantic, rather than syntactic, basis for simplification abstraction; in this respect predicate relaxation is similar to the semantic abstractions described by Plaisted for automatic theorem proving [Plaisted 1981, 53]. Christensen also describes techniques by which relaxed predicates can be used to endow planning systems with a limited form of reactivity.

A very different notion of abstraction and hierarchy has been developed in research on control systems, integrated intelligent architectures, and dynamic-world planning systems. Much of this work can be seen as following from Simon's observation that "Hierarchy ... is one of the central structural schemes that the architect of complexity uses" [Simon 1969, 87]. Simon's notion of hierarchy is based on the concept of "near decomposability," in which:

> Intracomponent linkages are generally stronger than intercomponent linkages. This fact has the effect of separating the high-frequency dynamics of a hierarchy—involving the internal structure of the components—from the low-frequency dynamics—involving interaction among components. [Simon 1969, 106]

Here Simon expresses a central idea in hierarchical control systems—that modularity is a valuable tool for organizing complex systems, particularly when the dynamics of the system (or, in the planning/control context, the reactivity of the system) is an issue. An important feature of this notion of hierarchy is that simplification does not necessarily play any part in the criterion used to partition the hierarchy. This notion of hierarchy is enormously important in the design of real-time control systems and hence in the design of dynamic-world planners. Roitblat makes the connection to notions of abstraction explicit:

> Standard single-layer control systems are severely limited in their ability to deal with the kinds of problems faced by autonomous robots. Psychological investigations suggest that organisms employ multi-level representations of their behavior that include abstract descriptions of the behaviors' goals and alternative means of achieving those goals. [Roitblat 1991, 449]

This form of abstraction will be discussed in greater detail in Section 3.3.

Psychological theories of modularity are numerous. Fodor, in The Modularity of Mind [Fodor 1983] restricts modularity to "input systems," but other researchers disagree, describing "central systems" as modular as well. Shallice writes that:

> Functional dissociation data suggest that not only are input systems organised modularly, but so are central systems. This conclusion is supported by findings on impairment of knowledge, visual attention, supervisory functions, memory, and consciousness. [Shallice 1991, 429]

A host of other modularity and level-based theses can be found in the psychological literature. The connection of such theses to ideas about abstraction has a long history:

> In Avicenna's (Ibn Sina, 980–1037) treatises on psychology, for example, there are various degrees of abstraction of forms which correspond to the ascending sequence of cognitive powers, the sensitive, the imaginative, the estimative, and finally the intellective. [Weinberg 1973, 3]

In more recent work, [Gardner 1983] proposes a "theory of multiple intelligences," [Jackendoff 1987] presents a detailed, level-based exposition of the “modularity of the computational mind,” and [Cermak and Craik 1979] present a collection of papers by various authors on theories of “levels of processing” in human memory. Grafman, on the basis of data from patients with frontal lobe lesions, suggests a theory of planning and action in which “managerial knowledge units” are arranged in an abstraction hierarchy [Grafman 1989]. Jaques presents a comprehensive psychological theory of levels of abstraction and relates it to theories ranging from developmental to organizational psychology [Jaques, Gibson, and Isaac 1978].

Jackendoff’s theory utilizes various notions of abstraction, and he makes explicit a <u>compositional</u> notion of abstraction that appears in several modular theories. This idea, that higher levels are <u>built</u> <u>up</u> out of lower level entities, contrasts with the idea of abstraction-as-simplification used in reduced partition abstraction:

> Let me be slightly more precise about what I mean by a level of representation: it consists of a structured repertoire of distinctions that can be encoded by the combinatorial organization of the computational mind. In the theories to be discussed here this structured repertoire is built up from a finite set of primitive distinctions, plus a finite set of principles of combination that make it possible to build primitives into larger information structures. [Jackendoff 1987, 47]

Gerard takes the compositional conception further, noting that in some hierarchies the whole is greater than the sum of its parts:

> Again, many of you will remember Eddington’s lovely statement, “We used to think if we knew one, we knew two, because one and one are two. We are finding out that we must learn a great deal more about ‘and’.” This is the combinatorial problem and the essence of hierarchies; they are more than mere assemblies of units. [Gerard 1969, 220]

The modularity theses from psychology can have an impact on the design of dynamic-world planning systems. Some of the theories (e.g., that of [Grafman 1989]) present explicit prescriptions for models of planning. Others, though not addressed to planning *per se,* provide insights into techniques that might be used to provide sought-after properties of planning systems. For example, the hypotheses of Jackendoff have direct bearing on the problems of integrated behavior that were discussed in Section 2.3:

> By breaking up processing into these components, we can achieve the observed interpenetration of top-down, bottom-up, and intralevel holistic effects, without resorting to a chaotic unstructured free-for-all in processing; what we have instead is a very tightly structured free-for-all! [Jackendoff 1987, 258]

Nilsson, in his Forward to Readings in Planning [Allen, Hendler, and Tate 1990], points out that a concept of hierarchy has been implicit in the idea of "planning" from the earliest days of AI planning research. Hints of abstraction and of hierarchy can even be found in dictionary definitions of "plan," such as, "A systematic arrangement of details; an outline or sketch: *the plan of a story"* [Morris 1978]. Hence work in planning is relevant to a discussion of abstraction, just as much of the work on abstraction (e.g., the hierarchical knowledge structures of [Schank 1982]) is directly applicable to planning. As it would be impractical to follow all of these connections, the remainder of this chapter details only the two notions of abstraction that appear to be most important for dynamic-world planning. The next section examines reduced partition abstraction, which is the most prevalent form of abstraction in the static-world total-planning framework. The following section examines partitioned control abstraction, which captures important level-based ideas from hierarchical control theory and robotics.

3.2 Reduced Partition Abstraction

In the previous section, I defined partitioning abstraction as a type of abstraction in which the problem-solving domain is partitioned into distinct segments, and simplification abstraction as a type of abstraction in which abstract representations are merely simplified versions or their lower-level counterparts. In addition, I noted that relaxed-model abstraction has been used to refer to instances of simplification abstraction in which simplification is achieved by nothing more than the removal of information. I will refer to the variant of simplification abstraction which is both partitioned and based on simplification-by-information-removal as reduced partition abstraction.[8] Simplification abstraction as has been studied extensively, and has proven useful in areas outside of planning, for example in automatic theorem proving [Plaisted 1981]. Within the planning literature reduced partition abstraction has been the dominant form of simplification abstraction. Reduced partition abstraction is often referred to as ABSTRIPS-style abstraction since ABSTRIPS and its descendants are all variants of reduced partition abstraction. Several recent papers have contributed significantly to the development and formalization of the theory (e.g., [Bacchus and Yang 1991], [Elkan 1990], [Knoblock 1991a], [Knoblock 1991b], [Knoblock, Tenenberg, and Yang 1991]). In this section I outline the basic ideas of this work and provide an example, taken from [Knoblock, Tenenberg, and Yang 1991], that shows reduced partition abstraction in action. I will return to this example in Chapter 6 where I show that reduced partition abstraction is a special case of a generalized form of abstraction called supervenience.

Sacerdoti's ABSTRIPS (Abstraction-Based STRIPS) system [Sacerdoti 1974] introduced reduced partition abstraction to AI planning systems. ABSTRIPS is an extension of STRIPS, and is based on the same operator formalism and representational conventions as is STRIPS. A STRIPS operator is specified as a list of

[8] A distinction has been made in the literature between relaxed model abstraction, considered to occur when only preconditions are removed, and reduced model abstraction, considered to occur when other removals (e.g., from the state descriptions) are allowed as well [Knoblock 1991a, 26–28]. The distinction is not relevant to the purposes of this dissertation. The phrase "reduced partition abstraction" is used in favor of "relaxed partition abstraction" because it more clearly expresses the idea of information-removal.

preconditions, an add list, and a delete list. The preconditions specify the set of formulae that must be true in order for the operator to be applicable, the add list specifies the set of formulae that will be made true by the application of the operator, and the delete list specifies the set of formulae that will be made false by the application of the operator. As with STRIPS, the system operates by applying operators to "reduce differences" between specified initial and goal states. A recursive algorithm can be employed to back-chain on preconditions not satisfied in the initial state and to forward-chain on residual differences not reduced by the chosen operator. Much of the research in the total-planning framework has focused on the improvement of this algorithm (e.g., [Sacerdoti 1975], [Tate 1977], and [Chapman 1987]).

ABSTRIPS differs from STRIPS in that an ABSTRIPS system defines a set of reduced search spaces, formed from the original search space by the removal of preconditions from the operators. Each literal in the domain is assigned an integer "criticality" value, and operators at abstraction-level *n* are formed by removing all preconditions containing literals with criticality less than *n*. This allows for a search strategy, called "length-first" search in [Sacerdoti 1974], in which high-level "skeletal" plans are generated first, to be refined at lower levels by the reintroduction of the eliminated preconditions.

The use of abstraction in ABSTRIPS has had a significant impact on planning research, and several researchers have developed and extended the idea (see, e.g., [Tenenberg 1987], [Wilkins 1988], [Yang and Tenenberg 1990], and [Christensen 1991]). Knoblock has recently provided an analysis of planners of this type, and has shown that a "spectrum" of abstraction hierarchies can be constructed on the basis of the types of constraints placed on the assignment of criticalities to operator preconditions [Knoblock, Tenenberg, and Yang 1991]. In addition, he shows that his analysis is useful in designing systems to automatically generate "good" abstraction hierarchies for ABSTRIPS-style problem solvers [Knoblock 1991a]. In the course of his analyses Knoblock presents the reduced partition abstraction approach to the classic Towers of Hanoi problem. This problem will serve as an exemplar of the kind of abstraction used in reduced partition abstraction systems.

The Towers of Hanoi domain consists of 3 pegs and some number of disks of varying sizes (with no two disks being the same size) that can be stacked on the pegs. I will consider only the 3-disk version of the problem, though the discussion can be easily generalized to *n* disks. States in this domain consist of descriptions of the positions of all of the disks, and problems are specified as pairs of initial and goal states. Operators are provided for moving any disk that is on the top of a stack to any other peg, provided that a larger disk is never placed on a smaller disk. Following [Knoblock, Tenenberg, and Yang 1991], I label the three pegs P_1, P_2, and P_3, I label the disks **Large**, **Medium**, and **Small**, and I represent the locations of the disks using the literals **OnLarge**(x), **OnMedium**(x) and **OnSmall**(x). Using negated literals to express the delete list of a STRIPS operator, the following operators can be used for solving problems in the 3-disk Towers of Hanoi domain:

Operator: **MoveL**(x,y)
Preconditions: ¬ **OnSmall**(x), ¬ **OnSmall**(y),
¬ **OnMedium**(x), ¬ **OnMedium**(y), **OnLarge**(x)
Effects: ¬ **OnLarge**(x), **OnLarge**(y)

Operator: **MoveM**(x,y)
Preconditions: ¬ **OnSmall**(x), ¬ **OnSmall**(y), **OnMedium**(x)
Effects: ¬ **OnMedium**(x), **OnMedium**(y)

Operator: **MoveS**(x,y)
Preconditions: **OnSmall**(x)
Effects: ¬ **OnSmall**(x), **OnSmall**(y)

A reasonable reduced partition abstraction hierarchy can be formed by assigning a criticality of 2 to **OnLarge**, a criticality of 1 to **OnMedium**, and a criticality of 0 to **OnSmall**. A plan at the highest level is generated by removing all preconditions with criticality less than 2 and invoking a standard planning algorithm. Supposing that all three disks are initially stacked on peg P_1 and that the goal is to stack them all on peg P_3, one resulting plan would be:

<**MoveL**(P_1,P_3), **MoveM**(P_1,P_3), **MoveS**(P_1,P_3)>

This plan must be refined when considered at level 1; for example, the reintroduced precondition ¬ **OnMedium**(x) is not true in the initial state. Using the techniques described in [Knoblock, Tenenberg, and Yang 1991] and elsewhere, the plan can be refined first to:

<**MoveM**(P_1,P_2), **MoveL**(P_1,P_3), **MoveM**(P_2,P_1), **MoveM**(P_1,P_3), **MoveS**(P_1,P_3)>

at level 1 and then to a complete plan at level 0. The details of the algorithms that accomplish this refinement, and the properties of abstractions that allow such algorithms to work efficiently, are reviewed in [Knoblock, Tenenberg, and Yang 1991] and will not be discussed further in this dissertation. The essential point is that reduced partition planners are able to improve the efficiency of planning by solving simplified versions of planning problems, obtained by simple syntactic manipulation of the planning operators, and by subsequently using the solutions of the simplified problems to guide search on the full problem. The technique of simplification by precondition elimination has been well studied, and can be extended to an arbitrary number of levels of abstraction. The extent of the efficiency gains is significant; [Knoblock 1991b] shows that this technique can reduce the size of the search space from exponential to near linear in the solution length.

3.3 Partitioned Control Abstraction

The principal use of abstraction in planning has been the application of reduced partition abstraction to static-world domains in the total-planning framework. Other fields closely related to planning, however, have long utilized a different concept of abstraction to address problems that arise in dynamic, uncertain domains. This type of abstraction, which I will call <u>partitioned</u> <u>control</u> <u>abstraction</u>, is defined in terms of the manner in which control is distributed throughout a system, rather than in terms of the knowledge structures or search spaces at each level. In this section I will discuss partitioned control abstraction, distinguish it from reduced partition abstraction, and show how partitioned control abstraction allows systems to perform well in complex, dynamic worlds. In addition, I will review some of the dynamic-world planning systems mentioned in Chapter 2 vis-à-vis their status as instances of partitioned control abstraction.

Churchland and Sejnowski [Churchland and Sejnowski 1988] discuss the notion of “level” in cognitive science, and expand on the well-known division of cognition into computational, algorithmic, and implementational levels due to Marr [Marr 1982]. The computational level consists of the abstract structure of problems and of problem-solving search-spaces, independent of the methods that may be used to search them. The algorithmic level consists of particular procedures, data-structures, etc., that may be used in problem-solving, while the implementational level consists of the physical artifacts that execute the algorithms. Churchland and Sejnowski note that in the brain and in reasonable models of cognition “there are many levels of implementation and . . . the notion of *the* algorithmic level is as oversimplified as the notion of *the* implementation level” (p. 742). They paint a picture in which there are many levels, each of which is in some sense computational, algorithmic, and implementational.

A similar view has become popular in dynamic-world planning research and is implicit in hierarchical control theory and other approaches to the design of real-time robotic systems. The concept of levels that emerges from such a view is a form of partitioning abstraction in which each level contains not only a set of representations and perhaps procedures, but also full-fledged level-specific problem-solving systems. It is such systems that I will refer to as instances of partitioned control abstraction.

In 1969 Mesarovic and Macko, who have been credited as founders of hierarchical control theory,[9] listed five general characteristics of "stratified" systems. Three of their characteristics are particularly relevant:

> 3) There exists an asymmetrical interdependence between functioning of a system on different strata. . . .
> 4) Each stratum has its own set of terms, concepts and principles and what is considered as a system and its objects are different on each stratum. Furthermore, there is a hierarchy of objects and languages in which they are described. . . .
> 5) Starting from any given stratum, understanding of a system increases by crossing strata: moving down the hierarchy one obtains a more detailed explanation while moving up the hierarchy one obtains a deeper understanding of its significance. [Mesarovic and Macko 1969, 33–35]

Characteristic #5 also applies to reduced partition abstraction, except that in that case the "deeper understanding of its significance" may not be as profound as the language of Mesarovic and Macko suggests. Characteristic #3 describes an important feature of most multilevel systems; in fact, a notion of asymmetrical interdependence forms the core of the theory of supervenience that will be presented in Part II of this dissertation. Characteristic #4 contributes significantly to the attractiveness of hierarchical control systems, and I will consider it to be part of the concept of partitioned control abstraction even though the connection to the concept of control is

[9] In the Preface to [Findeisen et al. 1980].

not explicit. The ability to use different languages within each module often emerges naturally when a system is partitioned into modules with independent control.

The distinction between control partitioned abstraction and partitioned abstraction *simpliciter* is not always crisp; for example, some ABSTRIPS-style systems could be viewed as either control partitioned or not, depending on their implementations. The distinction is useful nonetheless, as it serves to distinguish systems that are clearly not control partitioned, such as CAKE (which has 8 layers which must all cooperate: the truth-maintenance layer, the equality layer, the demon layer, the algebraic layer, the types layer, the plan calculus layer, the plan recognition layer, and the plan synthesis layer [Rich 1985]) from systems that clearly are control partitioned, such as the blackboard system in [Hayes-Roth and Hayes-Roth 1979].

Hanks and Firby discuss general issues in the design of dynamic-world planning architectures and argue explicitly for systems that use partitioned control abstraction (which they refer to as "layered architectures" or as "partitioned architectures") [Hanks and Firby 1990]. They claim that "different reasoning processes are appropriate for acting and deliberation because action is driven by urgency while deliberation needs time to consider alternatives and consequences" (p. 68). They also note that partitioned architectures face communication problems that don't arise in "uniform" architectures, and they propose to handle such problems by employing a single plan representation at both levels of their 2-level system. They note that in other layered architectures such as the Entropy Reduction Engine [Bresina and Drummond 1990] limitations on the communication between layers restrict the flexibility of systems in uncertain domains.

The suggestions of Hanks and Firby have a long history in the literature of robotics and control theory, as well as in the literature of planning proper. Hierarchical control systems based upon similar principles have been put to a wide range of uses, ranging from the control of steel plants [Williams 1985] to the control of autonomous vehicles (e.g., [Payton, Rosenblatt, and Keirsey 1990]). Mitchell, Payton, and Keirsey describe their architecture in terms similar to those used in the dynamic-world planning literature:

> The architecture is structured such that lower-level modules perform tasks requiring greatest immediacy, while higher-level modules perform tasks involving greater assimilation of sensor data, making use of large amounts of *a priori* knowledge. [Mitchell, Payton, and Keirsey 1987, 129]
>
> Each specialized layer processes data in a manner specific to its own operational goals, and issues control commands accordingly. Although higher level layers are expected to exert influence upon lower level layers, the lower layers are free to exert control over the vehicle without waiting for assimilation to be completed by the upper levels. [Mitchell, Payton, and Keirsey 1987, 137]

Perhaps the most ambitious control partitioned architectures have been those developed by Albus and his associates at the National Institute of Standards and Technology (formerly the National Bureau of Standards). In a paper entitled “Outline for a Theory of Intelligence” Albus synthesizes much of this work into a hierarchical, control partitioned “general theoretical model of intelligence.” Partitioned control abstraction is a central feature of the model, as evidenced in the abstract:

> At each level, functional modules perform behavior generation (task decomposition planning and execution), world modeling, sensory processing, and value judgement. Sensory feedback control loops are closed at every level. [Albus 1991, 473]

Knowledge-rich architectures such as Albus’s (see also [Meystel 1987] and [Arkin 1990]) are often distinguished from the <u>subsumption</u> <u>architecture</u> developed by Brooks, in which world modeling is avoided entirely, and in which all levels have access to sensor input and actuator control [Brooks 1991]. However, the principles of control abstraction are utilized by all of these systems, regardless of differences in philosophy on other points. Arkin has also argued that the opposition between the two camps are artificial; using “reactive” and “hierarchical” to refer to the knowledge-poor and knowledge-rich control architectures respectively, he writes, “The illusion that reactive and hierarchical planning methods are at odds with each other needs to be dropped” [Arkin 1989]. All of these systems partition their active elements into control modules, arranged into abstraction hierarchies, in order to organize the flow of information and to allow for flexible response to events in dynamic environments.

Several dynamic-world planning systems have been partitioned on the premise that lower levels can be engineered to use faster, simpler algorithms to obtain improved reactivity. Firby [Firby 1989] describes a three level architecture in which the lowest level performs robot control, the central level is a reactive executor called the RAP system, and the highest level is a more traditional planner. The central level uses precomputed “reactive action packages” that are run by a fairly simple execution algorithm. The highest level, which is responsible for predicting future events, creating “sketchy plans” and monitoring execution, would presumably use more complex algorithms. Firby’s system has been influential, and several other systems use similar partitioning schemes (e.g., [Durfee 1990], [McDermott 1990]).

Hayes-Roth and Hayes-Roth, as early as 1979, proposed a planning model that utilizes a robust form of partitioned control abstraction [Hayes-Roth and Hayes-Roth 1979]. Their blackboard-based model, motivated largely by psychological evidence regarding human plan construction, plans at multiple levels simultaneously in order to model “opportunistic” reasoning processes. Although their planning task is not performed in a dynamic domain, they are attempting to model the dynamics of the planning process itself, and hence similar considerations apply. Later work by Hayes-Roth combines reduced partition abstraction with partitioned control abstraction in a more dynamic domain [Washington and Hayes-Roth 1990]. Once again, levels of control are used to partition the activity of a large system, providing for flexibility in the face of a complex domain.

Further examples of partitioned control abstraction are prevalent, both in the planning literature and in the literature of hierarchical control theory. Gat alludes to the ubiquity of the concept in his summary of “Other Architectures”:

> There is a vast array of other robot control architectures in the literature. Nearly all of them are variants on the traditional sense-plan-act architecture where a planner constructs a plan from a world model to be executed by an execution system. There are innumerable variations on this theme. The most common is some sort of hierarchical generalization of the basic approach, where one planner generates a plan at a high level of abstraction which gets fed to another planner which fills in the details (e.g. NASREM [Smith89]).
>
> ---
>
> [Smith89] D. B. Smith and J. R. Matijevic, “A System Architecture for a Planetary Rover,” *Proceedings of the NASA Conference on Space Telerobotics,* vol. 1, JPL Publication 89-7 California Institute of Technology Jet Propulsion Laboratory, 1989.

In the next Section I will compare control partitioned abstraction, the dominant form of abstraction in control and reactive systems, with reduced partition abstraction, the most thoroughly investigated form of abstraction in the planning literature. It will turn out that the two types of abstraction are compatible, and that their synthesis forms an important concept for the design of systems that must function in complex, dynamic domains.

3.4 Reduced Partitions and Partitioned Control

Reduced partition abstraction systems are characterized by the relation between knowledge structures at different levels; higher levels use “simpler” representations, and the particular kind of simplification that is employed differentiates between variants of the framework. Partitioned control systems are characterized by the relation between <u>processes</u> at different levels, and hence they may be compared with one another according to criteria such as the following:

1) How are the levels related with respect to the problems that they solve?

2) How are the levels related with respect to the algorithms used?

3) How are the levels integrated into a single physical/computational system?

These criteria also allow for the comparison of partitioned control abstraction with reduced partition abstraction. Reduced partition systems might provide the following answers to these questions:

1) Higher-level problems are simplified versions of the lower level problems.

2) All levels use the same basic planning algorithm.

3) The levels are run one at a time, from most to least abstract, with possible backtracking to higher levels.

Control partitioned systems may provide broader answers to all three questions; levels need not be related by simplification, variations in algorithm are acceptable, and concurrency and flexible interaction between levels are desirable. This characterization leads to the view that partitioned control abstraction is simply a generalization of reduced partition abstraction. Indeed, some partitioned control systems explicitly call for the use of simplification in generating higher-level representations (e.g., see [Albus 1991, 481]). Hayes-Roth and Hayes-Roth also suggest such a view when they compare the "opportunistic" planning that their system models to the "top-down" planning methods used in systems such as NOAH [Sacerdoti 1975], [Sacerdoti 1977]. They suggest that "One resolution of the apparent conflict between the two models would simply incorporate the top-down model as a special case of the opportunistic model" [Hayes-Roth and Hayes-Roth 1979, 307].

On the other hand, the problems that reduced partition abstraction was developed to solve (complex static search problems) are qualitatively different from those (principally problems of dynamics) that motivated the development of partitioned control abstraction. As a result, the methodologies of the two types of abstraction are sufficiently dissimilar that the statement that one type is <u>simply</u> a generalization of the other is not appropriate. A system utilizing reduced partition abstraction might, in addition, partition control; but this says nothing about the nature of the partitioning schemes, the reasons that the two kinds of partitioning are helpful, or the reasons that the two partitioning schemes may or may not coincide. Previous dynamic-world planning systems have married the two types of abstraction (e.g., [Washington and Hayes-Roth 1990]), but little has been written to support the hypothesis that the marriage will be a happy one.

Nonetheless, there is an important sense in which the two types of abstraction can be unified. Reduced partition abstraction was developed to reduce large search spaces by removing (temporarily) insignificant details, while partitioned control abstraction was developed to organize large processes by ignoring (delegating) routine tasks. Both of these frameworks benefit by abstracting <u>away</u> from the world; that is, by reasoning in some idealized (though not always simpler) model of the real world. The idealized model of the world is in both cases <u>determined</u> by the more realistic, lower level model, but for various reasons it is advantageous to form and to compute within the idealized models. These notions of abstraction as distance from the world, and of the "determination" relation between levels of abstraction, allow for a more principled unification of reduced partition abstraction and partitioned control abstraction; they will be expanded upon and formalized in the subsequent chapters.

Part II

Supervenient Levels

Chapter 4

Supervenience

> First there will be the usual crabby complaints from philosophers that we are misusing phrases like “theory of knowledge” and “epistemology”, and that our discussions have nothing whatever to do with what has been meant by those phrases. . . .
>
> . . . Fortunately, we need not detain ourselves with that sort of argument because, as luck would have it, no one in AI will understand it anyway.
>
> Warwick Yolks
> [From “There’s Always Room at the Top, or How Frames Gave My Life Meaning,” in SIGART Newsletter, August 1975.]

In the preceding chapters, I discussed the trend toward multilevel architectures for the integration of reasoning and action, and the use of multilevel representation systems in AI planning systems. In this chapter I provide an intuitive characterization of a variety of partitioning abstraction—called “supervenience”—that ties together various notions of partitioning abstraction in the literature. In particular, supervenience can be seen as a generalization of reduced partition abstraction, and as a special case of partitioned control abstraction. In the subsequent chapter, I will provide a formal definition for supervenience, and will further relate supervenience to other forms of abstraction.

The principal goal of this dissertation is to examine and to refine the notion of “knowledge level” that is most consequential in the design of intelligent reactive agents. We have seen that certain specialized notions of knowledge abstraction, or of knowledge layering, have benefits for search-space reduction and/or for flexible control. I will argue that the most important of these notions of layering and of abstraction are manifestations of a more general principle—that representations at the lower levels are epistemologically “closer to the world,” while representations at higher levels in some sense “depend” on those at lower levels. This dependence needs not be so strong as to imply that the higher level knowledge structures are merely simplifications of those at lower levels. While the lowest level of an ABSTRIPS system in fact contains the knowledge of the entire system, the lowest level of a multilevel control architecture might be the most knowledge-poor level of the system. Nonetheless, in both cases the higher levels “depend” on the lower levels. On the other hand, a more intimate dependency relation can be specified than is mandated by the structure of multilevel control systems. Such systems generally rely on a criterion of modularity holding between levels, but no dependency relation between knowledge structures is enforced by the architectures. I shall argue that such a relation forms the core idea of abstraction, and of “knowledge levels,” that is most useful for dynamic-world planning. I will adopt a term from philosophy—supervenience—to label this relation. The discussions of supervenience in the philosophic literature provide important insight into the supervenience relation, even though the philosophers are generally concerned with matters quite distant from dynamic-world planning, and even though they do not speak with a single voice (far from it). The idea of a non-reductive (i.e., the levels are distinct) dependency relation between domains of discourse involves certain subtleties and leads to certain formal structures whether the subject under consideration is ethics or robotics (or both).

The concept of supervenience originated in the 1950’s in discussions of the objectivity of moral laws. In 1983 a conference was held at Memphis State University on the topic “The Concept of Supervenience in Contemporary Philosophy,” and papers from the conference were published as a supplement to The Southern Journal of Philosophy [Horgan 1984]. In his contribution to the supplement, Gillespie provides the following history of the concept:

> When G.E. Moore introduced the concept of supervenience, he was attempting to explicate what he took to be the relationship between the descriptive and the evaluative properties of individual acts, objects, etc. Having argued that the two cannot be identical, he nonetheless believed the latter to be objective, and was searching for a way to explain the nonidentical, objective relation between descriptive and evaluative properties (or truths). The concept of supervenience has recently been employed in the philosophy of mind to characterize the relation . . . between mental and physical properties; and some materialists have proposed generalizing this approach to all phenomena so that, in their words, "the (micro) physical facts determine all the facts," or as it is sometimes put, "no change without physical change."[3] [Gillespie 1984, 111]
>
> [3] Geoffrey Hellman and Frank Wilson Thompson in "Physicalist Materialism," *Nous* XI (1977), pp. 309–345, and Terrance Horgan in "Supervenience and Microphysics," *Pacific Philosophical Quarterly* 63 (1982), pp. 29–43, develop the basic principles for this conception of materialism and (micro)physical determination.

The 1983 conference brought forward a number of new uses for the concept of supervenience, ranging from aesthetics to an analysis of public moods [Levinson 1984]. Additionally, several of the participants provided formal-logical accounts of supervenience, or critiques of formal accounts that had been provided by others.[10] The various accounts, formal and otherwise, diverged from one another considerably, resulting in some measure of controversy and perhaps even cynicism:

> 'Supervenience' seems over the years to have become an accordion-word: indefinitely stretchable, covering a bewildering variety of ideas related perhaps only by family-resemblance. [Post 1984, 163]

Subsequently, and possibly due to this embarrassment of riches, the concept seems to have fallen into even greater disfavor. In 1990 we find the following characterization:

[10]More recent formal analyses are also available; see, for example, [Blackburn 1988].

> In 1952 Hare, in *The Language of Morals*, did use and explain the term 'supervenience' as a philosophical term of art. He has acknowledged that he did not originate the use, but was only following a use already familiar at Oxford. (Indeed, I am sure he would not want to be responsible for introducing the term, given the morass of confusion it has produced.) [Klagge 1990]

The controversy continues,[11] but the concept has its uses nonetheless. Several authors have seized upon "supervenience" as the most appropriate term for some particular relation under study, and we need not be deterred by the fact that all of the relations so described are not strictly identical. Within the literature of AI the term is not well known, but John Pollock makes use of it in his How to Build a Person, where he provides the following intuitive example:

> Perhaps the simplest example is that of a statue and the lump of clay from which it is fashioned. It is not wholly inaccurate to say that the statue *is* the lump of clay formed into a certain shape, but there is a simple argument that shows quite conclusively that the statue is not the same thing as the lump of clay. . . . Thus the statue and the lump of clay do not share all the same properties and hence must be different objects. Still, it is obvious that there is an intimate connection between them. In some sense, the statue *consists of* the lump of clay, or as I will say, it *supervenes* on the lump of clay. [Pollock 1989, 32]

He makes this more precise by stating that, "A supervenient object is one that is composed of another object or group of objects without being identical with that object or group of objects" [Pollock 1989, 34]. He distinguishes this sort of "object supervenience" from the more primitive "property supervenience," for which he also provides a more formal account (which I will not adopt, but see Section 5.3). An important aspect of Pollock's definition is that he has generalized the notion of supervenience from a two-level notion (moral/physical, mental/physical, aesthetic/physical, etc.) to a multilevel notion. Object supervenience, for example, leads to the observation that "Physical objects can be arranged into a multi-level

[11]See, for example, [Miller 1990] and [Hellman 1992].

hierarchy, with the objects higher in the hierarchy being composed of objects lower in the hierarchy" [Pollock 1989, 33]. Several of the formal accounts in [Horgan 1984] are similarly extensible to multilevel systems, but Pollock provides intuitive examples that make the multilevel nature of supervenience more conspicuous.

Lennon expresses the supervenience of one (higher-level) body of facts upon another (lower-level) body of facts by saying that the lower-level facts are the facts "in virtue of which" the higher-level facts are true [Lennon 1990, 106].[12] This very general notion of supervenience leads to cases of supervenience well outside of the scope of the earlier two-level notions:

> (a) 'The dish is fragile' may be true in virtue of the dish being made of glass, or of its being made of china;
> (b) 'A signalled' may be true in virtue of A putting her arm out of the window of her car (given certain conventions governing signalling);
> (c) 'Britain entered the war' may be true in virtue of Parliament making a declaration, armies being mobilised, given certain directives, etc. [Lennon 1990, 107]

This characterization is interesting in part because Lennon's examples are almost identical to examples used by other writers as instances of "level generation." Level generation was devised by Goldman [Goldman 1970] to formalize the logical underpinnings of the "by" locution, used in sentence such as "A signalled by putting her hand out the window." Goldman's notion has been imported into AI in work on event representation by Allen [Allen 1984], and formalized in work on discourse processing [Pollack 1986]. Hence it is likely that fruitful comparisons could be made between work on level generation and supervenience, although such comparisons are outside the scope of this dissertation. Note that such a characterization says nothing about the higher levels being simpler than the lower levels—the lower levels are lower not because they are more detailed but because they correspond more obviously

[12] Lennon credits previous authors for this phrase.

to the world as physically described.

Lennon's discussion is also interesting because it deals explicitly with the factors that make supervenience, in its broader formulations, so intuitively appealing. Lennon is concerned with issues involving the explanation of human action. Irrespective of a particular philosopher's willingness to reduce higher to lower levels (for example, the mental to the physical), it is clear that the higher, supervening levels of description are useful in providing explanations of human actions in ways that the lower levels are not. For example, if we are interested in explaining how A signalled a turn, an answer in terms of signalling conventions and A's gross body motions ("she opened the window and extended her arm") would be appropriate. An answer in terms of lower-level physical primitives ("the molecules in the car moved according to the formula...") might describe the same event,[13] but the explanatory power has been lost by moving to the lower level of description. This captures something of the motivation within many fields for using multiple levels of description. Certain facts are simply easier or more natural to describe using high level descriptive terms. In many cases the "laws" of a system, or in planning applications we might say the "operators" of a system, can only be reasonably expressed in such a high-level language. Simon observes that "In the face of complexity, an in-principle reductionist may be at the same time a pragmatic holist" [Simon 1969, 86], implying that a hierarchical description can be useful even when the higher levels could "in principle" be reduced to lower level descriptions. From this pragmatic perspective, the question of whether or not the high-level descriptions could in some sense be reduced to the lowest level is not the issue; the issue is whether it is useful to have distinct levels. When the difference in utility is significant, the distinction between "in

[13]The question of how many events take place in such a situation, and of how they are to be individuated, is a major issue in the philosophical literature of events and actions, of which [Goldman 1970] is a part. See [Davis 1979] for contrasting opinions and further references.

principle" anti-reductionism and "pragmatic" anti-reductionism loses its force. Lennon concludes that:

> ... if we were to abandon our psychological mode of classification we would both lose a way of grouping together states, which in terms of our physical theory are distinct, and lose a way of capturing law-like generalisations which transcend those expressible in purely physical vocabulary. [Lennon 1990, 122]

Resistance to "reductionism" on the basis of the greater explanatory power of higher-level descriptions is also evidenced by Steiner's opposition to the Quinean reduction of arithmetic to set theory [Steiner 1979]. Steiner shows that explanatory mathematical proofs perform their explanatory function by exploiting analogies in the domain of high-level algebraic and geometric concepts. Mathematical entities fall into "natural kinds" that disintegrate upon reduction to their set theoretic bases. Hence while a geometric proof may in some sense be "reduced" to set theory, it will in the process lose its explanatory force. Therefore the geometric domain has some sort of autonomy, even though it "depends" on set theory. Although he doesn't use the term, we might say that Steiner is asserting that geometry <u>supervenes</u> on set theory.[14]

In a more explicitly computational framework, Marr also motivates the adoption of multilevel systems with a discussion of the utility (or necessity) of high-level languages in forming explanations:

> If one hopes to achieve a full understanding of a system as complicated as a nervous system, a developing embryo, a set of metabolic pathways, a bottle of gas, or even a large computer program, then one must be prepared to contemplate different kinds of explanation at different levels of description that are linked, at least in principle, into a cohesive whole, even if linking the levels in complete detail is impractical. [Marr 1982, 20].
>
> The levels idea is crucial, and perception cannot be understood without

[14] Further discussion of the relation between supervenience and explanation may be found in [Sayre-McCord 1988].

> it—never by thinking just about synaptic vesicles or about neurons and axons, just as flight cannot be understood by studying only feathers. Aerodynamics provides the context in which to properly understand feathers. Another key point is that explanations of a given phenomenon must be sought at the appropriate level. It's no use, for example, trying to understand the fast Fourier transform in terms of transistors as it runs on an IBM 370. There's just no point—it's too difficult. [Marr 1982, 336–337]

Marr does not speak of supervenience *per se,* but his call for multiple levels of representation is similar in spirit to the arguments of several proponents of supervenience. The actual levels that he proposes (computational, representational, and implementational [Marr 1982]) are, on the other hand, of a very different flavor than the levels proposed, for example, by the ethical theorists. But Marr is not the only aberration; even the level systems suggested by those who explicitly use the term "supervenience" form a strikingly non-uniform set. For example, the levels of physical composition suggested by Pollack are certainly not identical to the levels of explanatory power suggested by Lennon. What then are we to make of this array of "supervenience" concepts? I will not try to formulate a universal definition that subsumes all previous definitions of supervenience, if only because it is not quite clear how to spell out the previous definitions themselves. (However, see [Teller 1984] for a valiant effort at just that.) I will try, however, to sum up the essential features of the accounts, and to reformulate these features in a manner that will turn out to be useful for dynamic-world planning.[15]

All of the accounts of supervenience are concerned with a relation between two or more domains of discourse, neither of which is reducible to the other. The operative notion of "reducible" varies, but each account is motivated by a conviction that the domains of discourse ought to be (or must be) kept distinct from one another. In many cases the supervenient levels all "represent the same things," or even "do the same

[15]Chapters 5 and 10 include further, albeit brief, remarks on the relations between various forms of supervenience.

things," although again, the senses in which these terms might apply vary from theory to theory. Each account expresses a need for multiple levels of description, and each account allows individual levels to apply level-specific representations and methodologies to a problem at hand.

The relation between levels is not arbitrary. Rather, a critical idea shared by all accounts of supervenience is that the set of levels is ordered by an asymmetric dependency relation—that the higher levels depend on the lower levels or equivalently that the lower levels determine the higher levels. This dependency relation has been expressed in various ways, but each account specifies some notion of dependence that makes supervenience an asymmetric relation.

Another reading of the dependence/determination relation that accords with most of the previous definitions is that the lower levels are "closer to the facts," or "closer to the world." We can flesh this out a bit by saying that the lower levels "know more about the facts" that are expressible at both levels. This is the notion of supervenience that will be used in the remainder of this dissertation. As an example, if level L_1 and level L_2 can both express A, and if L_2 supervenes on L_1, then L_1 "knows more about" A than does L_2; that is, if L_1 says A is false, and L_2 says that A is true, then L_1 is right and L_2 is wrong—A is false. This is just what it means for L_1 to determine the facts at L_2, or for L_2 to depend on L_1, etc.[16]

The next chapter formalizes this relation in the context of nonmonotonic reasoning systems, using the concept of "defeasibility" inherent in nonmonotonic systems to spell out the meaning of "knowing more". Supervenience is defined to be the case in which lower levels can defeat higher level facts but not vice versa.

This definition accords well with many of the uses to which supervenience has historically been put. For example, to say that the mental supervenes on the physical

[16]The logic of supervenience is slightly more complicated than this suggests; see Chapter 5 for details.

is to say that the physical facts determine the mental facts, which is to say that the physical is "closer to the world" than the mental, which is to say that although we might make certain presumptions about a person's mental state (for example, that she is happy), these presumptions may be defeated by physical level facts (for example, that she is dead).

The concept of supervenience has had a stormy history but has nonetheless proven to be a useful idea in several areas. The essential idea is that of an asymmetric dependency relation between two or more domains of discourse which are not reducible to one another. This dependency relation can be characterized as a metric on epistemological "closeness to the world" or in terms of unidirectional defeasibility. These characterizations will be particularly useful in formalizing supervenience and in relating the theory of supervenience to work in dynamic-world planning.

Chapter 5

Supervenience Formalized

5.1 The Role and Nature of the Formalism

In the previous chapter I outlined the theory of supervenience, and in subsequent chapters I will discuss the role that supervenience plays in understanding and in designing intelligent agents. In the present chapter I provide a formal characterization of supervenience within the context of nonmonotonic reasoning systems. In this section I will discuss the reasons for developing a formal characterization at all, as well as motivations for the particular choice of the formal framework.

Formalization plays many roles in artificial intelligence research and in computer science generally. Although AI, at least insofar as its trends are reflected in the major conferences, is becoming increasingly enamored with formal models, there are those who question the wisdom of this tendency. Leith, in the Preface to his Formalism in AI and Computer Science, states that, “Computer science has... got its head stuck in the ditch of formalism. I would like to help pull it out” [Leith 1990, 9]. Davis and Hersh provide the following characterization:

> Formalism is a medical researcher who uses a computer to calculate standard deviations in an experiment that was ill conceived and ill executed. The statistical information is put into the paper simply because in today’s research world it is expected. It is part of the credentiation process. [Davis and Hersh 1986, 287]

Such scepticism may provide an important counter-balance to unrestrained formalization, but there are good reasons to formalize in computer science.

Formalization is a prerequisite for many types of analysis and comparison; an empirical study of the use of formal analysis in recent AI, along with methodological prescriptions for the field as a whole, can be found in [Cohen 1991]. One lesson to be learned from the debates about the role of formalism is that it is important to be clear about <u>why</u> formal models are being developed and the about the uses to which the formal models will be put.

The role of the formalism in this dissertation is largely descriptive; that is, by formalizing the notion of supervenience I hope to be able to communicate it more precisely. This use of logic is similar to that expressed by Shoham:

> Let me then summarize my position on logic. I will be using it as a way of lending clarity to my formulations. There may be other ways, but I have chosen logic, which on the one hand enjoys uncontested solid foundations, and on the other hand turns out to be useful for my purposes. [Shoham 1988, 24]

The purpose of the formal description of supervenience is to make precise the intuitive notion of supervenience outlined in Chapter 4. I will formalize a computational structure, called a <u>supervenient planning hierarchy</u>, that turns out to be quite powerful—powerful in the sense that just about any computational system that one might devise (more precisely, any computational system that can be formalized in one of the prevailing nonmonotonic logics) can be expressed as a supervenient planning hierarchy. Whether it would be <u>useful</u> to cast some given system as a supervenient planning hierarchy is another question entirely. I contend only that it is useful to look at dynamic-world planning systems in this way, and that the formalism can help researchers understand the notions of hierarchy and of levels of abstraction that have been used in the literature of dynamic-world planning.

I do not prove any theorems about supervenient planning hierarchies, nor, aside from a demonstration of the relation between supervenience and ABSTRIPS-style abstraction, do I prove anything about the relation between supervenience and other formal theories of reasoning. The implementation of the supervenience architecture, described in Part III, is not a direct implementation of the formal characterization of supervenience. Nonetheless, the formalism plays an important role in the exposition of the theory.[17] The existence of the formal characterization also allows for analyses and comparisons that would not otherwise be possible; some suggestions for future work along these lines are provided in Chapter 10.

Previous accounts of supervenience have included formal theories, and the literature also contains arguments concerning the extent to which each formal theory captures the appropriate intuitive notions (e.g., several papers in [Horgan 1984]). There is no consensus regarding which formal characterization is best, and the characterization provided in this dissertation is qualitatively different from those that have appeared before. For this reason it is not particularly useful to compare the present theory to past formalizations of supervenience in detail. Very broadly, however, the following characterization is appropriate: while previous formal theories of supervenience have modeled the notion of dependence indirectly, as an effect of equivalence and indiscernability relations that hold within and between levels of representation, the present theory models dependence directly, as an explicit feature of the logical machinery.[18]

At the end of Chapter 4, supervenience was characterized as follows:

> The essential idea is that of an asymmetric dependency relation between two or more domains of discourse which are not reducible to one another. This dependency relation can be characterized as a metric on "closeness to the world" or in terms of unidirectional defeasibility.

---

[17]See Section 7.2 for a discussion of the relation between the formal characterization of supervenience and its implementation in the supervenience architecture.

[18]I will have a bit more to say about comparisons to previous theories of supervenience in Chapter 10.

Domain *A* supervenes on domain *B* just in case representations in domain *A* depend on representations in domain *B*; that is, just in case, in matters of interest to both domains, the representations in domain *B* take precedence (because domain *B* is "closer to the world"). The motivation for the use of supervenience, rather than just "reduction" of *A*–representations to *B*–representations, is that *A* and *B* might use different languages (with neither a subset of the other), different rules of inference, etc. This idea of multiple domains of discourse, each with its own rules, is modeled as a set of separate logical reasoning systems. The reasoning systems are connected to one another in a hierarchical structure, and the notion of supervenience is formalized as a particular communication regime defined over the hierarchy.

The intuitive descriptions of supervenience use several terms that have been previously formalized in the context of nonmonotonic reasoning systems (for example, preference, dependence, defeasibility, etc.) Hence it is natural to formalize supervenience in a nonmonotonic framework. In particular, the notion of defeasibility can play a central role; the asymmetry of supervenience is modeled in the formalism as an asymmetric definition of defeasibility across the levels of a supervenient planning hierarchy.

There are additional reasons for basing the formalism on theories of nonmonotonic reasoning. Nonmonotonic reasoning systems have historically been motivated by the very domains of reasoning—reasoning about action, time, causality, etc.—that are of greatest concern in developing dynamic-world planners. As interest in nonmonotonic reasoning has grown, it has become clear that nonmonotonicity is an important part of many aspects of intelligence, and nonmonotonic logics have been proposed for a wide range of AI applications. Further, nonmonotonic logics generally subsume the standard monotonic logics as special cases, so formalizing in a nonmonotonic framework provides some measure of generality.

Konolige has developed a hierarchical theory of nonmonotonic reasoning, called hierarchic autoepistemic logic (HAEL), based on the autoepistemic logic of [Moore 1987]. He describes it as follows:

> In HAEL, the primary structure is not a single uniform theory, but rather, a collection of spaces linked in a hierarchy. Spaces represent different sources of information available to an agent, and the hierarchy expresses the way in which this information is combined. [Konolige 1988, 43]

HAEL might form an attractive basis for a formal theory of supervenience, except that the "spaces" in an HAEL theory are each individually monotonic. Defeasibility in HAEL occurs only between, and never within, levels. This is useful for Konolige, since he is seeking to avoid semantic and computational difficulties of more general nonmonotonic reasoning schemes. The formalization of supervenience is more general, but work on HAEL systems should be applicable to instances of supervenience that meet the HAEL restrictions.

Another possible basis for a formal theory of supervenience is the planning level hierarchies of [Hendler and Subrahmanian 1990], in which a set of planning levels, each based on an independent abstract logic, communicate by sharing logical conclusions. This model was motivated by many of the same considerations that motivated the present dissertation, and initial attempts were made to extend the theory to model supervenience. But the theory of planning level hierarchies, despite its generality in other respects, is an inappropriate basis for supervenience for the same reason as is HAEL: the nonmonotonic character of supervenience cannot be easily expressed. Hendler and Subrahmanian view their monotonicity restriction as temporary, even though it is currently stronger than that of HAEL.[19] Perhaps the theory of supervenience will be more readily expressed within the theory of planning

[19] In HAEL each individual level is monotonic; in the planning level hierarchies of Hendler and Subrahmanian the entire system must be monotonic as well.

level hierarchies once this restriction is removed.

5.2 Argument Systems

Lin and Shoham have developed a very general variety of nonmonotonic logic called an argument system [Lin and Shoham 1989]. Lin and Shoham's argument systems were chosen as the basis of the formal characterization of supervenience because of the generality of the argument system framework,[20] and because the proof-theoretic character of argument systems is more akin to the algorithmic specifications used in the planning literature than are the fixed-point and minimal model specifications used for other nonmonotonic reasoning systems.[21]

In this section Argument Systems are formally defined. The definitions are for the most part taken from [Lin 1991]; the syntax and notation have been modified to conform to the style of this dissertation, and explicit statements have been added pertaining to the use of variables.

Definition: Argument System

An Argument System Ψ is a 7-tuple (**Lang**, **Base**, **Mon**, **Non**, **Arg**, **Comp**, **Struct**) where:

- **Lang** is any language containing some set of well-formed formulae (wffs) and the symbol "¬" such that for any wff *w* in **Lang**, ¬ *w* is also in **Lang**. The symbol "True" must also be in **Lang**. **Lang** will not be specified in examples unless it is necessary to do so.

[20]Lin and Shoham claim that default logic, autoepistemic logic, the negation as failure principle, and circumscription are all special cases of their argument system framework [Lin and Shoham 1989].

[21]Lin provides further motivations for the study of argument-based systems, including the claim that the reasoning in argument-based systems "tends to be closer to human commonsense reasoning" [Lin 1991].

- **Base** is a subset of the wffs of **Lang** also called the set of <u>base</u> <u>facts</u>.
- **Mon** is a set of expressions called <u>monotonic</u> <u>rules</u> of the form $a_1, ..., a_n \rightarrow c$, where the a's and c are wffs of **Lang**. For any rule containing variables (specified with lower-case Greek letters; for example, α, β, γ, ...), rules corresponding to all possible instantiations of the variables are also assumed to be present.
- **Non** is a set of expressions called <u>nonmonotonic</u> <u>rules</u> of the form $a_1, ..., a_n \Rightarrow c$, where the a's and c are wffs of **Lang**. For any rule containing variables (specified with lower-case Greek letters; for example, α, β, γ, ...), rules corresponding to all possible instantiations of the variables are also assumed to be present.
- **Arg** is a set of <u>arguments</u>, formed by chaining elements of **Base**, **Mon**, and **Non** together into trees. The set of arguments is defined inductively as follows:
    1. If b is a base fact then the tree consisting of b as a single node is an argument. b is the called the <u>root</u> of that argument.
    2. If $A_1, ..., A_n$ are arguments whose roots are $a_1, ..., a_n$, respectively, and $a_1, ..., a_n \rightarrow c$ is a monotonic rule such that c is not a node in any one of the trees $A_1, ..., A_n$, then the tree with c as its root and $A_1, ..., A_n$ as its immediate subtrees is an argument, formed monotonically from $A_1, ..., A_n$.
    3. If $A_1, ..., A_n$ are arguments whose roots are $a_1, ..., a_n$, respectively, and $a_1, ..., a_n \Rightarrow c$ is a nonmonotonic rule such that c is not a node in any one of the trees $A_1, ..., A_n$, then the tree with c as its root and $A_1, ..., A_n$ as its immediate subtrees is an argument, formed nonmonotonically from $A_1, ..., A_n$.
- **Comp** is a subset of the wffs of **Lang** called the <u>completeness</u> <u>conditions</u>.
- **Struct** is a set of <u>argument</u> <u>structures</u>, each of which is a logically consistent subset of **Arg** which meets the completeness conditions **Comp**. A subset **S** of **Arg** is an argument structure (and hence <u>may</u> be in **Struct**) if the following conditions

are satisfied:

1. If b is a base fact in Ψ, then $b \in$ **S**.
2. **S** is closed by the subtree relation, that is, for any $t \in$ **S**, if t' is a subtree of t, then $t' \in$ **S**.
3. **S** is monotonically closed, that is, if t' is formed from $t_1, ..., t_n$ in **S** by a monotonic rule, then t' is also in **S**.
4. **S** is consistent, that is, it does not contain arguments for both w and $\neg w$, for any wff w.
5. **S** is complete, that is, for any $c \in$ **Comp**, either there is an argument in **S** for c or there is an argument in **S** for $\neg c$.

Additional Terminology

An argument with c as its root is said to be an argument for c, and c is said to be supported by the given argument.

If **S** is an argument structure then we use **Sup**(**S**) to refer to the set of all wffs supported by the arguments in **S**.

Example

The following is a simple syntactic reformulation of the "Penguins do not fly" example from [Lin 1991]. The "ab" predicate is to be read as "abnormal," with ab(α, β, γ) read as "α is an abnormal β with respect to γ." For example, the monotonic rule Penguin(α) $\rightarrow$ ab(α, bird, flight) is to read as "from the fact that α is a penguin, infer (monotonically) that α is an abnormal bird with respect to flight."

Let Animal-Categories = (**Lang**, **Base**, **Mon**, **Non**, **Arg**, **Struct**), where:

Base={ b_1: True,
b_2: Penguin(Opus) }

Mon={ m_1: Penguin(α) $\rightarrow$ Bird(α),
m_2: Penguin(α), $\neg$ ab(α, penguin, nonflight) $\rightarrow$ $\neg$ Fly(α),
m_3: Bird(α), $\neg$ ab(α, bird, flight) $\rightarrow$ Fly(α),
m_4: Penguin(α) $\rightarrow$ ab(α, bird, flight) }

Non={ n_1: True $\Rightarrow$ $\neg$ ab(α, penguin, nonflight),
n_2: True $\Rightarrow$ $\neg$ ab(α, bird, flight) }

Arg={ p_1: True,
p_2: Penguin(Opus),
p_3: $p_1 \Rightarrow^{n_1} \neg$ ab(Opus, penguin, nonflight),
p_4: $p_1 \Rightarrow^{n_2} \neg$ ab(Opus, bird, flight),
p_5: $p_2 \rightarrow^{m_1}$ Bird(Opus),
p_6: $p_2 \rightarrow^{m_4}$ ab(Opus, bird, flight),
p_7: p_3, $p_2 \rightarrow^{m_2} \neg$ Fly(Opus),
p_8: p_4, $p_5 \rightarrow^{m_3}$ Fly(Opus) }

If the completeness conditions are:

Comp={ ab(Opus, penguin, nonflight),
ab(Opus, bird, flight) }

then **S** is an argument structure which may be in **Struct**:

S={p_1, p_2, p_3, p_5, p_6, p_7}

Sup(**S**)={ True,
Penguin(Opus),
Bird(Opus),
$\neg$ ab(Opus, penguin, nonflight),
ab(Opus, bird, flight)
$\neg$ Fly(Opus) }

5.3 Layered Argument Systems and Supervenient Planning Hierarchies

The notion of an argument system can be extended to that of a layered argument system as follows: A layered argument system is a collection of some number of argument systems, over which a partial order « has been defined. The set of wffs supported by any argument in any argument structure of a given argument system is referred to as the proclamations of the system. The component systems of a layered argument system communicate by sharing proclamations; if system S_1 proclaims wff *w*, and if *w* is comprehensible to system S_2 (that is, if it is in the language of S_2), then *w* is made "available" to system S_2. The effect of such availability can be defined in various ways, resulting in different varieties of layered argument systems. The simplest communication mechanism would be for wffs available to a given system to be added as base facts in that system.

In a supervenient planning hierarchy the effects of availability are defined with respect to the ordering relation «. If S_1 makes *w* available to S_2 and S_1 « S_2, then *w* is added as a base fact of S_2, but if S_1 makes *w* available to S_2 and it is not the case that S_1 « S_2 then "True $\Rightarrow$ *w*" is added as a nonmonotonic rule of S_2. In other words, wffs made available from below are absorbed as facts, while wffs made available from elsewhere are absorbed as possible assumptions. Two new components are added to each argument system to accommodate the newly available wffs: $\mathbf{Base}_{avail}$ and $\mathbf{Non}_{avail}$. For the purposes of within-system reasoning, wffs in $\mathbf{Base}_{avail}$ are considered to be in **Base**, and wffs in $\mathbf{Non}_{avail}$ are considered to be in **Non**. The new components allow for expression of the dynamism of the communicative process; $\mathbf{Base}_{avail}$ and $\mathbf{Non}_{avail}$ are determined at any given time from the proclamations of other systems, while **Base** and **Non** are static components of each argument system.

Definition: Supervenient Planning Hierarchy

A Supervenient Planning Hierarchy Λ is a pair (Σ, «) where:

- Σ is a set of argument systems.
- The set of wffs supported by any argument structure of argument system Ψ in Σ is referred to as **Proclamations**(Ψ); that is,

  Proclamations(Ψ)=$\{w \mid \mathbf{S} \in \mathbf{Struct}(\Psi) \text{ and } w \in \mathbf{Sup}(\mathbf{S})\}$
- « is a partial order defined over Σ.
- Each argument system Ψ in Σ has an additional component $\mathbf{Base}_{avail}(\Psi)$. For the sake of argument construction, **Base**(Ψ) is considered to be

  $\mathbf{Base}(\Psi) \cup \mathbf{Base}_{avail}(\Psi)$. $\mathbf{Base}_{avail}(\Psi)$ is determined from the proclamations of other argument systems in Σ:

  $\mathbf{Base}_{avail}(\Psi) = \mathbf{Lang}(\Psi) \cap \{\mathbf{Proclamations}(\Phi) \mid \Phi \ll \Psi\}$.
- Each argument system Ψ in Σ has an additional component $\mathbf{Non}_{avail}(\Psi)$. For the sake of argument construction, **Non**(Ψ) is considered to be $\mathbf{Non}(\Psi) \cup \mathbf{Non}_{avail}(\Psi)$.

  $\mathbf{Non}_{avail}(\Psi)$ is determined from the proclamations of other argument systems in Σ:

  $\mathbf{Non}_{avail}(\Psi)=\{\text{“True} \Rightarrow w\text{”} \mid w \in \{\mathbf{Lang}(\Psi) \cap \{\mathbf{Proclamations}(\Phi) \mid \neg(\Phi \ll \Psi)\}\}\}$.

Example

Consider the two-level argument system with Σ={Animal-Categories, Animal-Physics}, where Animal-Categories is the argument system of the previous example, where Animal-Physics « Animal-Categories, and where Animal-Physics is specified as follows:

Let Animal-Physics = (**Lang**, **Base**, **Mon**, **Non**, **Arg**, **Comp**, **Struct**), where:

Base={ b_1: True,
b_2: Bird(Tweety),
b_3: Broken-wing(Tweety) }

Mon={ m_1: Broken-wing(α) $\rightarrow$ ab(α, bird, flight) }
Non={ }
Arg={ p_1: True,
p_2: Bird(Tweety),
p_3: Broken-wing(Tweety),
p_4: $p_3 \rightarrow^{m_1}$ ab(Tweety, bird, flight) }
Struct={ $\{p_1, p_2, p_3, p_4\}$ }
Comp={ }
Proclamations(Animal-Physics)={ True,
Bird(Tweety),
Broken-wing(Tweety),
ab(Tweety, bird, flight) }

Since Animal-Physics « Animal-Categories, the proclamations of Animal-Physics that are comprehensible to Animal-Categories are added to the base facts of Animal-Categories as elements of $\mathbf{Base}_{avail}$(Animal-Categories). Assuming that Broken-wing is the only predicate of Animal-Physics that is not in **Lang**(Animal-Categories), the following two base facts are added to Animal-Categories:

b_3: Bird(Tweety),
b_4: ab(Tweety, bird, flight)

This prevents argument structures of Animal-Categories from including Fly(Tweety); the lower-level knowledge of Tweety's broken wing defeats the higher level assumption of Tweety's flying ability, even though the Broken-wing predicate is available only at the lower level. Such an effect could also be achieved in a one-level system, but the multilevel approach allows for use of the abstraction and modularization mechanisms of partitioned control abstraction. Even in this tiny example, the rules and languages of each component system are smaller than those of the system as a whole;[22] it is therefore likely that the modularization will allow proofs to be more efficiently found. Note also that different proof strategies could be used at each level. Once Animal-Physics proclaims that Tweety is abnormal with respect to flight, Animal-Categories must revise its argument structures to accord with that knowledge. It is immaterial to Animal-Categories, however, how or why Animal-Physics comes to make such a proclamation.

[22]Although the languages have not been explicitly specified, it is reasonable to suppose that predicates such as Penguin are not in the lower-level (Animal-Physics) language. Each system clearly has <u>rules</u> that are not present in the other system.

Chapter 6

Supervenience and ABSTRIPS

The nonmonotonic formulation of supervenience is a generalization of simplification abstraction as used in ABSTRIPS-style systems. There is a trivial sense in which this is true: since each level of a supervenient planning hierarchy is a fairly general nonmonotonic reasoning system [Lin and Shoham 1989], any formulation of ABSTRIPS within a nonmonotonic reasoning system could be expressed as a single level of a supervenient planning hierarchy, and the supervenience relation would play no part in the formulation. I will show, however, that ABSTRIPS can be expressed as supervenience in the more intuitive sense in which levels of "criticality" in ABSTRIPS are transformed into the component systems in a supervenient planning hierarchy. The supervenience relation on the layered argument system corresponds to a relation between levels that is implicit in ABSTRIPS systems, but the more general nature of supervenience allows for the integration of hierarchical control concepts as well.

The demonstration in this chapter serves to show that the <u>type of abstraction</u> used in ABSTRIPS style systems is a special case of supervenience. But an implemented ABSTRIPS system is more than an embodiment of a type of abstraction—it is also a set of algorithms (for example, for plan refinement). No explicit transformation of ABSTRIPS algorithms into proof strategies on supervenient planning hierarchies is given in this dissertation. This is because the translation of such algorithms into logic is complex, and because such translations are not directly relevant to the issue at hand. The relation between supervenience and ABSTRIPS-style abstraction is addressed by exploring the relation between unidirectional defeasibility and simplification. The translation of algorithms from one formal system to the other is a different matter entirely.

The first step in the transformation involves the reformulation of STRIPS operators as logical inference rules. Expressing a delete-list as a set of negated literals, a typical STRIPS operator might be the following:

Operator: **PutOn**(x,y)
Preconditions: **Clear**(x), **Clear**(y)
Effects: **On**(x,y), ¬ **Clear**(y)

The logical construction of this operator involves the introduction of an additional variable for the situation in which the operator is applied, as in the situation calculus of [McCarthy 1968]. The operator is expressed as a rule in which the preconditions imply that the effects will be true in the situation that results from the performance of the action:

$$\mathbf{Clear}(\alpha, \sigma), \mathbf{Clear}(\beta, \sigma) \rightarrow \mathbf{On}(\alpha, \beta, \mathbf{PutOn}(\alpha,\beta,\sigma)) \;\&\neg\, \mathbf{Clear}(\beta, \mathbf{PutOn}(\alpha,\beta,\sigma))$$

The problem of finding a plan to satisfy a particular goal becomes one of finding an argument that proves that the goal holds in a situation formed by applying a sequence of operators. This technique has been referred to as Green's Formulation (e.g., see [Nilsson 1980, 308]). In order to mimic planning systems that make the STRIPS assumption, frame axioms must be added as well.

To model ABSTRIPS as a supervenient planning hierarchy an argument system is created for each criticality level, with each argument system having the same (complete) language. The operator-to-rule mappings just described are performed for each level, but at level i we remove all base facts with criticality less than i, and we also remove all literals with criticality less than i from the consequents of the monotonic rules. Rules for which this results in null consequents can be deleted altogether. In addition, we add nonmonotonic inference rules "True $\Rightarrow$ p" corresponding to all literals with criticality less than i. Note that the "proof power" of a level is not reduced by the deletions of sub-critical base facts or consequents; any deleted base fact or consequent will also be the consequent of a nonmonotonic rule, and may therefore be presumed at any time. Hence planning at level i is just like planning at the base level except that it is assumed that all preconditions of criticality less than i are satisfied without proof (or action). In ABSTRIPS this assumption is implemented by removing the preconditions from the operators; in the supervenient planning hierarchy the assumption is implemented by allowing the preconditions to be proven nonmonotonically, and by removing the possibility that the preconditions will be monotonically contradicted. This reformulation allows for easier expression of the interaction between levels during the development of a plan. For example, if p is assumed at level 2 but later $\neg p$ is proven at level 1, the assumption at level 2 will be defeated and operators with $\neg p$ as a precondition will be applicable at both levels. This upward communication is more important for a supervenient planning hierarchy operating in a dynamic domain than for an ABSTRIPS-style system in which control flow is primarily top-down.

Consider the 3-disk Towers of Hanoi problem that was discussed in Section 3.2. The criticality assignments <**OnLarge**, 2>, <**OnMedium**, 1>, and <**OnSmall**, 0> were used with the operators:

Operator: **MoveL**(x,y)
Preconditions: ¬ **OnSmall**(x), ¬ **OnSmall**(y), ¬ **OnMedium**(x),
¬ **OnMedium**(y), **OnLarge**(x)
Effects: ¬ **OnLarge**(x), **OnLarge**(y)

Operator: **MoveM**(x,y)
Preconditions: ¬ **OnSmall**(x), ¬ **OnSmall**(y), **OnMedium**(x)
Effects: ¬ **OnMedium**(x), **OnMedium**(y)

Operator: **MoveS**(x,y)
Preconditions: **OnSmall**(x)
Effects: ¬ **OnSmall**(x), **OnSmall**(y)

Translating this system into the layered argument system formalism we get at the base level:

Hanoi-0 = (**Lang**, **Base**, **Mon**, **Non**, **Arg**, **Comp**, **Struct**), where:

Base={ b_1: True,
b_2: **OnSmall**(P_1, Init),
b_3: **OnMedium**(P_1, Init),
b_4: **OnLarge**(P_1, Init) }

Mon={ m_1: ¬ **OnSmall**(α, σ), ¬ **OnSmall**(β, σ), ¬ **OnMedium**(α, σ),
¬ **OnMedium**(β, σ), **OnLarge**(α, σ)
→ ¬ **OnLarge**(α, **MoveL**(α,β,σ)),
OnLarge(β, **MoveL**(α,β,σ)),
m_2: ¬ **OnSmall**(α, σ), ¬ **OnSmall**(β, σ), **OnMedium**(α, σ)
→ ¬ **OnMedium**(α, **MoveM**(α,β,σ)),
OnMedium(β, **MoveM**(α,β,σ)),
m_3: **OnSmall**(α, σ)
→ ¬ **OnSmall**(α, **MoveS**(α,β,σ)),
OnSmall(β, **MoveS**(α,β,σ)),
m_4: **OnSmall**(α, σ) → **OnSmall**(α, **MoveM**(α,β,σ)),
m_5: **OnSmall**(α, σ) → **OnSmall**(α, **MoveL**(α,β,σ)),
m_6: **OnMedium**(α, σ) → **OnMedium**(α, **MoveS**(α,β,σ)),
m_7: **OnMedium**(α, σ) → **OnMedium**(α, **MoveL**(α,β,σ)),

m_8: **OnLarge**$(\alpha, \sigma) \rightarrow$ **OnLarge**$(\alpha,$ **MoveS**$(\alpha,\beta,\sigma))$,
m_9: **OnLarge**$(\alpha, \sigma) \rightarrow$ **OnLarge**$(\alpha,$ **MoveM**$(\alpha,\beta,\sigma))$,
m_{10}: $\neg$ **OnSmall**$(\alpha, \sigma) \rightarrow \neg$ **OnSmall**$(\alpha,$ **MoveM**$(\alpha,\beta,\sigma))$,
m_{11}: $\neg$ **OnSmall**$(\alpha, \sigma) \rightarrow \neg$ **OnSmall**$(\alpha,$ **MoveL**$(\alpha,\beta,\sigma))$,
m_{12}: $\neg$ **OnMedium**$(\alpha, \sigma) \rightarrow \neg$ **OnMedium**$(\alpha,$ **MoveS**$(\alpha,\beta,\sigma))$,
m_{13}: $\neg$ **OnMedium**$(\alpha, \sigma) \rightarrow \neg$ **OnMedium**$(\alpha,$ **MoveL**$(\alpha,\beta,\sigma))$,
m_{14}: $\neg$ **OnLarge**$(\alpha, \sigma) \rightarrow \neg$ **OnLarge**$(\alpha,$ **MoveS**$(\alpha,\beta,\sigma))$,
m_{15}: $\neg$ **OnLarge**$(\alpha, \sigma) \rightarrow \neg$ **OnLarge**$(\alpha,$ **MoveM**$(\alpha,\beta,\sigma))$}

Non={ }

Monotonic rules m_1–m_3 are derived directly from the STRIPS operators. Monotonic rules m_4–m_{15} are frame axioms; they are not needed in an ABSTRIPS-style system because the STRIPS-assumption is implicit in the planning algorithm, but in the present formalism the frame axioms are introduced explicitly. Various techniques exist for reducing the number of required frame axioms (e.g., Kowalski's Formulation [Nilsson 1980, 311–315]), but they are not pertinent to the arguments of this dissertation. Alternatively, the frame problem might be handled by making use of the nonmonotonicity inherent in the argument systems.[23]

At level 1 the system is reduced by the removal of all base facts and monotonic consequents containing **OnSmall.** In addition, nonmonotonic rules are added to allow the presumption of any needed **OnSmall** formulae:

Hanoi-1 = (**Lang**, **Base**, **Mon**, **Non**, **Arg**, **Comp**, **Struct**), where

Base={ b_1: True,
b_2: **OnMedium**$(P_1$, Init$)$,
b_3: **OnLarge**$(P_1$, Init$)$ }

Mon={ m_1: $\neg$ **OnSmall**(α, σ), $\neg$ **OnSmall**(β, σ), $\neg$ **OnMedium**(α, σ),
$\neg$ **OnMedium**(β, σ), **OnLarge**(α, σ)

[23]Several nonmonotonic approaches have been suggested for the frame problem; see discussions in [Hanks and McDermott 1990] and [Kyburg, Loui, and Carlson 1990, Part I].

$\rightarrow \quad \neg$ **OnLarge**(α, **MoveL**(α,β,σ)),

OnLarge(β, **MoveL**(α,β,σ)),

m_2: $\neg$ **OnSmall**(α, σ), $\neg$ **OnSmall**(β, σ), **OnMedium**(α, σ)

$\rightarrow \quad \neg$ **OnMedium**(α, **MoveM**(α,β,σ)),

OnMedium(β, **MoveM**(α,β,σ)),

m_3: **OnMedium**(α, σ) $\rightarrow$ **OnMedium**(α, **MoveS**(α,β,σ)),

m_4: **OnMedium**(α, σ) $\rightarrow$ **OnMedium**(α, **MoveL**(α,β,σ)),

m_5: **OnLarge**(α, σ) $\rightarrow$ **OnLarge**(α, **MoveS**(α,β,σ)),

m_6: **OnLarge**(α, σ) $\rightarrow$ **OnLarge**(α, **MoveM**(α,β,σ)),

m_7: $\neg$ **OnMedium**(α, σ) $\rightarrow$ $\neg$ **OnMedium**(α, **MoveS**(α,β,σ)),

m_8: $\neg$ **OnMedium**(α, σ) $\rightarrow$ $\neg$ **OnMedium**(α, **MoveL**(α,β,σ)),

m_9: $\neg$ **OnLarge**(α, σ) $\rightarrow$ $\neg$ **OnLarge**(α, **MoveS**(α,β,σ)),

m_{10}: $\neg$ **OnLarge**(α, σ) $\rightarrow$ $\neg$ **OnLarge**(α, **MoveM**(α,β,σ))}

Non={ n_1: True $\Rightarrow$ **OnSmall**(α, σ),

n_2: True $\Rightarrow$ $\neg$ **OnSmall**(α, σ)}

At level 2 the **OnMedium** base facts and monotonic consequents are removed, and nonmonotonic rules for **OnMedium** are added:

Hanoi-2 = (**Lang**, **Base**, **Mon**, **Non**, **Arg**, **Comp**, **Struct**), where

Base={ b_1: True,

b_2: **OnLarge**(P_1, Init) }

Mon={ m_1: $\neg$ **OnSmall**(α, σ), $\neg$ **OnSmall**(β, σ), $\neg$ **OnMedium**(α, σ),

$\neg$ **OnMedium**(β, σ), **OnLarge**(α, σ)

$\rightarrow \quad \neg$ **OnLarge**(α, **MoveL**(α,β,σ)),

OnLarge(β, **MoveL**(α,β,σ))

m_2: **OnLarge**(α, σ) $\rightarrow$ **OnLarge**(α, **MoveS**(α,β,σ)),

m_3: **OnLarge**(α, σ) $\rightarrow$ **OnLarge**(α, **MoveM**(α,β,σ))

m_4: $\neg$ **OnLarge**(α, σ) $\rightarrow$ $\neg$ **OnLarge**(α, **MoveS**(α,β,σ)),

m_5: $\neg$ **OnLarge**(α, σ) $\rightarrow$ $\neg$ **OnLarge**(α, **MoveM**(α,β,σ))}

Non={ n_1: True $\Rightarrow$ **OnSmall**(α, σ),

n_2: True $\Rightarrow$ $\neg$ **OnSmall**(α, σ),

n_3: True $\Rightarrow$ **OnMedium**(α, σ),

n_4: True $\Rightarrow \neg$ **OnMedium**(α, σ) }

The set of arguments and argument structures generated by this example are considerably more complex than for the simple examples considered earlier. However, it is fairly easy to see how an argument for the abstract plan from Section 3.2 can be formed. The abstract plan:

<**MoveL**(P_1,P_3), **MoveM**(P_1,P_3), **MoveS**(P_1,P_3)>

is expressed in our new formalism as:

MoveS(P_1,P_3, **MoveM**(P_1,P_3, **MoveL**(P_1,P_3, Init)))

The statement that this plan solves the given problem is expressed as the following three assertions:[24]

OnLarge(P_3, **MoveS**(P_1,P_3, **MoveM**(P_1,P_3, **MoveL**(P_1,P_3, Init)))),

OnMedium(P_3, **MoveS**(P_1,P_3, **MoveM**(P_1,P_3, **MoveL**(P_1,P_3, Init)))),

OnSmall(P_3, **MoveS**(P_1,P_3, **MoveM**(P_1,P_3, **MoveL**(P_1,P_3, Init)))).

An argument for the first of these is formed at level 2 as follows:

p_1: True $\Rightarrow^{n_2} \neg$ **OnSmall**(P_1, Init)

p_2: True $\Rightarrow^{n_2} \neg$ **OnSmall**(P_3, Init)

p_3: True $\Rightarrow^{n_4} \neg$ **OnMedium**(P_1, Init)

p_4: True $\Rightarrow^{n_4} \neg$ **OnMedium**(P_3, Init)

p_5: p_1, p_2, p_3, p_4, b_2 $\rightarrow^{m_1} \neg$ **OnLarge**(P_1, **MoveL**(P_1,P_3,Init)),

OnLarge(P_3, **MoveL**(P_1,P_3,Init))

p_6: p_5 $\rightarrow^{m_5}$ **OnLarge**(P_3, **MoveM**(P_1,P_3,**MoveL**(P_1,P_3,Init)))

p_7: p_6 $\rightarrow^{m_4}$ **OnLarge**(P_3, **MoveS**(P_1,P_3,**MoveM**(P_1,P_3,**MoveL**(P_1,P_3,Init))))

---

[24]A more complete statement of the solution might include additional statements indicating, for example, that the disks are <u>not</u> on peg P_1 at the completion of the plan. Such complications will be omitted as they add nothing to the substance of the discussion.

Arguments for the second and third assertions follow trivially from the nonmonotonic rules. The Hanoi-2 argument system produces a plan for the Towers of Hanoi problem assuming that the small and medium disks are always in the "right" places at the right times. This use of "default" assumptions for search-space reduction is implicit in [Knoblock, Tenenberg, and Yang 1991], and is discussed explicitly in [Elkan 1990] and [Ginsberg 1991]. Of course, the assumptions are not true in the given problem at level 1. At level 1 arguments must be produced to show that the medium disk <u>is</u> in the right place or to describe the actions which would <u>make</u> it be in the right place. As with ABSTRIPS-style systems, proof-control algorithms could be constructed to use the level 2 solution to guide the construction of a level 1 argument, and such a procedures could be generalized to an arbitrary number of levels.[25] Unlike ABSTRIPS systems, the supervenient planning hierarchy specifies that disproved assumptions are made available to the higher-level system, which may combine the new information with other knowledge, possibly unavailable at the lower level, in order to formulate additional plans.

The nature of the top-down communication is roughly, "Here is a skeletal plan that might work." As with ABSTRIPS, the lower levels of a supervenient planning hierarchy can be used to refine and eventually to execute the plan.[26] The supervenient planning hierarchy also provides bottom-up communication of the form, "Here is an assumption that turned out to be incorrect." This information may be ignored by the higher level (if it has no relevant rules), or it may be used to revise or to revoke the original skeletal plan. This kind of "backtracking" also occurs in some ABSTRIPS-style systems, as part of the search process in the static-world total-planning

[25]The details of such procedures are outside the scope of this dissertation.

[26]For purposes of comparison it is assumed that the ABSTRIPS system is connected to an execution module at its lowest level.

framework.[27] Within the framework of generating planned activity, such "backtracking" is essential—it allows for reaction to changes in the world, and for the integration of reaction with higher-level planning as discussed in Section 2.3. The theory of supervenience formalizes this bidirectional communication and the intuition that the essential constraint on such communication is that the lower levels, being closer to the world, "know more" about the world. In saying that the lower levels "know more" about the world I mean that their knowledge of the world cannot be defeated from above. In ABSTRIPS-style systems this constraint is met because lower levels know more—period. In a supervenient planning hierarchy the lower levels know more about the world, but the higher levels may know more about abstract representations of the world and about their interactions. It is in this sense that supervenience is a generalization of ABSTRIPS-style abstraction.

The above transformation of ABSTRIPS systems to layered argument systems is not particularly concise,[28] but it does serve to show that reduced partition abstraction can be expressed in a supervenient planning hierarchy. The unidirectional flow of defeasibility models the condition, implicit in ABSTRIPS-style systems, that the lower-level constraints are more restrictive than higher-level constraints. However, in matters not constrained by lower level knowledge, supervenience allows for arbitrary higher level reasoning, while reduced partition abstraction allows for none.

Knoblock has studied various properties of abstraction hierarchies and their impact on the efficiency of ABSTRIPS-style problem solvers. For example, the ordered monotonicity property states that high-level solutions may have to be refined, but will never have to be reformulated, at the lower levels of an abstraction hierarchy.

[27]Knoblock has shown that under certain conditions such backtracking can be avoided [Knoblock 1991a].

[28]The frame axioms are to blame for much of the mess, but the transformation is somewhat cumbersome even without them.

Hierarchies that have the ordered monotonicity property allow for the use of more efficient search algorithms, and hierarchies with this property can be generated automatically [Knoblock 1991a]. There is no reason that such an analysis cannot be brought to bear on the algorithms of supervenient planning systems as well. But they are not always appropriate; dynamic domains sometimes force the reformulation of high-level plans, and in such cases the hierarchical control principles of robotic systems and of recent dynamic-world planners provide better guidance to a problem solver than do the principles of reduced partition abstraction. Supervenient planning hierarchies were designed to capture the essential aspects of both paradigms of abstraction, insofar as they contribute to the design of intelligent reactive agents.

The transformed Towers of Hanoi problem serves to relate supervenience to reduced partition abstraction, but the Towers of Hanoi problem is a static-world problem. The problem can be made dynamic by assuming the existence of demons that manipulate the disks during the game.[29] Of course, if the demons are fast enough then the problem solver will always fail. But assuming some lethargy on the part of the demons, the problem-solver should eventually succeed by noticing changes in the world and reformulating its plans appropriately.

It is not clear that the above supervenient planning hierarchy would be the best supervenient planning hierarchy for the Demons of Hanoi domain. This is because size of a disk has little to do with "closeness to the world" in the sense that motivates the use of supervenience. Smaller disks are closer to the world in one sense, and this is also the reason that the partitioning by disk size works well for ABSTRIPS: smaller disks can be moved more directly (they'll have less on top of them) and plans for the movements of smaller disks will therefore generally run into less complications. Plans for the movement of large disks are messier and therefore "further from the world."

[29] Cf. the "mischief in the blocks world" domain of [Schoppers 1987].

The benefits of the supervenient planning hierarchy will be more apparent, however, if we complicate the domain a bit more. Suppose that the game-playing robot has a vision system that is much better at detecting small disks than it is at detecting large disks. The position of the medium disk could be inferred from the position of the small disk or through further perceptual processing, but this would take time. Likewise the position of the large disk could be inferred or computed through an even greater expenditure of computational resources. This domain is intentionally rigged to fit the supervenient planning system that has already been developed, but it can be used to reveal the strengths of the supervenience approach nonetheless.

The rules for inferring the positions of the larger disks constitute new non-simplifying knowledge at higher levels, which supervenience allows but which reduced partition abstraction does not. The procedures of the lowest level could be optimized for discovering plans that only involve movement of the small disk, the middle level could be similarly optimized for the medium disk, and the highest level for the large disk. When an unexpected disk position is detected, the lowest level could attempt to fix the problem first, since the small disk is always directly movable and since knowledge of the small disk's position is most sure. If time permits, or if it becomes necessary, higher levels could make their analyses known. When the world is "static enough" the behavior of an ABSTRIPS-style system could re-emerge; the higher levels could direct the search of the lowest level, which would eventually solve the problem.

Further complications of the domain can make the supervenient planning hierarchy even more attractive. For example, suppose that the problem solver knows that the demons can be appeased by the construction of a temporary tower on the middle peg. This knowledge could be encoded into the rules of the highest level, where it could be used to generate skeletal plans that include demon-appeasement.

There is no reason to incorporate such knowledge into the lowest level, as mandated in ABSTRIPS-style systems—it would only contribute to the bloating of the lower levels and to the diminution of reactivity. The bottom levels of a reactive system are the worst place to introduce complexity, and in a supervenient planning hierarchy the lowest levels need not be burdened with high-level knowledge.

Hence ABSTRIPS-style reasoning can be used in a supervenient planning hierarchy to improve search efficiency on essentially static components of dynamic problems while the hierarchical control features of the system are used to improve reactivity. Finding domain partitions that facilitate such applications in real-world domains may be more difficult; this topic is given further attention in Section 8.2.

Part III

Implementation

Chapter 7

The Supervenience Architecture

7.1 Introduction

In Part II of this dissertation I described a relation called supervenience that holds between levels in certain multilevel reasoning systems. I provided an intuitive characterization of supervenience, related supervenience to other level-based ideas from AI, philosophy and psychology, and provided a formal characterization of supervenience in terms of nonmonotonic reasoning systems. In Part III, I will describe an implemented dynamic-world planning system developed on the basis of this notion of supervenience.

The implementation serves to highlight the connection, implied by the theory of supervenience, between dynamic-world planning research and hierarchical control theory. While the programming concepts are reasonably typical of dynamic-world planning research, the resulting system is accurately described by accounts of hierarchical control theory:

> If we now look at the hierarchical systems as a whole . . . we see that they have one feature in common: the decision making has been divided. Moreover, it has been divided in a way leading to hierarchical dependence. This means, *that there exist several decision units in the structure, but only some of them have direct access to the controlled system. The others are at a higher level of the hierarchy—they define the tasks and coordinate the lower-level units, but they do not override their decisions.* [Findeisen et al. 1980, 12]

In the remainder of this chapter I will describe the <u>supervenience</u> <u>architecture</u>, which is the general architectural model that embodies the notion of supervenience.

In the following chapters I will describe the Abstraction-Partitioned Evaluator (APE), which is an implementation of the supervenience architecture, and HomeBot, a system built using APE.

7.2 The Gulf Between Theory and Practice

The program that will be described is not a direct implementation of the formal characterization of supervenience. It is rather an application of the <u>idea</u> of supervenience to a procedural programming system; it resembles previous dynamic-world planning systems more closely than it resembles logic programming systems. The formal characterization of supervenience was expressed in a layered, proof-theoretic argument system. It should be possible to implement such an argument system, to formulate planning problems as proof problems at various levels, and to thereby implement a dynamic-world planner. While such a direct implementation would provide the best support for the theoretical work, it also presents certain difficulties, and a procedural implementation strategy was adopted instead.

There are several reasons for this gap between theory and practice. Foremost is the fact that procedural specifications have played a greater role than have logic-based representations in modern implementation-based planning research. This is true for the expression of basic planning algorithms and in some cases for the expression of problem-solving knowledge (for example, procedural operators expressed in the SOUP code of NOAH [Sacerdoti 1975]). In dynamic-world planning systems in particular, complex control structures with intuitive procedural definitions have often been used (for example, the plan nets of [Drummond 1990] or the RAPS of [Firby 1989]). While the translation of such control concepts into a logic-based formalism is possible in theory, the retention of the procedural style of programming and representation allows for more natural specifications, and for straightforward comparisons to other work in the field.

Another problem with "direct implementation" is that logical argument systems are difficult to implement in the first place. While [Lin 1991] describes work in this direction, including an implemented system that answers queries for certain default theories, this work is still preliminary. Problems regarding termination and correctness still remain (e.g., see [Lin 1991, 39]). The representational requirements of large dynamic-world planning systems are not yet well enough understood to ascertain the extent to which they are compatible with the restrictions of currently implemented argument systems.

An additional reason for the gap between theory and practice has to do with the process by which the theory was developed. For purposes of exposition the theory has been presented as the foundation on which the implementation would be built. This is only partially true—in fact, the theoretical and implementational aspects of the work were carried out in tandem. Requirements discovered during programming sessions influenced the development of the theory, just as principles from the emerging theory influenced the design of the program. Only once the theory had been fully developed could a project of "direct implementation" be launched (this is a possible avenue for future work).

The idea of supervenience influenced the design of the program in several ways. The overall architecture, consisting of a number of independent reasoning systems, organized into levels by "closeness to the world," is an obvious influence. The choice of specific levels for any implementation should also be based on the idea that upper levels should supervene on the lower levels in the sense explicated in Chapter 4.[30] Most significantly, the manner in which levels of representation are permitted to communicate is based on the concept of unidirectional defeasibility developed in

[30]See Section 8.2 for a discussion of the supervenience relations that hold amongst the specific levels chosen for APE.

formalizing supervenience.

The formal characterization of supervenience in Part II can be abbreviated as "assertions up, assumptions down." This captures the idea that higher-level systems in a supervenient planning hierarchy absorb lower-level knowledge as <u>facts</u>, while lower-level systems absorb higher-level knowledge as <u>assumptions</u> or <u>defaults</u>. In the supervenience architecture the characterization becomes "world knowledge up, goals down." The change in terminology follows the transition from a logical to a procedural framework. The logical equivalence of defaults and goals is a presupposition of this transition; evidence for such an equivalence is provided by Horty's analysis of imperatives in the context of default logic [Horty 1992].

7.3 General Architecture

In this section I provide a generic specification for the design of supervenient dynamic-world planners. I refer to this generic specification as the <u>supervenience architecture</u>. Many of the details that must be specified for an actual implementation are not specified by the architecture *per se.* Rather, the architecture specifies, in general terms, the set of basic components out of which any such implementation would be composed, and the manner in which these components may be connected. The purpose of the generic specification is to describe a framework for planning at multiple levels of supervenience, uncluttered by the implementation details of any given system. In the subsequent chapter I describe the Abstraction Partitioned Evaluator (APE), an implemented dynamic-world planning system for which the generic specifications of the supervenience architecture have been specialized to concrete details.

The supervenience architecture is based on the architectural model of multi-level blackboard systems [Erman and Lesser 1975], [Erman et al. 1980]. Even in the earliest blackboard systems, provisions were made for blackboards that could be partitioned by “level of abstraction,” and for systems in which procedural components were allowed to execute in parallel. More recently, several researchers have been exploring techniques for actually realizing the promise of parallelizability inherent in the blackboard model [Bisiani and Forin 1989], [Corkill 1989], [Craig 1989], [Jagannathan 1989]. Others have been exploring the requirements of real-time AI systems, and the modifications of the blackboard model that may be necessary for their satisfaction [Dodhiawala, Sridharan, and Pickering 1989], [Fehling, Altman, and Wilber 1989], [Hewett and Hayes-Roth 1989], [Lesser, Pavlin, and Durfee 1989], [Raulefs 1989].

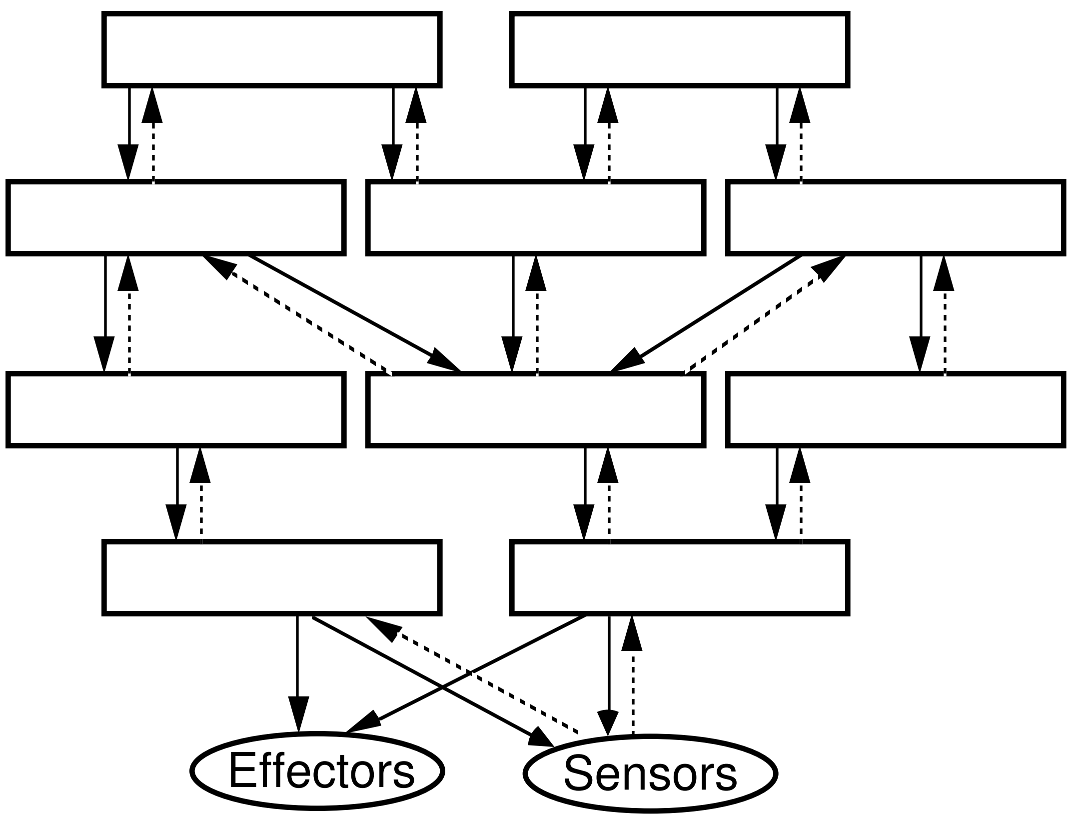

Figure 2. A partially ordered set of planning levels.

The supervenience architecture consists of a number of data/processing levels arranged in a partial order, with the minimal elements of the partial order connected to sensors and effectors. Figure 2 shows one such possible arrangement. The solid arrows define the partial order, while both the solid and dotted arrows indicate flow of information. Figure 3 illustrates the special case of total-ordering. Each level communicates only with those levels immediately above and below itself in the hierarchy. This communication regime is more restrictive than that specified in the formalization of Chapter 5, in which each level can communicate with each other level in the system. The same effect can be achieved in the supervenience architecture either by adding links to the partial order or by "relaying" information through intermediate levels. The restriction to communication between immediate neighbors is of no theoretical significance, but it simplifies implementation of the communication procedures. Only the lowest levels (the minimal elements of the partial order) have direct access to sensors and effectors. The levels are permitted to run in parallel and to communicate asynchronously.

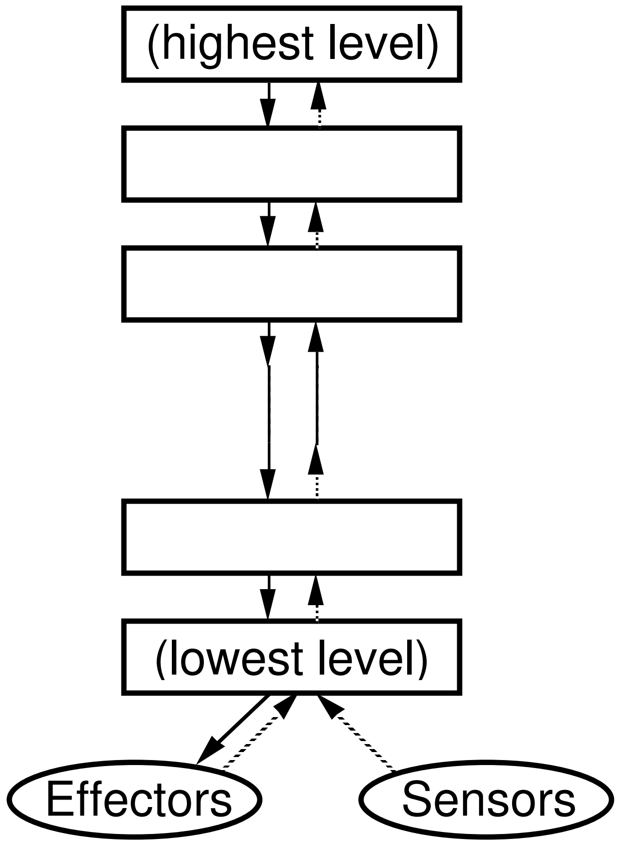

Figure 3. A totally ordered supervenient planning hierarchy.

Figure 4 shows a single level of the system in greater detail. Each level contains both procedural and declarative components. The declarative knowledge at each level resides in a blackboard structure accessible to all procedures at that level. This shared knowledge forms a representation of the current state of the world as seen by the given level. Borrowing terminology from [Hendler and Sanborn 1987], this state-of-the-world-at-a-given-level is called the “state of affairs” or SOA.

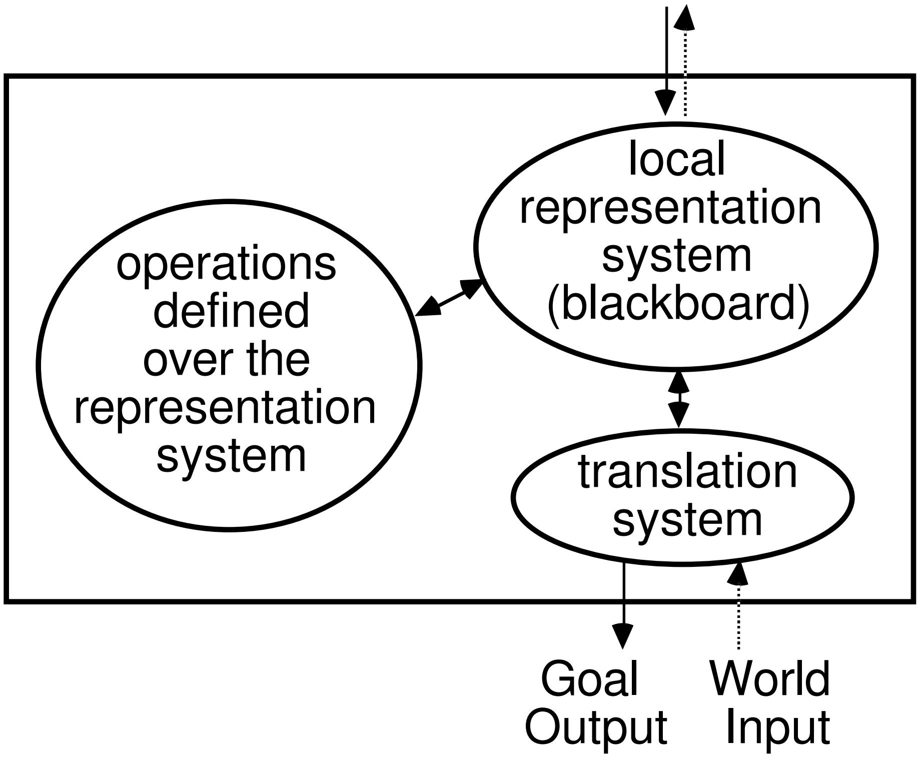

Figure 4. A single level of the supervenience architecture.

Communication between levels is facilitated by translation systems situated "beneath" each planning level (except for those at the bottom of the hierarchy). The translation systems pass goals downward and information about the state of the world upward. They also perform syntactic manipulation of blackboard sentences in order to accommodate the variations in languages across the levels of the hierarchy. Translation systems play a critical role in the supervenience architecture. They implement the asymmetric supervenience relation between levels; that is, they implement the dependence of high-level knowledge upon low-level knowledge, as described in Part II. This is accomplished by specifying that only goal-related knowledge is passed downward, while knowledge about the state of the world is passed upward. This "world knowledge up, goals down" style of communication is the implementational analog to the "assertions up, assumptions down" communications regime developed for the formal characterization of supervenience in Part II. In APE, translation systems are specified using an economic producer/consumer metaphor, casting lower levels as producers and higher levels as consumers (see Section 8.6). The translation systems are also permitted to run in parallel with the operations at each level.

Although the procedures at each level should possess certain general properties (such as interruptability at some level of temporal granularity) the architecture makes no stipulations about the reasoning strategies or micro-architectures within the levels themselves. The system might even allow the use of radically different planning methodologies at each level. For example, one might use a connectionist system at the lowest level, logic programming techniques at intermediate levels, and a planning system such as NONLIN at the highest levels. The supervenience architecture is intended to accommodate any such a range of techniques, as long as all of the planning systems interact with the blackboard mechanism using a uniform protocol.

7.4 Comparison to the Subsumption Architecture

The subsumption architecture of Brooks is a successful and well known architecture for mobile robots [Brooks 1990]. Although Brooks's anti-representation stance is at odds with the approach to dynamic-world planning advocated in this dissertation (see Section 3.3), it can nonetheless be useful to compare the subsumption and supervenience architectures. Representational philosophy aside, the subsumption and supervenience architectures can be meaningfully compared with respect to the issues of horizontal vs. vertical decomposition, the complexity of reasoning allowed at each level, and the kind of communication allowed between levels.

Brooks characterizes previous systems as decomposing behavioral problems into a series of "functional units" that can be illustrated by a series of "vertical slices" (see

Figure 5). He prefers a decomposition into "task achieving behaviors," illustrated by a series of "horizontal slices" (see Figure 6).

The supervenience architecture combines these two methods of decomposition (as do others, e.g. [Albus 1991]). The supervenience architecture allows, as does the subsumption architecture, the direct coupling of sensation to action at the lowest level of the system. The lowest levels can provide basic "competencies" without the intervention of higher, more complex levels. On the other hand, the supervenience architecture stipulates that the higher levels receive their knowledge of the world via the lower levels of the system, and that they pass commands for action through the lower levels as well. This accords with some of the principles of "vertically sliced" systems; for example, that the knowledge required for planning is provided via modeling from perception, and not from the sensors directly. The resulting mix of horizontal and vertical decomposition is quite practical—indeed, some examples of the subsumption architecture exhibit such a combination [Brooks 1990, 15–19].

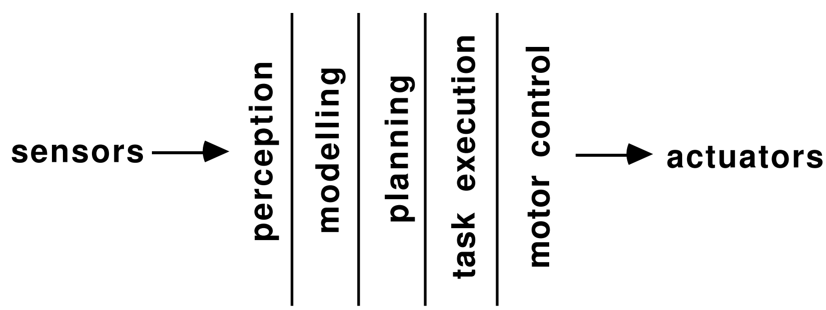

Figure 5. "Traditional" control system decomposition adapted from [Brooks 1990].

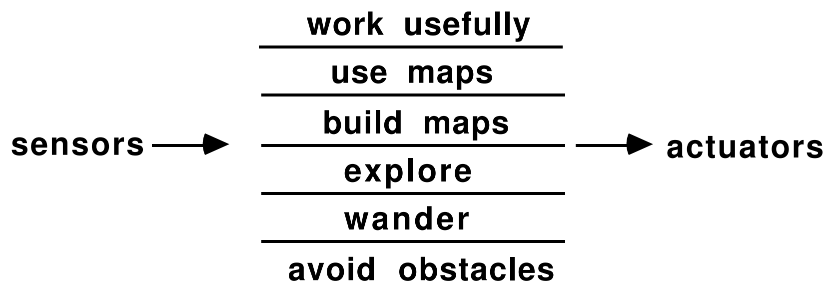

Figure 6. “Task achieving” decomposition adapted from [Brooks 1990].

In the subsumption architecture, the “reasoning” at each level is performed by a set of “modules” implemented as finite state machines. The finite state machines are “hard wired”—they are created when the system is built and are never changed. The topology of the module network is also static. The modules may contain Lisp data and reasonably complex functions for computing outputs, changing state, etc., but their power is still quite limited. Regardless of the theoretical computational power of such systems, it is not at all clear how one would encode complex symbolic AI systems as finite state machines. The supervenience architecture, in contrast, allows for arbitrary symbolic processes at each level.

Bottom-up communication in the subsumption architecture is simply a matter of connecting the appropriate “wires” so that low-level data flows to high-level modules. Modulo the existence of translation systems, bottom-up communication in the supervenience architecture is similar. A more significant difference appears with respect to top-down communication. In the subsumption architecture, top-down communication occurs by “inhibition” of the outputs of low-level modules, and by the “suppression” and “replacement” of the inputs to low-level modules. In the case of inhibition, high-level modules simply cancel the effects of low-level modules that continue to run. In the case of substitution/replacement, higher-level modules <u>replace</u> the inputs of lower-level modules, thereby altering their behavior.

The use of suppression/replacement for knowledge about the world is entirely inconsistent with the principles of the supervenience architecture. In the supervenience architecture lower levels are epistemologically “closer to the world”—their knowledge cannot be defeated by knowledge from higher levels. Higher levels in the supervenience architecture communicate with lower levels via goals—they configure the behavior of the lower levels, but they do not override lower-level knowledge of the world.

The use of action inhibition is also inconsistent with the principles of the supervenience architecture. Lower-level procedures, on the basis of goals (perhaps obtained from higher levels) and knowledge of the world, compute and execute actions to achieve goals; they are presumed to have the necessary knowledge to achieve their goals at their level. The higher levels may provide goals, but the determination of appropriate lower-level actions to achieve such goals is the prerogative of the lower, not the higher, level.

It is possible to restrict the top-down communication in a subsumption system to conform with the principles of the supervenience architecture; the use of inhibition can be avoided, and inputs can be grouped as “goal” and “world” inputs, with suppression/replacement allowed only for the former. Such restrictions are not part of the subsumption architecture, however, and their absence marks an important difference between the subsumption and supervenience architectures.

Chapter 8

The Abstraction-Partitioned Evaluator (APE)

8.1 Introduction

The Abstraction Partitioned Evaluator (APE) is an implementation of one example of a system based on the supervenience architecture. While the supervenience architecture specifies the general structure of supervenient planning systems, APE specifies concrete implementation decisions for a single program.[31] APE is important as a demonstration of the supervenience architecture, but the mechanisms described in this chapter should not be viewed as essential to the architecture; rather, they should be viewed as illustrative of the implementational possibilities.

APE was programmed in Common Lisp on an Apple Macintosh computer.[32] The graphing tools and a few of the timing procedures used for simulating parallelism depend on specifics of Macintosh Common Lisp. The bulk of the code, however, is in portable Common Lisp (with CLOS). Although I have endeavored to keep details of coding to a minimum, some familiarity with the Common Lisp language, as described in [Steele 1990], will be assumed in the description of APE.

---

[31] APE, though a single program, is actually a "shell" with which any number of dynamic-world planning systems may be generated. HomeBot (Chapter 9) is one such system.

[32] Most of the development work was done on a Macintosh IIcx, which contains a Motorola 68030 processor and a Motorola 68882 floating-point coprocessor, running with a clock frequency of 15.6672 MHz.

8.2 Specific Levels

The supervenience architecture does not dictate any particular set of levels, but each implementation of the supervenience architecture must make some such choice. The concept of supervenience provides guidance in choosing levels, stipulating that knowledge at the higher levels should depend on knowledge at the lower levels in the appropriate sense, and that knowledge at the lower levels should be epistemologically "closer to the world." The levels should also be chosen such that a significant portion of the agent's functionality is implemented by each level, and such that the ordering of the levels is consistent with the principles of reactive, hierarchical control. The lowest levels should be fast, and time-critical reactions should not require the propagation of knowledge to the higher, slower levels.

There may be many sets of levels that meet these constraints, and the issue of how a set of levels ought to be chosen is interesting in its own right. One approach would be empirical and domain-specific; a system designer could collect a large set of representations from a given domain and divide them into appropriate levels either by inspection or by trial and error. A rather different, non-empirical approach is to devise a set of domain-<u>independent</u> levels based on *a priori* knowledge representation considerations. This latter approach was used in the development of APE, but both of these approaches are compatible with the supervenience architecture.

The levels for APE were chosen on the basis of an analysis of the general structure of event-related knowledge, as exposed in the philosophical and psychological literature. Section 8.2.1 surveys the philosophical and psychological sources that influenced the choice of specific levels. Section 8.2.2 summarizes the set of levels used in APE, and Section 8.2.3 describes, in more concrete terms, the types of knowledge represented at each level.

8.2.1 Philosophical and Psychological Evidence

The knowledge in a dynamic-world planning system is likely to be primarily about events in the world and about actions that an agent can perform. The representation of events and actions is a topic with a considerable history in philosophy (see [Davis 1979] for a survey). Goldman's theory of human action [Goldman 1970] was mentioned previously in Chapter 4 because his concept of level generation is similar to the concept of supervenience. Goldman distinguishes four sub-species of the level generation relation: causal generation, conventional generation, simple generation, and augmentation generation. Thalberg, in his rival "component" account, distinguishes several classes of potential components of a given event, including: "purely relational" consequences, causal consequences, and conventional consequences [Thalberg 1977].

The classes of consequences offered by Thalberg and the types of level generation specified by Goldman have certain similarities. Both accounts make use of a category based on the existence of "conventions." In addition, both accounts have a special place for knowledge about causality. Thalberg's "purely relational" consequences and Goldman's "simple generation" are also very close; referring to the case in which A level generates A', Goldman says that, "In simple generation the existence of certain circumstances, conjoined with the performance of A, ensures that the agent has performed A'" [Goldman 1970, 26].

The relative autonomy of some domain of causal reasoning is evidenced by the volume of literature, across many disciplines, on theories of causation and of causal modelling. This literature ranges from the presocratic Greek philosophers to current work in developmental psychology, computational simulation, and theoretical AI. Piaget, for example, discusses a range of conceptions of causality in the developing child (see, e.g., [Piaget and Inhelder 1969]). Shoham provides a survey and discussion of philosophical theories of causation and develops a computational account based on temporal logic [Shoham 1988]. Further discussions of reasoning about causation in AI systems may be found in several papers in [Brachman and Levesque 1985], [Horn 1990], and [Weld and de Kleer 1990, Chapter 9].

Causal relations may be related to various other types of knowledge, but “theories of causation” generally provide systems or sets of principles by which statements about causality can be analyzed and manipulated independent from other descriptions of the world. For example, a theory of causation might assert that causation is irreflexive; this would sanction the denial of the statement “*A* caused *A*” regardless of the status of *A* vis-à-vis other relations. While there is considerable controversy about the proper analysis of causal relations, the existence of a separable domain of causal reasoning is well entrenched in the Western intellectual framework.

Similar remarks may be made for conventional reasoning, although the literature is not as extensive. An analysis of conventional behavior as a separable domain of reasoning can be found in [Shwayder 1965]. The “personal intelligences” in Gardner’s theory of “multiple intelligences” are responsible for the cognitive aspects of culturally and conventionally constrained interactions [Gardner 1983]. The papers in [Gellatly, Rogers, and Sloboda 1989] document several recent trends in the study of socially organized cognition in cognitive psychology. Aspects of certain speech acts (for example, promising, apologizing, protesting, etc.) might also be seen as defining a domain of conventional reasoning [Austin 1975]. If so, then current AI work in speech act theory (see, e.g., several papers in [Cohen, Morgan, and Pollack 1990]) would also attest to the existence of a conventional domain.

These considerations suggest at least two general, domain independent levels of event representation: a <u>conventional</u> level and a <u>causal</u> level. In contrast to “simple” or “purely relational” descriptions of events and actions, causal and conventional descriptions rely on large bodies of knowledge about the world. Causal descriptions (for example, "The heat made the milk boil.") rely on access to some theory of the causal structure of the universe, be it a theory from modern physics or of some common sense "naive physics" [Hayes 1985]. Likewise conventional descriptions (for example, "Ginger snubbed Oliver") rely on some theory of human conventions and social practices.

I will also argue that conventional knowledge supervenes on causal knowledge, and thus I place the conventional level above the causal level in APE. Conventional facts depend on causal facts in the manner specified in the definition and discussion of supervenience in Chapter 4. For example, if Ginger charmed Oliver by dropping her handkerchief in his path, then "Ginger charmed Oliver" (conventional) depends on "Ginger dropped her handkerchief in Oliver's path" (causal). Had the handkerchief not been dropped, Oliver would not have been charmed (at least not for the specified reason).

This is not to say that conventional and causal facts cannot be commingled such that some dependency relations go the other way; for example, "Ginger's snub raised Oliver's blood pressure." This does not refute the claim that non-conventional causal statements are "closer to the world" than are conventional statements. Facts about conventions don't refute facts about the causal structure of the world, and this is the essence of the supervenience relation. Ginger's claim that there was no snub, merely a lapse of memory, wouldn't lead us to send the sphygmomanometer in for repairs. The supervenience of the conventional on the causal is similar in spirit to the supervenience of the moral on the physical; this is the relation that brought "supervenience" into the philosophical lexicon in the first place (see Chapter 4).

The supervenience architecture specifies that only the "minimal" levels are connected directly to sensors and effectors. In APE a single level, called the perceptual/manual level, is dedicated to all sensor and effector interaction. All other levels in APE supervene on the perceptual/manual level, which is "closest to the world" of all.

The perceptual/manual level also has a basis in the psychological literature. Gardner's "bodily-kinesthetic intelligence" is the intelligence that provides "the capacity to work skillfully with objects, both those that involve the fine motor movements of one's fingers and hands and those that exploit gross motor movements of the body" [Gardner 1983, 206]. Gardner defends the designation of bodily-kinesthetic intelligence as an intelligence, and argues for the modularity of bodily-kinesthetic intelligence on neurophysiological grounds.

Jaques, Gibson, and Isaac, in a book entitled Levels of Abstraction in Logic and Human Action, provide five different five-level systems, based on a variety of considerations ranging from developmental and organizational psychology to an analysis of the relations between symbolic logic and human action [Jaques, Gibson, and Isaac 1978]. These level systems are similar to the levels chosen for APE in several respects, but the most obvious point of contact is at the perceptual/manual level. In summarizing the first levels of the various systems, they write:

> The relationship with the object of activity is direct and concrete. Work is by the immediate process of operation upon the object, usually referred to as skill. This skill is based upon the 'touch and feel' of the situation . . . [Jaques, Gibson, and Isaac 1978, insert]

The volume of psychological literature on "motor behavior" and "motor learning" also speaks to the existence of a perceptual/manual level; see, for example, [Cratty 1973] and [Heuer, Kleinbeck, and Schmidt 1985].

A large class of robotic tasks can be accomplished without explicit manipulation of causal or conventional knowledge. Some such tasks may nonetheless require access to abstract (i.e., non-perceptual/manual) knowledge about the world.[33] APE therefore includes intermediate levels of representation that contain spatial and temporal models of the world, as well as procedures for reasoning about such models.

Although spatial and temporal reasoning processes are similar in certain respects, they have generally been discussed separately in psychology, and modeled independently in AI systems. A large body of literature exists on the modeling of spatial relations (for example, for cartographic knowledge bases), and there has also been progress in the development of qualitative spatial representations (e.g., [Mukerjee and Joe 1990]). Several papers on AI approaches to reasoning about shape and space may be found in [Weld and de Kleer 1990, Chapter 8]. Gardner posits a modular "spatial intelligence" in addition to the bodily-kinesthetic and personal intelligences mentioned above. The conception of space in children is an important topic in developmental psychology as well (see, e.g., [Piaget and Inhelder, 1967]). A range of topics on the psychology of spatial representation and spatial skills, including correlations to neurophysiological data, are surveyed in [Liben, Patterson, and Newcombe 1981] and [Potegal 1982].

The study of time and temporal cognition dates at least to Parmenides in the fifth century B.C., and plays a central role in many branches of modern philosophy and psychology. On the independence of space and time there is a range of opinions; Gale makes a case for independent treatment, stating, "the disanalogies between 'here' and 'now' are profound: space and time are radically different" [Gale 1971, 85].[34] Temporal representation and reasoning are well developed topics in psychology (see, e.g., [Levin and Zakay 1989], and the literature on the representation of temporal

[33]Proponents of theories of "pure reaction" would disagree; see Section 2.2.

[34]Reichenbach also objects to the parallel treatment of space and time, but argues, on the basis of the theory of relativity in physics, that "time is more fundamental than space" [Reichenbach 1958, 112]. Depending on one's interpretation of "fundamental," this view may or may not be compatible with the supervenience of the temporal on the spatial, for which I argue below.

relations in AI (much of which derives from [Allen 1984]) is also large and varied. Papers on qualitative reasoning about temporal relations may be found in [Weld and de Kleer 1991, Chapter 4].

The relation between temporal and spatial knowledge can also be seen as one of supervenience; temporal descriptions of the world <u>supervene</u> on spatial descriptions of the world. Statements about temporal relations holding in the world are often statements about changes, non-changes, coincidences, etc., of <u>spatial relations over time</u>. For example, "The cat was on the mat when the doorbell rang." This sentence asserts a temporal relation that holds <u>in virtue</u> of the holding of a spatial relation.

The supervenience of temporal upon spatial knowledge also accords with the greater immediacy with which spatial relations are perceived. Situated agents perceive spatial relations "now" while temporal relations (except for simultaneity) are computed after the fact; hence the spatial relations are epistemologically "closer to the world." It might be objected that this is an argument that temporal knowledge supervenes on <u>all non-temporal</u> (as opposed to just <u>spatial</u>) knowledge. The concept of "space" should be construed broadly to meet this objection; i.e., the entire structure of the immediate, atemporal, physical world is thought of as comprising the "spatial" structure of the world. With regard to non-temporal aspects of the higher (causal and conventional) levels of representation, comments similar to those made for the commingling of causal and conventional knowledge apply. The difference between the spatial and temporal levels is that the temporal level introduces explicit reasoning about time; temporal representations exist at <u>all</u> levels of the system, but explicit reasoning <u>about</u> time emerges only at the temporal level.

8.2.2 Summary of Levels in APE

The levels chosen for APE are, from lowest to highest: perceptual/manual, spatial, temporal, causal, and conventional (see Figure 7). A linear order is imposed on the levels, although the supervenience architecture allows any partial order. This is of little theoretical consequence, as supervenience is a transitive relation. The pragmatic effect of the linear order is that certain details of implementation are simplified (for example, the construction of translation systems), but representations must sometimes be passed through intervening levels.

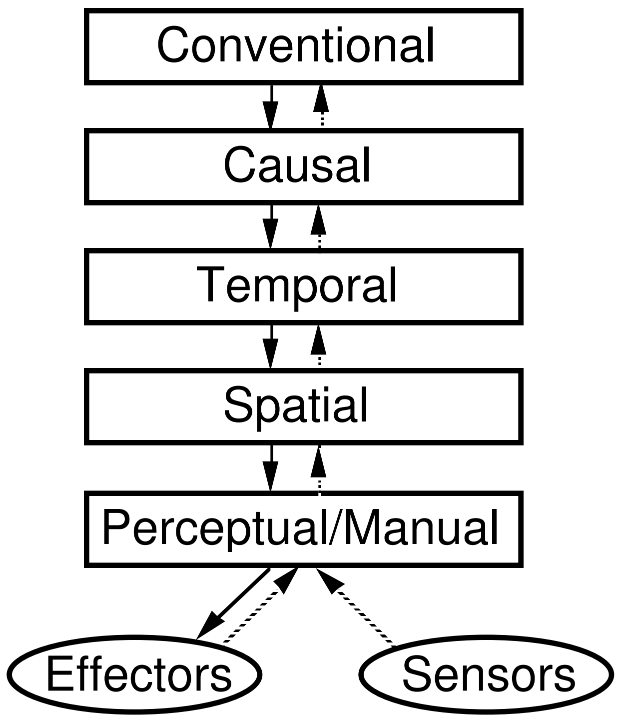

Figure 7. The specific levels of APE.

Supervenience claims are often couched in phrases of the form "no *x* change without *y* change"; for example, for psycho-physical supervenience, "No mental change without physical change."[35] The supervenience claims entailed by the choice of levels in APE can be summarized as follows:

[35]See the discussion in Chapter 4.

• No conventional change without causal change. That is, the causal structure of the an agent's world-view determines its conventional structure. The belief in a conventional difference between two situations can always be traced to a belief in a causal difference; for example, if action *x* is believed to be polite in one situation and impolite in another, then the beliefs about the causal relations between *x* and other events must also differ between the two situations.

• No causal change without temporal change. That is, the temporal structure of an agent's world-view determines its causal structure. The belief in a causal difference between two situations can always be traced to belief in a difference in the occurrence of non-causally described events over time; for example, if *x* is believed to cause *y* in one situation but not in another, then beliefs in the temporal, non-causal circumstances of *x* must also differ between the two situations.

• No temporal change without spatial change. That is, the spatial structure of an agent's world-view determines its temporal structure. Any difference between two situations vis-à-vis the temporal relations that are believed to hold can be traced to a difference vis-à-vis believed spatial relations, where "spatial" is construed broadly as "physical, non-temporal"; for example, if *x* is believed to precede *y* in one situation but not in another, then the situations must differ with respect to non-temporal beliefs as well.

• No spatial change without perceptual/manual change. That is, the perceptual/manual structure of an agent's world-view determines its spatial structure. Any difference between two situations vis-à-vis believed spatial relations can be traced to a difference vis-à-vis believed perceptual/manual relations; for example, if *x* is believed to be on top of *y* in one situation but not in another, then the situations must also differ with respect to beliefs about actions and/or perceptual processes.

It is again useful to make a comparison to Brooks's subsumption architecture; the similarities with respect to choice of levels are interesting, even though there are significant differences on other points. Brooks has used the following “levels of competence”:

0 Avoid contact with objects (whether the objects move or are stationary).
1 Wander aimlessly around without hitting things.
2 “Explore” the world by seeing places in the distance which look reachable and heading for them.
3 Build a map of the environment and plan routes from one place to another.
4 Note changes in the “static” environment.
5 Reason about the world in terms of identifiable objects and perform tasks related to certain objects.
6 Formulate and execute plans which involve changing the state of the world in some desirable way.
7 Reason about the behavior of objects in the world and modify plans accordingly. [Brooks 1990, 9]

While there are differences between these levels and the levels used in APE—for example, objects may have identities even at the lowest level in APE, while object identities arise only at level 5 in Brooks’s scheme—there are also striking similarities. Levels 0 and 1 provide competencies similar to those of APE’s perceptual manual level, levels 2 and 3 add spatial competencies, levels 4 and up introduce explicitly temporal concepts, and level 7 appears to perform some sort of causal reasoning.

8.2.2 Types of Knowledge at Each Level

APE provides knowledge representation and programming facilities with which knowledge structures and procedures can be implemented at each level, but it is in the application of APE to a given domain that the capabilities of each level are actually realized. Hence, while I will state, for example, that temporal projection is performed at the temporal level, this will only be true in instances of APE for which the appropriate temporal-level operators have been defined. In this sense APE can be thought of as a five-level “shell” for programming dynamic-world planning systems.

The perceptual/manual level represents events as simple sensory reports and as operators for effector manipulation. Perceptual/manual representations might be rendered in English as, “I see a sock at position 12, 17, 26, at time 3:45 PM,” or “To move forward, I should gain control of the main body motor, run it forwards, and check to see that I’ve moved.” The only “reasoning” at this level is the composition of such sequences of perceptual and manual tasks. The coupling of perception to action that results is in some ways analogous to simple “reflex arcs” in animals. Note that although spatial and temporal data is <u>present</u> at this level, reasoning <u>about</u> space and time occurs elsewhere.

The spatial level contains structures that organize perceptual data with respect to spatial relations. Synthetic spatial relations such as <u>on</u>, <u>above</u>, <u>near</u> etc., arise at this level, and hence representations such as “The cat is on the mat” become expressible. At the perceptual/manual level this would be represented only as the conjunction of two sensory reports: one describing the position of the cat, and one describing the position of the mat. Representations about events and actions that <u>change</u> the status of spatial relations (for example, “I put the banana peel into the trash can at 12:15 PM”) are expressible here, but the temporal and causal dimensions of such changes are not the subjects of spatial level reasoning. Operators for complex spatial reasoning (for example, path planning) also reside here.

The temporal level augments spatial representations with temporal relations, allowing for reasoning about deadlines and temporal projection. New concepts at this level include synthetic temporal relations such as <u>before</u>, <u>during</u>, and <u>after</u>, and other concepts specific to temporal reasoning such as <u>expect</u>, <u>delay</u>, and <u>late</u>. While every level represents time in some manner (and certainly every level acts <u>in</u> time), only at the temporal level do representations such as “The casserole has been in the oven too long” arise. Lower levels simply tag representations with temporal information, and perform actions as quickly as they can. It is at the temporal level that the system can reason <u>about</u> time, scheduling actions to meet deadlines. This reasoning might be accomplished, for example, with a temporal logic or a “time map management” system [Dean 1985].

The causal level contains representations that embody the agent’s conception of the causal structure of the world. This may include causal rules and causally deduced facts such as “I prevented the human from falling.” At this level concepts that are defined on the basis of causal models of the world are introduced; for example, <u>cause</u>, <u>effect</u>, <u>boil</u>, and <u>melt</u> . While the representation of a melting of an ice cube might be represented at the temporal level as a conjunction of observations or predictions about the relative sizes of the cube and of the resulting puddle over time, at the causal level there may be a <u>melt</u> concept that is integrated into a theory of temperatures, state changes, etc.

The conventional level contains knowledge about facts that are true by convention; for example, that a certain hand gesture is a signal for a turn, or that a dirty sock “belongs” in the dirty clothes hamper. Procedures at this level should embody some theory of social rules and interaction. For example, the knowledge that it is <u>impolite</u> to leave the room when being addressed might be encoded here.

These levels partition the system’s knowledge (both declarative and procedural) into a set of connected, communicating systems, each of which exhibits expertise at a particular level of “abstraction” from the world. It is important to note that this is <u>not</u> to say that the representations characteristic of one level are entirely absent from all other levels. As mentioned earlier, for example, temporal representations exist at <u>all</u> levels of the system, even though explicit reasoning <u>about</u> time occurs only at the temporal level. Similar statements can be made for the other levels as well.

8.3 Specialization of the Supervenience Architecture

The supervenience architecture, described in Chapter 7, specifies a general design for multilevel dynamic-world planning systems. In this section I describe the details by which the general design is implemented in APE.

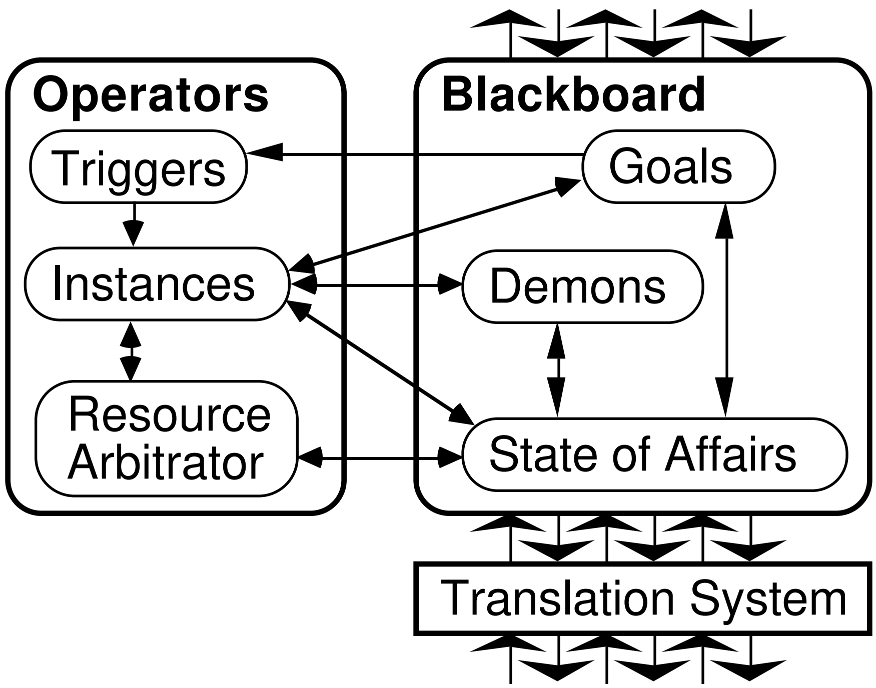

Figure 8. A single level of APE in detail.

Figure 8 adds detail to the picture of the individual levels of the supervenience architecture for APE (cf. Figure 4). The problem-solving (planning) procedures are implemented as <u>operators</u> that are instantiated in a goal-driven fashion. Operators in the APE are similar to the <u>knowledge</u> <u>sources</u> of traditional blackboard systems; i.e., “diverse and independent programs” [Erman et al. 1980, 218]. Within the literature of dynamic-world planning, they are similar to the <u>Knowledge</u> <u>Areas</u> of PRS [Georgeff and Lansky 1990], [Georgeff and Ingrand 1989]. There are differences in detail, but APE operators, like PRS knowledge areas, are procedural behavior specifications that allow for incremental expansion and execution. The architecture of APE at a single level also bears comparison to the architecture of PRS systems,[36] and I have taken the liberty of borrowing some of the terminology used in describing PRS; for example, "triggers" and "invocation conditions" (see below). APE's operator formalism is described in detail in Section 8.5.

Structures called triggers handle the instantiation of operators, allowing for multiple instantiations of the same operator (for different goals) and for instantiations of different operators working on the same goal. Each operator has an invocation condition which is matched against goals on the blackboard. When such a match is detected, the corresponding trigger checks if the given operator has already been instantiated for the given match. If not, the trigger fires, generating a new instance of the operator. The new instance becomes an active procedural element of the given level of the planning hierarchy, running asynchronously and in parallel with the other operator instances at that level.[37] The instance runs until it succeeds, fails, or simply terminates. An operator may specify different blackboard modifications and "cleanup procedures" for each of these outcomes.

Operator instances have priorities that can be passed through goals; if a goal is posted with a high priority then that priority is inherited by operators triggered by the goal. These priorities are used by resource arbitrators to handle resource conflicts amongst operator instances. The resource arbitrators store information about resource usage on the blackboard, and this information can be accessed and reasoned about by operator instances. The resource arbitrators suspend lower-priority operator instances competing for a given resource and resume the highest-priority suspended instance

[36]PRS was also "designed to allow several instantiations of the basic system to run in parallel" [Georgeff and Lansky 1990]. This raises the possibility of implementing the supervenience architecture as a collection of PRS systems, one per level.

[37]The parallelism in APE is simulated; see section 8.8.

when the resource becomes available. Operators may perform special actions upon suspension and upon resumption.

APE supports two types of goals: <u>teleological</u> goals, which are goals for the achievement of states of affairs, and <u>teleoepistemic</u> goals, which are goals for the attainment of knowledge. Other researchers have described a wide range of goal types, and have discussed the possible relations between various types of goals (e.g., [Lesser et al. 1989], [Wilensky 1983]). Such work can provide a basis for extending goal representation in APE, but at present only teleological and teleoepistemic goals are supported. The mechanisms provided for goal representation in APE are explained in greater detail in Section 8.4.

APE also provides for "active" elements on the blackboard called <u>demons</u>. Demons provide an intuitive mechanism for monitoring the state of affairs and for reporting changes to operator instances. (See Section 8.7 for further discussion on strategies for monitoring in APE). The use of demons to inform operator instances that pending goals have been satisfied also helps to simplify the simulation of parallelism in APE (see Section 8.4).

The lowest level of APE is responsible for communication with sensors and effectors, and hence its structure is somewhat different (see Figure 9). There is no translation system since there are no lower-level planning systems with which to communicate. The operator instances call sensor and effector functions directly, and sensor data is returned to the calling operator instances for processing. Knowledge ascertained from sensor input can then be entered by the operator instances into the state of affairs.

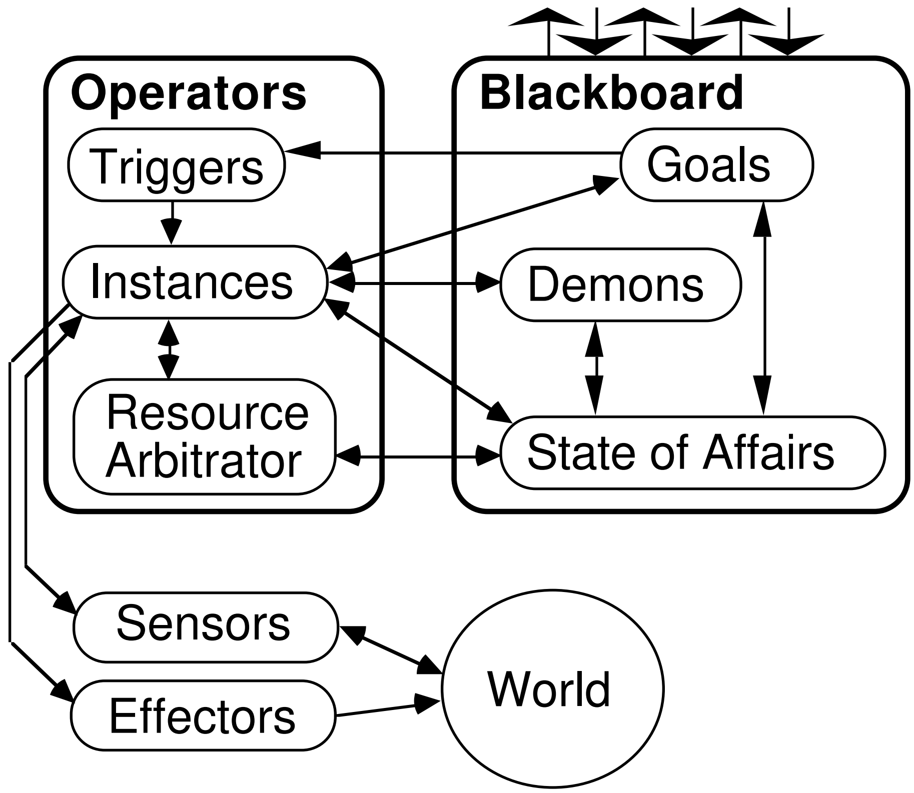

Figure 9. A detailed look at the bottom level of APE.

8.4 Knowledge Representation

Warning: Use At Your Own Risk

> This machine is not a certified knowledge server: it is not sound, it is not complete, and there is no guarantee that it will produce an answer in a tractable amount of time. There is in fact no precise characterization of its running time at all. How any being in its right mind could use such a service is utterly beyond me.
>
> (signed)
> F. Lou deKoop
> Knowledge Representation General
>
> [Davis 1991]

The supervenience architecture requires a knowledge representation system to handle transactions on the blackboards at each level. Different knowledge representation systems could be used at each level of the system; this is consistent with the design principles of the supervenience architecture. For simplicity, APE uses the same knowledge representation system at each level, although each level makes use of the system in different ways. The knowledge representation system described in the remainder of this section was developed specifically for APE. Of course, the literature of knowledge representation research is large and well developed, and it is likely that other, more complex knowledge representation systems could be integrated with APE as well.

Aside from the representation of facts about the world, APE also requires the representation of demons, priorities and resources, and two types of goals. The types of goals are <u>teleological goals</u> (or <u>achievement goals</u>) and <u>teleoepistemic goals</u> (or <u>knowledge goals</u>). The distinction is between goals for <u>bringing about</u> some state of affairs (teleological) and goals for <u>finding out</u> whether some state of affairs obtains (teleoepistemic). The distinction is important whenever it is possible for an agent to have partial information about the state of the world.

Agents with incomplete knowledge must often perform inferences and/or actions to determine the state of the world. Teleoepistemic goals are the goals that trigger such inferences and actions, thereby generating activity that updates the system's <u>beliefs</u>. In contrast, teleological goals are goals for achieving conditions in the <u>world</u>. An example of a teleoepistemic goal is the goal of "knowing if the cat is on the mat," while an example of a teleological goal is the goal of "getting the cat onto the mat."

A similar distinction is expressible in PRS through the use of the "!" and "?" goal-forming operators:

> The expression (!*p*), where p is some state description (possibly involving logical connectives), is true of a sequence of states if *p* is true of the last state in the sequence; that is, it denotes those behaviors that *achieve p*. . . . Similarly, (?*p*) is true if *p* is true of the first state in a sequence—that is, it can be considered to denote those behaviors that result from a successful *test* for *p*. [Georgeff and Lansky 1990, 731][38]

The distinction between teleological and teleoepistemic goals is particularly important for multilevel architectures. The knowledge in <u>any</u> dynamic-world planning system might be incomplete on account of insufficient sensing and/or inference. But in multilevel systems such incompleteness might also be due to insufficient inter-level communication; a given level might not have the belief that the cat is on the mat only because the belief has not yet been propagated from the level at which the belief was deduced.

The use of demons allows for intuitive specifications of <u>monitors</u> that inform operator instances of changes in the state of affairs. Demons are pieces of code that can be placed on the blackboard to "watch" for the occurrence of certain events, and to "fire" (execute their code) when such events occur. Demons are also used in APE for communication to goal-posting operator instances about the achievement of their goals. When an operator instance posts a subgoal it can also post a <u>call-back</u> demon that will run when the subgoal is achieved. Until the call-back, the components of the operator that require prior achievement of the subgoal can be suspended, and it is not necessary for the operator instance to poll the blackboard to check on the status of the subgoal. Demons in APE are attached directly to the blackboard items that they are monitoring, and they require no run-time except when fired. The firing of demons is handled by the blackboard-update procedures; when a blackboard-update indicates that the goal has been achieved, the demon will be called, informing the posting operator instance that it can proceed.

[38]PRS also includes a "#" goal-forming operator that allows for the expression of <u>maintenance</u> goals, for which there is no analog in APE.

The purpose of the priority and resource mechanism is to allow resource arbitrators to make reasonable decisions about which operator instances to suspend when resource conflicts are detected. Priorities are attached to goals, and they are inherited by the operator instances triggered by the goals. When an operator instance posts a subgoal it can either pass its own priority to the subgoal or it can attach some other priority. APE resources are single entities (represented as symbolic atoms) that can have only single users,[39] and APE priorities are symbolic atoms arranged in a partial order. The set of priority symbols is a parameter of the system; examples to date have required only `background`, `normal`, and `urgent`. The usage of a resource is recorded on the blackboard with a sentence (see next paragraph) such as `(using :arm-control grab-23)`. Resource requests that have not yet been fulfilled are recorded with sentences such as `(waiting :arm-control grab-24)`. The invocation of the resource arbitrator is described in Section 8.5. Previous planning systems have included complex mechanisms for the representation of resources and/or priorities (e.g., [Wilkins 1988]); such mechanisms could be incorporated into future versions of APE.

Each item on the blackboard contains a <u>sentence</u> that expresses the semantic "content" of the item. Sentences are lists of symbolic atoms, possibly including the variable `:?`. For example, a belief that the cat is on the mat might be recorded on the blackboard with an item having the sentence `(on cat mat)`. Variables in sentences are to be read as "something"; for example, the sentence `(on cat :?)`, if believed, would mean that the system believes that the cat is on something. The system may or may not have additional beliefs about what it is that the cat is on; for example, `(on cat mat)` or `(on cat car)`. The items on the blackboard are stored in a discrimination net that uses the sequences of atoms in sentences for discrimination

[39] Two operators working on goals related as subgoals count as a "single user" in this context.

keys, and there can be only one item on the blackboard for any given sentence.

Each blackboard item also contains three tags, each of which has, at any given time, a single value from the set {+, –, ?}. The three tags are epistemic, teleoepistemic, and teleological. The epistemic tag indicates the status of the blackboard item's sentence with regard to belief. A "+" value for the epistemic tag means that the system believes that the sentence is true, a "–" value means that the system believes that the sentence is false, and a "?" value means that the system has no belief regarding the truth or falsity of the sentence. The teleoepistemic tag indicates the status of the blackboard item's sentence as a teleoepistemic (knowledge) goal. A "+" value for the teleoepistemic tag means that the system wants to know if the sentence is true, a "–" value means that the system wants to know if the sentence is false, and a "?" value means that the system has no goal regarding knowledge about the truth or falsity of the sentence.[40] The teleological tag indicates the status of the blackboard item's sentence as a teleological (achievement) goal. A "+" value for the teleological tag means that the system wants to make the sentence true, a "–" value means that the system wants to make the sentence false, and a "?" value means that the system has no goal regarding achievement of truth or falsity of the sentence.

	epistemic	teleoepistemic	teleological
+	believed	discovering	achieving
–	rejected	debunking	abolishing
?	doubted	ignoring	neglecting

Figure 10. Query functions.

[40]The attributions of propositional attitudes (e.g., "believes" and "knows") to the system in this paragraph, and in the remainder of the dissertation, are not intended to make or to imply metaphysical claims. They are used for convenience; it is simpler to say "wants to know if *x*" than "has a goal of changing the epistemic tag of *x* to '+' via the application of knowledge-oriented (as opposed to achievement-oriented) operators."

Figure 10 shows the functions that are provided for querying the status of tags corresponding to a given sentence. When they are called with variable-free sentences their behavior is straightforward: the single blackboard item that matches the sentence is examined, and a boolean value is returned indicating whether or not the specified tag has the specified value. If no match is found, then a new blackboard item with default tag values (all "?") is created. When variables are present they can be interpreted as universally quantified or as existentially quantified: Queries for positive ("+") and negative ("–") values are interpreted as existentially quantified, and queries for neutral ("?") values are interpreted as universally quantified. For example, `(achieving '(on cat :?))` will be true if there is <u>any</u> goal of getting the cat onto <u>anything</u>, while `(neglecting '(on :? :?))` will be true only when there are <u>no</u> goals for getting anything onto anything. Variants of these query functions (for example, queries for positive values that are interpreted universally) are also provided, and additional variants can be added without difficulty. A range of miscellaneous query functions have also been implemented as needed. For example, the function `get-all-rejected-ground` returns a list of all blackboard items without variables that match the given sentence, and that have negative epistemic tags.

	<u>epistemic</u>	<u>teleoepistemic</u>	<u>teleological</u>
+	believe	discover	achieve
–	reject	debunk	abolish
?	doubt	ignore	neglect

Figure 11. Command functions.

Figure 11 shows the command functions that are provided for changing the status of tags corresponding to a given sentence. In contrast to the query functions, the behavior of command functions may be complex even for variable-free sentences.

This is because the system maintains the consistency of the knowledge base with respect to sentences that contain variables. For example, suppose that the system does not initially believe `(on :? mat)`, but that the program then executes `(believe '(on cat mat))`. The system will set the epistemic tags of both `(on cat mat)` and `(on :? mat)` to "+". Additional complications arise here as well; for example, if a sentence is already believed then a call to `achieve` it does not change the teleological tag; it merely succeeds and returns immediately.

The commands `discover`, `debunk`, `achieve`, and `abolish` are all goal-posting commands. Goal-posting commands may include arbitrary lisp call-back functions that are installed as demons in the matching blackboard items. Call-back functions are functions of zero arguments that may be specified as lambda forms. When the sought-after tag states come to pass, the demons are executed and disposed of. The blackboard-update procedures are responsible for the firing of demons; whenever a change is made to a tag state, the update procedure fires any relevant demons. Since the demons are attached directly to their associated blackboard items, this process is quite efficient. As an example, one might issue the command `(discover '(on cat mat) :call-back #'(lambda () (print "The cat is on the mat!")))`. This command would post a goal to determine if the cat is on the mat, and the call-back function would print `"The cat is on the mat!"` as soon as the goal was satisfied. A lambda form used as call-back function is defined in the lexical environment in which the command is issued. Hence the lambda form may refer to lexical variables; for example, `(let ((exclamation "Hey!")) (discover '(hot hand) :call-back #'(lambda () (print exclamation))))`. Several operators may post the same goal and thereby attach demons to the same blackboard item.

Goal-posting commands may also include priorities. For example, one might issue the command `(abolish '(on cat stove) :priority :urgent)`. The priority of each matching blackboard item is set to be the maximum of the specified priority and the pre-existing priority of the item. If no priority is specified then the priority of the posting operator instance is used.[41]

APE's knowledge representation system meets the requirements of the architecture with few unnecessary complications. In spite of this intentional simplicity, the system has a few interesting properties. For example, the teleoepistemic and teleological queries correspond in a certain sense to meta-reasoning capabilities of more sophisticated systems: one can query not only about beliefs, but also about the status of the system's quest for beliefs. Possible enhancements to the knowledge representation system are described in Chapter 10.

8.5 Operators

APE operators resemble the knowledge sources of more traditional blackboard systems in that they are complex, independent, goal-triggered procedures. They also share features, such as add and delete lists, with traditional planning operators. The procedural aspects of APE operators are specified as Petri nets [Peterson 1981], allowing for comparison to other net-based procedural specifications in dynamic-world planning (e.g., [Drummond 1990], [Georgeff and Lansky 1990]). In this section the specification of operators is described in detail. A simple operator from the HomeBot domain is shown in Figure 12; more elaborate examples, along with graphic representations, are provided in Chapter 9.

---

[41]Each blackboard item actually has two priorities, one for teleological goals and one for teleoepistemic goals. For example, one can be simultaneously `discovering` if *x* with a priority of `:normal`, and `achieving` *x* with a priority of `:urgent`.

```
(defoperator do-laundry
  :level :causal
  :for (achieving (clean :?))
  :filters ((believed `(clothing ,?1)))
  :steps ((TAKE-TO-MACHINE
           (achieve `(on ,?1 laundry-machine)
                    :call-back
                    #'(lambda () (finish TAKE-TO-MACHINE))))
          (TAKE-TO-BED-WHEN-DONE
           (achieve `(after laundry-done on ,?1 bed)
                    :call-back
                    #'(lambda ()
                         (if (believed `(smells-bad ,?1))
                           (fail)
                           (succeed))))))
  :initially (TAKE-TO-MACHINE)
  :transitions ((TAKE-TO-MACHINE :> TAKE-TO-BED-WHEN-DONE))
  :on-success ((believe `(clean ,?1)))
  :on-failure ((believe '(probably-broken laundry-machine)))
```

Figure 12. A simple APE operator for doing laundry.

Operators are defined with the `defoperator` macro, which takes a name and the following keyword arguments: `level`, `for`, `variables`, `filters`, `steps`, `transitions`, `initially`, `on-failure`, `on-success`, `on-suspension`, `on-resumption`, and `on-termination`. Operator names are symbols, each of which must be unique.

The `level` argument specifies the level of the supervenient planning hierarchy at which the operator will be defined; its value must be `:perceptual-manual`, `:spatial`, `:temporal`, `:causal`, or `:conventional`.

The `for` argument specifies the invocation condition of the operator. The invocation condition is a goal that will trigger the instantiation of the operator. For example, an instance of the `do-laundry` operator in Figure 12 will be created whenever there is a goal of achieving `(clean :?)`. The present system allows only goals of the form `(<query> <sentence>)`. An obvious extension is to allow conjunctive and/or disjunctive goals to be specified in the `for` argument.[42] Conjunctive goal capabilities can be "faked" to some extent by posting goals that represent conjunctions; for example, `dog-fed-and-floor-swept`. This maneuver is not entirely satisfactory, but it suffices for the examples we have implemented to date.

When an operator is triggered the blackboard item that matches the `for` form is used to set the values of local variables of the form `?0`, `?1`, `?2`, etc. The first element of the sentence of the matching item is assigned to `?0`, the second to `?1`, and so on. These variables can be accessed by various other elements of the operator. For example, if the `do-laundry` operator is triggered by a goal to achieve `(clean shirt)`, then the references to `?1` in the `filters`, in the `steps`, and in the `on-success` form will all refer to `shirt`.

The `variables` argument specifies an additional list of local variables that can be set and accessed within the operator. The variables are local to each instance of the operator, and persist for the life of the instance. The `do-laundry` operator requires no such variables (but see Chapter 9 for further examples).

The `filters` argument specifies a list of arbitrary lisp forms, all of which must evaluate to a positive (non-null) value if the operator is to be instantiated. The numbered variables are bound prior to the evaluation of the filters, and the values of these variables are typically tested for the applicability of the operator. For example, the `do-laundry` operator will not be instantiated for the goal of achieving `(clean floor)` because the system will not have a belief that `(clothing floor)`.

The heart of an operator—the description of what it does—is specified in the `steps`, `transitions`, and `initially` arguments, which collectively describe a Petri net. Petri nets were chosen because of their capability for specifying asynchronous

[42]This is allowed, for example, in PRS [Georgeff and Lansky 1990].

parallel activity:

> In the Petri net model, two events which are both enabled and do not interact may occur independently. There is no need to synchronize events unless it is required by the underlying system which is being modeled. When synchronization is needed, it is easy to model this also. Thus, Petri nets would seem ideal for modeling systems of distributed control with multiple processes executing concurrently in time. [Peterson 1981, 35]

Petri nets are composed of two types of nodes, called places and transitions, and directed links that connect places to transitions and transitions to places (but never places to places or transitions to transitions). A place may contain tokens, in which case it is said to be marked. While the number of tokens in a place is usually significant in Petri net theory, in APE this number is limited to 1 or 0. Hence the state of a place in APE can always be described simply as marked or unmarked. The places from which links are directed at a given transition are called the input places of the transition, and the places to which links are directed from the transition are called the output places of the transition. A transition is enabled and can fire when all of its input places are marked. When a transition fires it removes a token from each of its input places (leaving them unmarked) and adds a token to each of its output places (marking those that were not marked already). The sequence of transition firings is nondeterministic; enabled transitions may fire at any time, and the sequence in which they fire may not be related to the sequence in which they became enabled. Sequential effects are obtained by structuring the net such that earlier transitions must fire in order to enable later transitions.

The basic Petri net model does not include mechanisms for modeling events that take time [Peterson 1981, 37–38], or for attaching code to events in the net.[43] In APE code is attached to places in the net, and execution (and hence delay) occurs during

[43]The introduction of timing information into Petri nets is an area of active research. See [Holliday and Vernon 1987] for an overview of some recent techniques.

the marking process. When a transition fires it does not immediately mark its output places; it merely queues the code from each output place for execution. When the code from a given place finishes execution, the corresponding place is actually marked and may thereby contribute to the enablement of other transitions.

The places of the Petri net are specified in the `steps` argument to `defoperator`. Each place is specified as a list containing the name of the place (a symbol) and the code for the place. The code for a place must execute `(finish <place-name>)` to inform the Petri net executor that it has completed, and that the corresponding place should be marked. The code for an place may also include `(succeed)`, `(fail)`, or `(terminate)`. Each of these results in the termination and deallocation of the operator instance; they differ only in the code that is executed during termination (see below). Whenever an operator instance terminates, all other operator instances working on subgoals of the operator also terminate. Operator instances are also terminated when their triggering goals are revoked.

The code in operator steps may also include calls to the resource-handling functions `start-using` and `stop-using`. These functions cause annotations about resource usage to be made on the blackboard and invoke the resource arbitrator if there is a conflict. If an operator is already using the `:arm-control` resource and a second operator executes `(start-using :arm-control)`, then the resource arbitrator will be invoked and the operator with the lower priority will be suspended. Priority ties are broken randomly. Resources are level-specific; for example, a resource at the perceptual/manual level can be used only by perceptual/manual operators.

The transitions of the Petri net are specified in the `transitions` argument to `defoperator`. Each transition is specified as a list of input place names, the special symbol "`:>`", and a list of output place names. The `do-laundry` operator has only one transition, with the single place `TAKE-TO-MACHINE` as an input place, and the single place `TAKE-TO-BED-WHEN-DONE` as an output place.

The initial marking of the Petri net is specified in the `initially` argument to `defoperator`. The code at each of the places named in `initially` is queued for execution when the operator is instantiated. Each place is marked when its code finishes executing.

The remaining arguments to `defoperator`—`on-failure`, `on-success`, `on-suspension`, `on-resumption`, and `on-termination`—each take arbitrary lisp code (lists of forms that will be evaluated within an implicit `progn`). Each is evaluated in the conditions suggested by its name, and code within each may refer to local variables specified in the `variables` argument, as well as to the numbered variables corresponding to elements of the triggering goal sentence.

The `on-success` argument corresponds to the add and delete lists of traditional STRIPS-like planning operators—it specifies the effects of the operator when all goes well. The on-success form typically includes calls to `believe` (corresponding to elements of a STRIPS operator's add list) and/or calls to `reject` (corresponding to elements of a STRIPS operator's delete list). In the `do-laundry` operator there is one such effect: when the operator succeeds it will assert a belief that the previously dirty object is now clean.

The `on-failure` argument specifies effects of the operator when all does <u>not</u> go well. Of course, some operators may be capable of failing in a variety of ways, and it may be difficult to specify <u>the</u> appropriate modifications to the blackboard in the wake of failures of such operators. Nonetheless, such specifications are often appropriate. The `do-laundry` operator can fail only by noticing, near the end of its task, that the piece of clothing still smells bad.[44] In such a case it is reasonable to

[44]The operator can "go wrong" in various other ways as well. For example, it can get "stuck" while a subgoal remains unachieved, and it can be terminated by the revocation of its triggering goal.

assume that the laundry machine is broken, and the operator asserts a belief to that effect.

Code in the `on-termination` argument is evaluated whenever the operator instance terminates, whether termination is caused by success, failure, revocation of the triggering goal, or an explicit call to `(terminate)`. The `on-termination` argument is the appropriate place for "clean up" procedures that must be executed before an operator instance is deallocated. For example, the low-level `roll` operator uses the `:body-motion` resource. The `on-termination` argument for `roll` includes the form `(stop-using :body-motion)`, ensuring that the `:body-motion` resource is released whenever an instance of `roll` that had seized `:body-motion` is terminated.

Code in the `on-suspension` argument is evaluated whenever the resource arbitrator suspends the operator to resolve a resource conflict. It can be used to specify "suspension clean up" procedures and to note the context in which suspension occurred. Recall the Wesson Oil problem of Section 2.3, in which a woman is frying chicken when her child gets hurt. The child screams and the woman stops cooking to take the child to the hospital, but only after turning off the stove. This example could be encoded in APE by using a priority of `:urgent` for the goal of taking the child to the hospital, and a priority of `:normal` for the goal of frying chicken. The screaming child would trigger a resource conflict, since both the hospital task and the frying task require use of the woman's body. Since the hospital goal would have a higher priority, the frying task would be suspended. Inclusion of the form `(abolish '(turned-on stove))` in the `on-suspension` argument of the `fry-chicken` operator would cause actions for turning off the stove to be triggered when the woman suspends the frying task.[45] The `on-resumption` argument can be used to

But failure *per se* occurs only as a result of the execution of the `(fail)` form. Note also that the choice of success or failure in `do-laundry` depends on the proper maintenance of `(smells-bad` *x*`)` sentences by other elements of the system.

[45]Inclusion of goal-posting commands in `on-suspension` forms is actually a bit more

reverse the effects of the on-suspension argument; for example, to post a goal to (achieve '(turned-on stove)). The on-resumption argument can also be used check the context in which the operator is resumed, possibly triggering immediate success or failure; for example, (if (believed '(soggy chicken)) (fail)).

8.6 Translators

The between-level translation systems of the supervenience architecture are implemented in APE as collections of units called translators. All communication between levels in APE is effected by the translators, and the asymmetry of the supervenience relation is reflected in the design of the translator system. Translators facilitate the “upward” movement of world knowledge, and the “downward” movement of goals; this is the implementational analog to the “assertions up, assumptions down” communications regime in the formal characterization of supervenience. Translators are small data structures containing lisp code that are similar in some respects to APE operators. In this section the specification of translators is presented in detail.

The laundry-pile translator, specified in Figure 13, assumes that the locations of objects are represented at the perceptual/manual level using sentences of the form (at <object-type> <object-name> <x> <y> <z>). The spatial level concept of a “laundry-pile” describes a collection of pieces of laundry at the same $<x, y>$ coordinates, with one at the height of the floor (with z=1) and one somewhat higher (with z=3). The locations of laundry-piles are recorded at the spatial level with sentences of the form: (at laundry-pile <x> <y> <z>).

---

complicated, since the same resources that caused the suspension might also be required for the operators that would achieve the newly posted goals. One solution is to use a “super-urgent” priority—in the given example the goal of abolishing (turned-on stove) should have the highest priority, the goal of going to the hospital should have the next highest priority, and the goal of

```
(deftranslator laundry-pile
  :xlevel :below-spatial
  :demand (discovering (locations laundry-pile))
  :commands ((discover `(locations laundry)))
  :supplies ((laundry-on-floor
              (get-all-believed `(at laundry :? :? :? 1)))
             (laundry-up-high
              (get-all-believed `(at laundry :? :? :? 3))))
  :test (and (eql (nth 3 (sentence laundry-on-floor)) ;; same x value
                  (nth 3 (sentence laundry-up-high)))
             (eql (nth 4 (sentence laundry-on-floor)) ;; same y value
                  (nth 4 (sentence laundry-up-high))))
  :products ((believe
              `(at laundry-pile ,(nth 3 (sentence laundry-on-floor))
                                ,(nth 4 (sentence laundry-on-floor))
                                1)))
  :advertisements ((believe
                    `(looked for laundry-pile at ,(the-time)))))
```

Figure 13. A translator for laundry-piles.

Translators are defined with the `deftranslator` macro, which takes a name and the following keyword arguments: `xlevel`, `demand`, `commands`, `supplies`, `test`, `products`, and `advertisements`.

The `xlevel` argument specifies the level beneath which the given translator is to be situated; its value must be one of `:below-spatial`, `:below-temporal`, `:below-causal`, or `:below-conventional`.

The remaining arguments specify the function of the translator with respect to an economic metaphor of communication. The economic model was chosen because it provides an intuitive metaphor for supervenience. The metaphor also provides mnemonic names for the various elements of the translator data structure (demand, supplies, products, etc.).

Each translator is situated between two planning levels, one higher and one lower. The higher level is conceptualized as an economic consumer and the lower level is conceptualized as an economic producer. The translator, on the basis of a perceived demand at the higher level, issues a set of production commands to the lower level.

The translator then collects supplies from the lower level and forms the combinations of them that pass the test into products. The products are then provided to the higher level, possibly accompanied by advertisements.

Note that an economic consumer depends on producers for material needs, and that producers are, with respect to their materials and products, "closer to the world." Producers and consumers are also independent entities, each of which has expertise that the other lacks. Consumers can procure only those things that producers produce, but the products are generally conceptualized differently by producer and consumer. While consumers can "demand" that certain products be produced, the availability of materials, feasibility of construction, etc., depend on factors to which the producer has more immediate access and more certain knowledge. In all of these respects, the economic relation of producer to consumer is similar to the relation of a lower level to a higher, supervenient level in a supervenient planning hierarchy.[46] These observations led to the development of the economic metaphor for knowledge transactions across levels of APE.

Once instantiated, a translator instance runs "continuously," issuing commands at the lower level and massaging low level supplies into high level products. In this respect APE's translators are similar to the translation systems of Jackendoff's multilevel cognitive model:

> For each set of correspondence rules that provides a translation from one level of representation L_i to another level L_j, there is a *translation processor* that automatically and compulsively translates whatever information is available at level L_i into information of level L_j, whenever such translation is possible. [Jackendoff 1987, 258]

The continuity of translation is simulated. Whenever processor time is allocated to a translator it re-posts its `commands` to the lower level, collects `supplies`, filters out

frying chicken should have the lowest priority. Such complications do not arise when non-goal-posting commands (e.g., `believe`, `reject`) are used in `on-suspension` forms.

supply combinations that don't pass the `test`, creates `products` and posts `advertisements` at the higher level.

Like operators, translators are triggered by goals on the blackboard. The `demand` argument of `deftranslator` is similar to the `for` argument of `defoperator` in that it sets up a trigger for the translator. Whenever the `demand` condition is met the translator is instantiated; for example, an instance of the `laundry-pile` translator will be created whenever there is a goal for discovering the locations of piles of laundry. Numbered variables (`?0`, `?1`, etc.) are set to the corresponding elements of the matching sentence and may be referenced by other elements of the translator.[47] As with operators, multiple instantiations of the same translator are allowed, but not for the same goal.

Translators also effect downward communication through their `commands`. While it is possible to include arbitrary lisp forms in the `commands`, the intention is that this capability be used for posting goals to the lower level. For example, the `laundry-pile` translator posts a goal to discover the locations of pieces of laundry at the lower, perceptual/manual level. Sometimes it may be appropriate to pass knowledge structures to lower levels that encode goals implicitly. For example, a path may be passed from the spatial to the perceptual/manual level, implicitly encoding several repositioning goals. Representation of the path in a non-goal form might be preferable due to details of the knowledge representation system.

The `supplies` argument to `deftranslator` is specified as a list of `(<supply-type> <generating-form>)` pairs. Each generating form should return a list of values of the given supply type. If multiple generating forms are provided for a single supply type then they are evaluated in an implicit `progn` and the value of the last form is used. In the `laundry-pile` translator there are two types of supplies: blackboard

[46]C.f. Part II of this dissertation.

items representing pieces of laundry on the floor (with z=1) and blackboard items representing pieces of laundry “up high” (with z=3).

Each translator generates supply tuples from the lists of supplies that it obtains. A supply tuple contains one supply element of each type, and supply tuples are generated for all possible such combinations. Each supply tuple is a “data set” upon which the `test` is run; supply tuples that pass the `test` are then used for the generation of `products`. In the `laundry-pile` example, if there are two pieces of laundry on the floor (say, `shirt-17` and `poncho-3`) and three pieces of laundry at z=3 (say `sock-32`, `towel-2` and `jeans-5`), then supply tuples will be generated for all of the combinations shown in Figure 14.

laundry-on-floor	laundry-up-high
shirt-17	sock-32
shirt-17	towel-2
shirt-17	jeans-5
poncho-3	sock-32
poncho-3	towel-2
poncho-3	jeans-5

Figure 14. Supply tuples for the `laundry-pile` translator example.

The `test` argument to `deftranslator` takes a single lisp form that is evaluated once for each supply tuple. The tuple’s supply element of a given type can be referenced within the `test` with the symbol that names the supply type. The `test` in `laundry-pile` specifies that the x and y coordinates of the `laundry-on-floor` and the `laundry-up-high` must match exactly. Suppose that the <x, y> coordinates of the pieces of laundry are as follows: `shirt-17` = <1, 1>, `poncho-3` = <2, 2>, `sock-32` = <2, 2>, `towel-2` = <1, 2>, `jeans-5` = <1, 1>. Then the supply tuples that will pass the `test` are <`poncho-3`, `sock-32`> and <`shirt-17`, `jeans-5`>.

The `products` and `advertisements` arguments to deftranslator each take lists of arbitrary lisp forms. The `products` forms are evaluated once for each supply tuple that passes the `test`, with blackboard modifications being made to the blackboard at the higher level. Within `products` the elements of the supply tuple can again be referenced with supply type symbols. In the `laundry-pile` example two sentences about the locations of laundry piles would be asserted at the spatial level: one at <*x*, *y*, *z*> = <1, 1, 1> and one at <*x*, *y*, *z*> = <2, 2, 1>. The `advertisements` forms are also evaluated at the higher level, but they are evaluated only once per activation of the translator, after all of the `products` forms have been evaluated. In the example a single form would be posted to the spatial level upon each activation, indicating the time at which the translator completed running.

8.7 Strategies for Monitoring

An important aspect of any dynamic-world planning system is the ease with which the system designer can incorporate monitoring tasks into the planning process. Monitoring tasks are tasks that continuously (or at least repetitively) check for the truth of some condition, triggering an appropriate reaction when the condition becomes true. APE provides several mechanisms for monitoring change in the environment. In this section I will briefly describe monitoring with demons, monitoring with looping steps, and monitoring with translators. These three mechanisms may also be combined to allow for more complex monitoring strategies.

In the supervenience architecture, only the lowest levels have direct access to sensors, and hence to the world. At the lowest levels, one can speak of the monitoring mechanisms as monitoring change "in the world," although it is more precise to speak of them as monitoring change "in the world-as-sensed." For higher levels the monitoring is less direct still; one can monitor changes in the state of affairs at that level when monitoring with demons or with looping steps, or at the immediately lower levels when monitoring with translators.

Demons are the simplest and most intuitive mechanism for monitoring the state of affairs. To monitor with a demon, one posts a teleoepistemic goal of discovering (or debunking) the existence of a condition of interest. All goal-posting blackboard commands allow for the specification a call-back function that is evaluated when the goal is achieved (see Section 8.4). Any actions that should result from the firing of the monitor can be included in the code of the call-back function. The reaction to a monitored condition may be a simple command to `finish` the step containing the monitor, thereby allowing for the firing of other transitions. But call-back reactions may also be arbitrarily complex; for example, if the coordinates of a known laundry pile are `x`, `y`, and `z`, then the step in Figure 15 can be used to monitor the position of the laundry pile, and to print a message and to cause the operator to fail if the pile moves.

```
(monitor-laundry-pile
  (debunk `(at laundry-pile ,x ,y ,z)
   :call-back #'(lambda () (print "The laundry moved!") (fail)))
```

Figure 15. Monitoring with a demon.

When demons are used for monitoring, the responsibility for actually discovering if the monitored condition holds falls on other operators and translators in the system. The demon posted in Figure 15 will fire when the belief that there is a laundry pile at the given location is debunked, but the demon itself performs no actions or inferences to bring about such a change of belief. The posting of the demon simply sets up a reaction to the discovery that the condition holds.

If a demon's condition holds when it is posted, the call-back function will be evaluated immediately. Otherwise, the demon will remain dormant until the condition is asserted. Other operators or translators may be instantiated by the system in an attempt to achieve the demon's goal. For example, the goal posted in Figure 15 might trigger a `watch-laundry-pile` operator with a `for` argument of `(debunking (at laundry-pile :? :? :?))`. Such a `watch-laundry-pile` operator would presumably trigger actions and/or inferences that would cause `` `(at laundry-pile ,x ,y ,z) `` to be rejected when the laundry pile in question is moved. This would cause the demon posted in Figure 15 to fire, printing the message `"The laundry moved!"` and triggering the failure of the operator that contains the step. It is also possible that other operators or translators would reject `` `(at laundry-pile ,x ,y ,z) `` serendipitously, as a side effect of some other process.

```
(monitor-laundry-pile
  (debunk `(at laundry-pile ,x ,y ,z)
   :call-back #'(lambda ()
                  (if (believed `(near laundry-pile ,x ,y ,z))
                    (print "The laundry moved!")
                    (print "The laundry disappeared!"))
                  (fail)))
```

Figure 16. Monitoring with a demon that checks other items on the blackboard.

Because demons are attached to individual blackboard items, demon-based monitoring works best when the condition to be monitored is expressed as a single sentence on the blackboard. It is possible, however, for a demon to check for other items on the blackboard when fired, allowing for demon-based monitoring of more complex conditions. The demon posted by the step in Figure 16 prints one of two messages, depending on the presence or absence of a second sentence on the blackboard. This technique becomes cumbersome, however, when the conditions of interest become more complex.

When it is necessary for an operator to monitor for a complex set of conditions, it is generally simpler to introduce an explicit monitoring loop in the operator's Petri net. This also allows the full power of the operator construction formalism to be utilized in structuring complex monitoring behaviors. There is a price for this added flexibility; in contrast to demons, looping step monitors require CPU time even when they have yet to detect the condition of interest.

The code fragment in Figure 17 shows part of an operator with a looping step monitor. The corresponding fragment of the operator's Petri net is graphed in Figure 18. The `check-for-pile-change` step checks for a variety of beliefs about laundry piles at and adjacent to the known laundry pile location. If the known pile is unmoved, but an additional pile is detected in an adjacent location, then a notation is made on the blackboard that there exists a laundry "mess." If there is no longer a pile at the original location, but there is a pile at an adjacent location then the operator assumes that the original pile was moved—if there is an explicit statement on the blackboard that such movements are unacceptable, then the operator fails. If there is no longer a pile at the original location, and there is not a pile at an adjacent location then the operator makes one of two assumptions, depending on the existence of other notations on the blackboard—if humans are believed to be home then the operator assumes that someone else took care of the laundry and the operator succeeds; otherwise it fails. The `check-if-care` and `check-for-pile-change` steps form a loop, allowing the `check-for-pile-change` step to run repeatedly. The loop can be turned off and on by other steps in the operator, or by other operators at the same level; asserting a belief in `(care-about pile-change)` turns the loop on, while rejecting such a belief turns the loop off.

Demons and looping step monitors provide mechanisms for monitoring the state of affairs at a given level, but access to other levels can be accomplished only with the aid of translators. In this context translators can be viewed as monitors; the translator runs continuously, monitoring the lower level for a condition (expressed in its `supplies` and `test`) and reporting to the higher level (via its `products` and `advertisements`) when the condition is met.

```
:level :spatial
:steps ...
(watch-for-new-piles
 (discover '(locations laundry-pile))
 (discover `(adjacent laundry-pile ,x ,y ,z))
 (finish watch-for-new-piles))
(care-about-pile-change
 (believe '(care-about pile-change))
 (finish care-about-pile-change))
(check-if-care
 (discover '(care-about pile-change)
          :call-back #'(lambda () (finish check-if-care))))
(check-for-pile-change
 (cond ( ;; pile unmoved, adjacent pile present also
        (and (believed `(adjacent laundry-pile ,x ,y :?))
             (believed `(at laundry-pile ,x ,y :?)))
        (believe `(laundry-mess ,x ,y :?))
        (finish check-for-pile-change))
       ( ;; pile moved to adjacent location
        (believed `(adjacent laundry-pile ,x ,y :?))
        (if (rejected '(acceptable laundry-movement))
          (fail)
          (finish check-for-pile-change)))
       ( ;; pile gone and nowhere close
        (rejected `(at laundry-pile ,x ,y :?))
        (if (believed '(home humans))
           (progn (print "Thanks for doing the laundry!")
                  (succeed))
           (fail)))
       (t ;; otherwise continue looping
        (finish check-for-pile-change))))
...
:transitions ...
(watch-for-new-piles :> care-about-pile-change)
(care-about-pile-change :> check-if-care)
(check-if-care :> check-for-pile-change)
(check-for-pile-change :> check-if-care)
...
```

Figure 17. A looping step monitor.

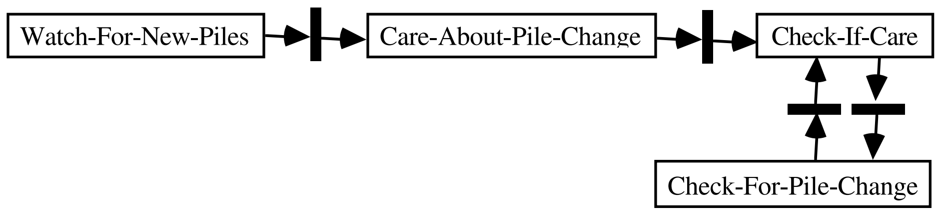

Figure 18. The Petri net fragment corresponding to the code in Figure 17.

The code fragment in Figure 17, in conjunction with the `laundry-pile` translator shown previously in Figure 13, serves to illustrate the use of translators in monitoring. The looping step monitor shown in Figures 17 and 18 monitors for appropriate changes *at the spatial level*. But in order for the changes to be noticed, they must first be propagated *to* the spatial level. The `laundry-pile` translator will be instantiated for this purpose (in response to the goal to `(discover '(locations laundry-pile))`), and will continually generate representations of laundry piles from perceptual/manual data. The looping step monitor is monitoring the spatial-level blackboard for laundry piles in certain locations, while the translator-based monitor is monitoring the perceptual/manual-level blackboard for new laundry piles that may be of interest *to the looping step monitor*. The `laundry-pile` translator can be thought of as monitoring the perceptual/manual level for the sake of the spatial level operator. This illustrates how cross-level (translator-based) monitoring can be combined with looping step monitoring; combinations with demon-based monitoring are possible as well.

Each of APE's monitoring techniques is appropriate in different circumstances. An operator can monitor a lower level of representation only with the aid of a translator. But while translators are well-suited for monitoring across levels, they cannot be used for monitoring *within* levels. Within a given level, the simplest and most efficient monitoring technique is provided by APE's demons. The monitoring of complex conditions can be accomplished with looping steps in an operator's Petri net, but such monitors may use CPU time even when the conditions of interest do not hold. The appropriateness of any monitoring method for a given task depends additionally on the details of the knowledge representation conventions employed by the programmer.

8.8 Parallelism: Theoretical and Simulated

One of the reasons that the blackboard model was chosen as a basis for the supervenience architecture was the promise of parallelizability inherent in the blackboard model. Parallel implementation has been a topic of interest in blackboard research since its earliest days (e.g., [Erman and Lesser 1975]). Recently, several researchers have been exploring techniques for actually realizing the promise of parallelizability [Bisiani and Forin 1989], [Corkill 1989], [Craig 1989], [Jagannathan 1989].

The supervenience architecture provides many opportunities for parallelism. APE was developed on an Apple Macintosh computer, and is written, for the most part, in generic Common Lisp. Since the Macintosh is a single processor machine, the parallelism inherent in the supervenience architecture is simulated by use of a scheduling mechanism. The simulation helps to illustrate that the program does not rely on sequence-related effects that would fail to obtain in a true parallel implementation.

The largest units of APE that may run in parallel are the planning levels. Each level is an independent planning system, receiving goals from above and world-knowledge from below. Each of the major components <u>within</u> each level may also run in parallel. Translation systems may run "continuously," moving goals and knowledge between adjacent levels. Blackboard maintenance procedures, including demons, may also run in parallel, assuming that some method is employed to avoid read/write conflicts and the like. The operator maintenance procedures, including triggers and resource arbitrators, may also run asynchronously and in parallel. While all of these processes are running, the operators themselves can be running, planning and initiating actions in the world. Of course, the number of active operator instances is dynamic, and this must be taken into account in simulating parallelism and in making optimal use of parallel hardware; the same holds for the number of translators, demons, etc.

The active operators at a given level may run in parallel with each other, and parallelism may also occur within operators. The Petri nets used in specifying operator steps allow not only for nondeterministic ordering of the execution of code, but also for the parallel execution of code. When a transition with two output places fires, the code from the two output places may be executed in parallel.

This aspect of APE, and of the supervenience architecture in general, is based on a very liberal notion of MIMD parallelism, in which the number of independent processors is essentially unlimited, and in which new processors can be allocated upon demand.

Parallelism in APE is simulated through the use of a data structure called a feap (short for "function heap") that implements an efficient timed function queue. Functions that are to be run in parallel are inserted into a queue, sorted by the amount of time that they have already run. When the feap is allocated processor time, the minimal element of the queue (i.e., the function that has run for the least time) is removed from the queue and executed. The execution of the function is timed, and the incremented run time of the function is used to reinsert the function into the queue at the proper position. A heap data structure is used to implement the queue, allowing for efficient computation of the removal and reinsertion operations (see, e.g., [Aho, Hopcroft, and Ullman 1983, 143–145]).

An <u>allocation</u> factor, used in scaling the recorded run time, can also be specified for each function in a feap. The allocation feature can be used either to simulate processors of various speeds, or to simulate (roughly) a prioritized scheduling system for a limited number of processors. If a given feap contains two functions, with allocations of 1 and 2, then the function with the allocation of 2 will, over time, receive approximately double the CPU time of the other function.

The use of feaps for the simulation of parallelism relies on the assumption that the functions to be run in parallel each run “for a little while” each time they are called, and that they will each be called several times before being removed from the feap. For example, if functions `fun1` and `fun2` are to be run in parallel, then they should be written such that multiple calls are required for each function to “do its job.” After `fun1` and `fun2` are inserted into feap `F`, successive calls to `(run-next F)` will execute whichever of `fun1` or `fun2` has thus far received the least time. The interleaving of calls to `fun1` and `fun2` will thereby simulate parallel execution. All of the functions in APE were written to conform to these assumptions. For example, the internal function that runs active operator instances executes, for each call, at most one `step` of one operator instance.

The top-level function of the APE system runs a loop that repeatedly executes and reinserts the minimal element of the main system feap. This feap contains only two functions, one of which runs the minimal element of the <u>levels</u> feap, and one of which runs the minimal element of the <u>translation</u> <u>levels</u> feap. The levels feap is composed of five functions, one for each planning level, each of which runs the minimal element of the feap for that level. The feap at each level contains a function for each active operator, and a single trigger function that performs operator instantiation. The translation levels feap is composed of four functions, one for each level of translators.

The feap at each translation level contains a function for each active translator, and a single trigger function that performs translator instantiation.

A problem that arises in using feaps for the simulation of parallel processing is that all functions on the feap get the same amount of processor time (modulo their allocation factors), regardless of whether or not they are getting any work done. In an environment with a large supply of processors, a "spinning" processor is a source of inefficiency but may not be a major concern. In an environment with only one processor, time wasted simulating spinning processors can be debilitating. For this reason modifications have been made to APE's feap mechanism to allow for intelligent "spin control." These modifications range from allowing for a minimum assumed run time (configurable by the user) to a mechanism whereby functions that are known to be spinning can be idled. Idled functions are stored on a secondary queue in the feap, and use no run time until they are explicitly unidled. This mechanism is used, for example, for operators that can make no progress until reactivated by demons on the blackboard.

The supervenience architecture allows for a great deal of parallelism in theory, but some of the practical issues of parallel implementation have yet to be addressed. Some of the issues (for example, asynchrony) have been addressed via simulation in the APE implementation, while others await further research. Future work may focus on improving the simulation or on porting the system to true parallel hardware (see Chapter 10).

---

[47]Such references are unnecessary in the `laundry-pile` translator, but see further examples in Chapter 9.

Chapter 9

HomeBot

9.1 Domain Description

Dynamic-world planning is difficult only in complex domains. When a domain is small enough and well enough understood, one can anticipate nearly all eventualities; in such cases even exponential searches may perform reasonably well. I have therefore applied APE within a relatively large and complex domain which is nonetheless familiar enough that it is easy to introspect about human behavior within it. This domain is suitable for practical dynamic-world planning research because of its size and complexity, and also because of the structural richness of the real-world events that can occur. While regimented domains such as Tileworld [Pollack and Ringuette 1990] are useful for certain quantitative studies, they do not give rise to high-level symbolic reasoning tasks with which real-world agents must contend. The Seaworld domain of [Vere and Bickmore 1990] and the Delivery Truck Domain of [Firby 1989] are both similar to the HomeBot domain in that realistic segments of the world are simulated to a level of detail intermediate between that of the real world and that of simple blocks world domains.

HomeBot is a simple one-armed, one-eyed robot that “lives” in a one bedroom apartment. HomeBot is expected to perform household tasks such as cleaning and protecting the human residents from danger. The apartment consists of six rooms (bedroom, kitchen, living room, bathroom and closets), quantized into 2,204 spatial locations in our current simulation.[48] A range of objects (furniture, pieces of laundry, etc.) may be present in the apartment at any given time. In addition, external occurrences may be triggered at any time. The doorbell may ring, an object may be moved, one of HomeBot's sensors may fail, etc.

The layout of the full HomeBot domain is shown in Figure 19. Figure 20 shows a snapshot of the user interface of the HomeBot system. The solid black square on the left side of the "World Simulator" window represents HomeBot, and the sparsely dotted square above it represents HomeBot's open hand. When HomeBot's hand is closed its pattern changes to stripes. The full set of objects shown in figure 19 is not present in the snapshot; objects can be created and added or deleted from the simulation as needed for a given problem. The snapshot shows a "popup menu" by which the experimenter can intervene in a simulation. The set of items in this menu is easily modified. The experimenter can also click on the various robot control buttons in order to move the robot in the midst of a simulation. HomeBot will not be aware of such manipulations until it performs sensing operations and infers that changes have occurred. The "Objects In View" window, created when the experimenter clicks on the "Sense Visual" button, shows the names of the objects currently in view of the robot. The experimenter can click on the name of an object to examine the object's properties in a separate window. The "Activity" window shows the names of the operators and translators currently running at each level. Question marks in the Activity window represent the activity of operator and translator instantiation mechanisms. The experimenter can click on the boxes on the right of the Activity window to change the allocations of the main feaps at each level (see Section 8.8), thereby effecting a simple form of dynamic load balancing during experiments. The system

[48]Our initial HomeBot world simulator quantized the world into over 70,000 spatial locations. The planning system performed well with the fine-grained simulation, but the world simulator itself ran slowly, making experimental use of the system impractical.

also includes a Petri net grapher and a scenario record/playback mechanism.

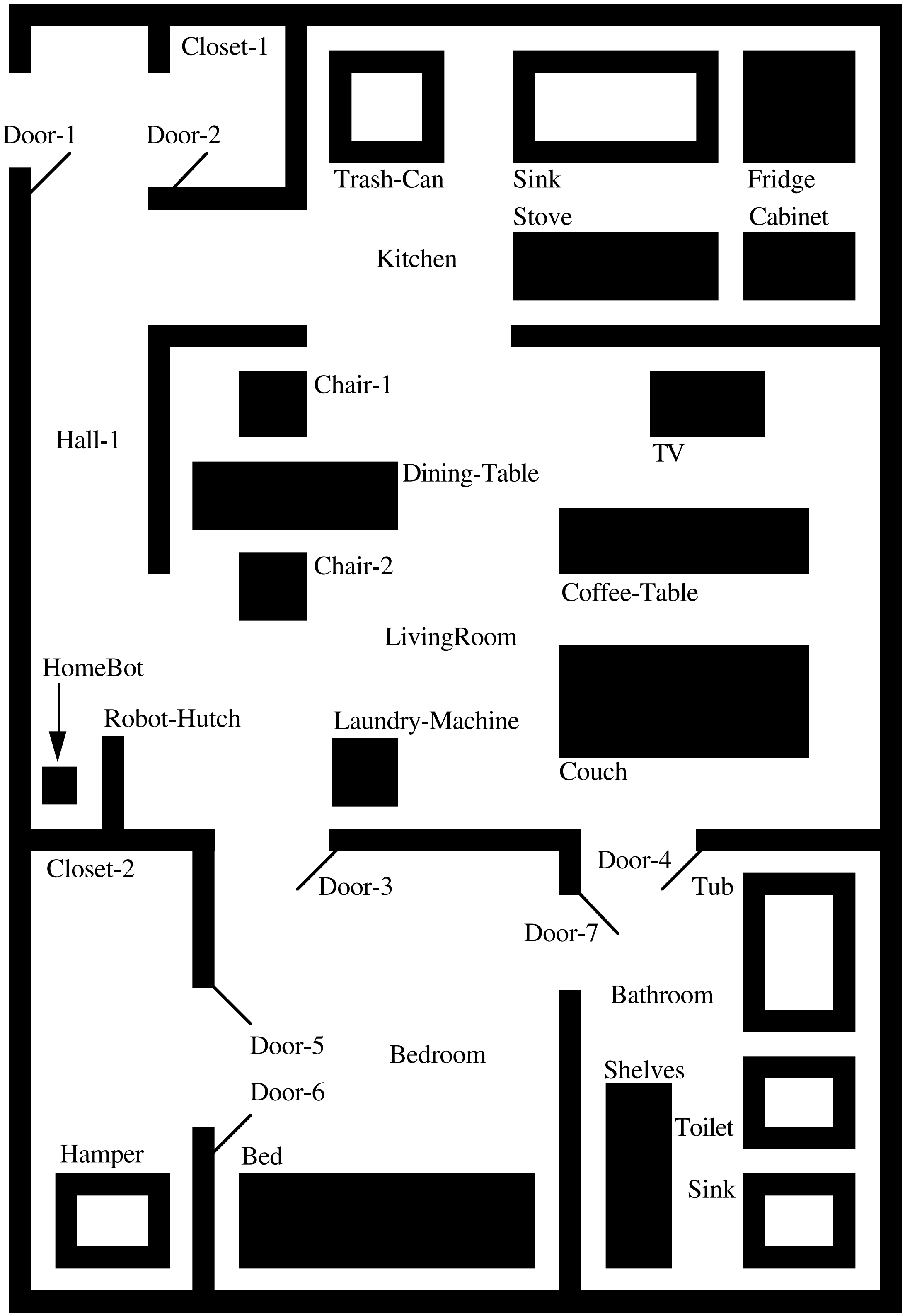

Figure 19. HomeBot's domain.

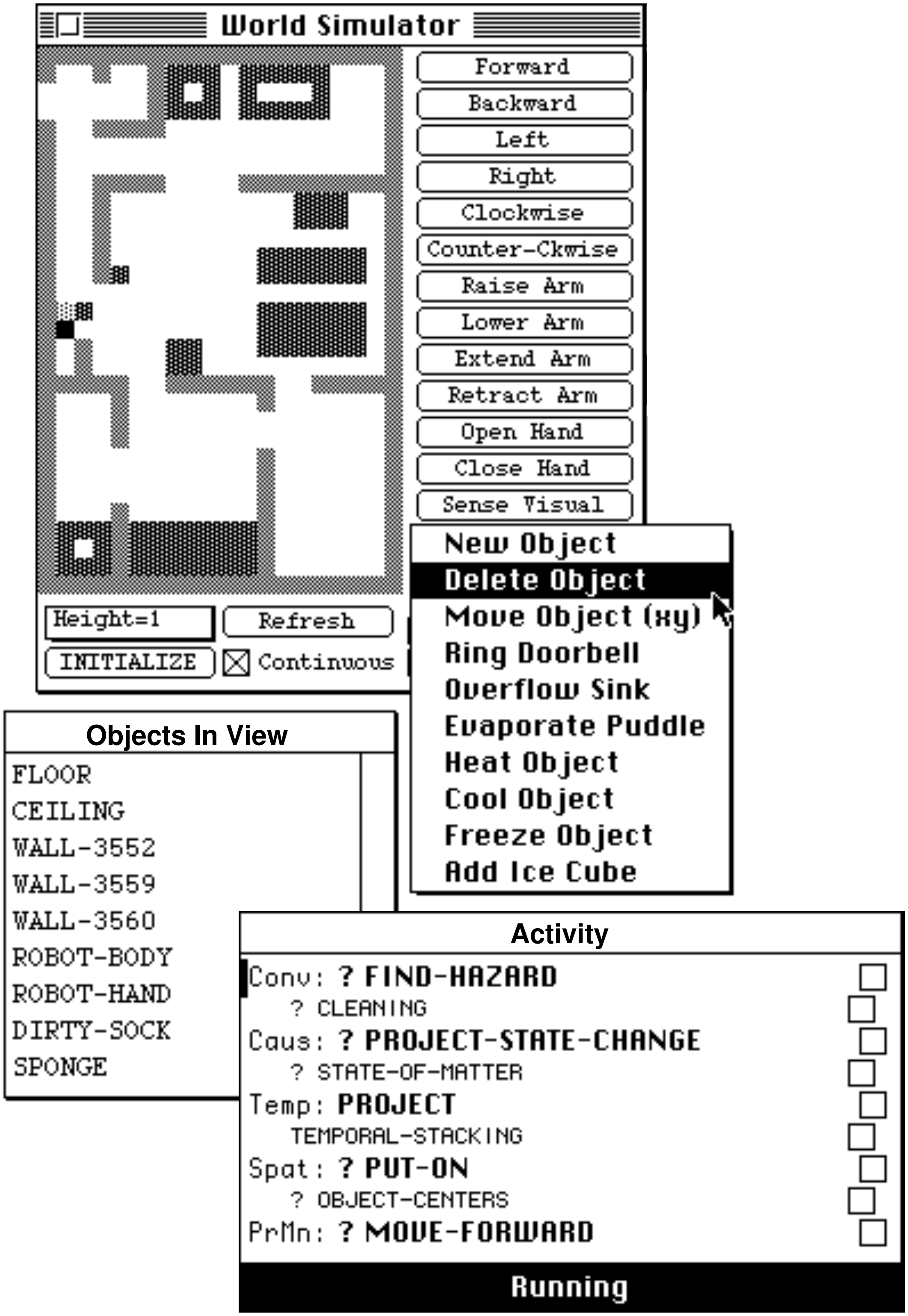

Figure 20. A snapshot of the user interface to the HomeBot system.

The HomeBot domain is a simulation; it is implemented as a program that runs asynchronously with APE. The top level loop of APE, described in Section 8.8, includes a call to the world simulator to update the state of the simulated environment. The agent interacts with the simulated world through the use of sensor and effector commands that approximate those currently used in actual robotic applications.

There are several benefits to working with simulators rather than with actual robots in the real world. Real robots are expensive, and working with them requires that a great deal of time be spent on issues peripheral to the topic under study. On the other hand, it is often difficult to determine if a simulator is simulating the "appropriate" aspects of the real world for properly testing a proposed system. Firby and Hanks discuss the trade-offs in using simulators, and argue that simulators can play an important role in the advancement of the field [Firby and Hanks 1987].

It is important that simulators be used correctly. If an agent is given arbitrary access to the simulator then it may be possible for it to "cheat" in various ways. For example, it might be possible for such an agent to know the locations of objects that it has not yet sensed, or for it to be automatically appraised of distant changes in the world. In order to avoid such possibilities a simulation-based planning system should include a clearly defined interface between the simulation and the agent's internal representations. This interface is defined for the HomeBot world simulator in Section 9.2 below.

The representation of the apartment, the position of objects within it, etc., is entirely distinct from HomeBot's internal representation of the world. HomeBot "knows" only that which it can infer from the data it receives through its sensors. This data is presented to HomeBot in real-time as sensory descriptions of pre-recognized objects. Although HomeBot is not required to perform object recognition from raw images, the domain forces system designers to handle some of the dynamism and unpredictableness of real-time sensory reports.

9.2 Application of APE

The application of APE to the HomeBot domain involves the implementation of the interface between the world simulator and the HomeBot's perceptual/manual level, and the design of operators and translators for the tasks that HomeBot is expected to handle. Operators and translators for the HomeBot domain are described in Section 9.3, in the context of the examples in which they are used.

The interface between world simulator and the HomeBot's perceptual/manual level consists of two functions: `sense` and `control`. All knowledge of the state of the world is acquired through the `sense` function, which takes a single `modality` argument. The valid sense modalities are: `:visual`, `:negative-visual`, `:body`, `:orientation`, `:hand-contents`, and `:tactile`. A call to `(sense :visual)` invokes a routine that simulates vision from HomeBot's perspective and returns a list of the objects that are within view. HomeBot has a limited visual range (approximately one third the length of the apartment) and the vision routines do not allow vision through objects. The `:negative-visual` modality returns a list of the visible coordinates at which there are no objects. The object descriptions returned by `(sense :visual)`, and by the other sense modalities as well, consist of the name, coordinates, color, material, and temperature of the sensed object.[49]

A call to `(sense :body)` returns a list of the objects that constitute HomeBot's body. Since the object descriptions include coordinates, this call can be used to locate HomeBot within the apartment. The `(sense :orientation)` command returns the

[49]APE operators that call `(sense :visual)` do not generally access the returned temperature value even though the simulator provides it; infrared sensing capability is not assumed.

direction (north, east, south, or west) that HomeBot is currently facing.

The `(sense :hand-contents)` command returns a list of the objects that HomeBot is carrying. This command is the principal means by which HomeBot can determine if a carried object has been dropped. Such information might also be inferred from visual data (for example, that the object is on the floor).

The `(sense :tactile)` command returns the temperature of HomeBot's hand. This is the principal means by which temperatures in the world are assessed.

The `control` function can be called with up to three arguments: a `body-part`, a `motion` specification, and a `direction`. The valid argument combinations are shown in Figure 21. In each case the simulator attempts to effect the specified motion without violating world constraints (for example, the solidity of walls). If such constraints would be violated then the simulator simply ignores the command; no direct feedback is provided for either success or failure. HomeBot must use the `sense` command, in conjunction with previous knowledge, to determine whether actions in the world have succeeded or failed.

body-part	motion	direction
:body	:forward	—
:body	:right	—
:body	:backward	—
:body	:left	—
:body	:rotate	:clockwise
:body	:rotate	:counter-clockwise
:arm	:extend	—
:arm	:retract	—
:arm	:raise	—
:arm	:lower	—
:hand	:open	—
:hand	:close	—

Figure 21. Valid argument combinations for `control`.

HomeBot's sense of time is provided by a system clock synchronized to the top-level loop of APE. On each passage through the loop the system introduces an updated sentence of the form `(homebot-time <integer>)` to the temporal-level blackboard.

9.3 Examples

In this section I provide examples of the operation of the HomeBot system. The HomeBot domain allows for a large range of tasks, and for a large and complex set of potential difficulties. The extent to which such tasks can be accomplished, and the extent to which such difficulties can be handled, depends on the specific set of operators and translators provided to the system. Section 9.3.1 describes a "basic" set of HomeBot operators and translators and provides detailed examples of their operation. Section 9.3.2 shows how HomeBot's functionality can be extended through the addition of new operators and translators.

HomeBot is a complex system, in which a large number of processes are generally running "at the same time," even in the simplest examples. The simulation of asynchronous parallelism, while helpful in assessing the prospects for future parallelization of the model, further complicates documentation and analysis of the system's behavior. Hence the examples are documented using a variety of devices, including "trace" output, code listings, graphs of operator Petri nets, operator activation graphs (generated from trace output), and English text.

9.3.1 Basic Examples

In this section I describe a basic set of operators and translators for HomeBot and provide detailed examples of their operation. Section 9.3.1.1 describes the operators and translators, and Sections 9.3.1.2, 9.3.1.3, and 9.3.1.4 provide the examples.

The first example, "HomeBot Feels Pain" (9.3.1.2), demonstrates that APE is capable of producing low-level reactive behavior. The pain reflex is implemented as a perceptual/manual process that requires no mediation by higher-level reasoning processes.

The second example, "HomeBot Navigates" (9.3.1.3), demonstrates how communication between the spatial and perceptual/manual levels allows HomeBot to make use of spatial reasoning procedures to solve navigation problems. In contrast to the pain example, the navigation example shows how APE allows for reactions that are computed on the basis of higher-level knowledge.

The third example, "HomeBot and the Ice Cube" (9.3.1.4), shows how HomeBot handles the Ice Cube problem described previously in Section 2.3. This example demonstrates the <u>integration</u> of deliberative and reactive behaviors across multiple levels of representation. Such problems were the motivation for the development of the supervenience architecture.

9.3.1.1 Basic Operators and Translators

This section describes the basic set of operators and translators that are used to exhibit APE's properties. The operators required for the examples of the next three sub-sections are listed in Figure 22, and the translators needed for these examples are listed in Figure 23. With these operators and translators installed, HomeBot's most functional subsystem is at the lowest level; the operators at the perceptual/manual level are complete enough to handle a fairly wide range of simple tasks. The spatial level is less complete, but still non-trivial. The temporal, causal, and conventional levels are quite simple; they are sufficient for a small set of examples, but no claims are made about their generality. In the following paragraphs I will sketch the capabilities of the operators and translators in this basic set; detailed expositions are provided in the context of examples in the subsequent sections.

The perceptual manual level organizes calls to `sense` and `control`, implementing base-level functionality for moving HomeBot around the apartment, for grasping and for releasing objects. The `roll` operator performs straight-line navigation, but complex path planning is performed at the spatial level. When the spatial level provides a path to the perceptual/manual level, the `roll-on-path` operator is used instead of the `roll` operator; the switching between `roll` and `roll-on-path` is accomplished with filter conditions. (See Section 9.3.1.3 for further detail.)

The spatial level performs path planning and navigational tasks, and is also responsible for computing and achieving spatial relations. The current capabilities are only for <u>stacking</u> relations, employing the spatial concept <u>on</u>. The system can be extended to perform analogous functions for other spatial concepts such as <u>in</u>, <u>around</u>, and so forth.

The temporal level performs scheduling tasks; for example, it passes down periodic sensing commands to ensure that HomeBot's world models are reasonably current. The `achieve-after` operator provides a primitive sequencing capability. It takes two conditions and issues a command for the achievement of the second only after the first is believed. The `project` operator provides a mechanism for asserting that conditions will become true in the future. It takes sentences of the form (at-time <time> <condition>) and creates demons, triggered by the system clock, that will assert beliefs in the appropriate conditions at the appropriate times.

The causal and conventional levels are quite rudimentary; they simply serve to show where and how causal and conventional knowledge can be integrated into an APE system.

```
Perceptual/Manual
look
feel
move-forward
move-backward
move-left
move-right
turn-clockwise
turn-counter-clockwise
raise-arm
lower-arm
extend-arm
retract-arm
open-hand
close-hand
pain-reflex
hand-pain-reflex
roll-on-path
roll
rotate
figure-direction
get-within-reach
get-rid-of-object
move-hand
grab

Spatial
move-object
put-on
find-space-above
navigate
plan-path
check-for-clear-path
compute-stacking-relations
compute-height-relations

Temporal
track-object-locations
maintain-body-awareness
check-time-since-looked
check-time-since-felt
achieve-after
project

Causal
do-laundry
project-state-change

Conventional
earn-praise
find-impropriety
find-hazard
sponge-trick
```

Figure 22. Basic HomeBot operators used in examples.

Below Spatial	Below Temporal
object-center	visual-scene
object-centers	spatial-visual-status
spatial-materials	tactile-scene
spatial-temperatures	spatial-tactile-status
closest-point	temporal-stacking
path	temporal-materials
object-motion	temporal-temperatures
hand-contents	
distance	Below Causal
motion-blockage	causal-stacking
relieved-blockage	delayed-achievement
free-space	state-of-matter
looking	
prmn-visual-status	Below Conventional
feeling	cleaning
prmn-tactile-status	conventional-stacking
	slip-hazard-avoidance
	slip-hazard-detection

Figure 23. Basic HomeBot translators used in examples.

HomeBot's basic set of translators follows a similar pattern, as shown in Figure 23. The lowest levels are the most complete, while the translators implemented at higher levels are targeted to a specific set of examples. The translators below the spatial level convert perceptual data into sentences about spatial level concepts such as object centers and distances. Paths and goals for which solutions can be achieved at the perceptual/manual level are passed down.

Several of the translators in the current HomeBot system exist solely to "relay" information between levels that must communicate, but that are not immediately adjacent. For example, when the temporal level schedules and then seeks to execute a sensing command, the spatial level must mediate the communication to the perceptual/manual level even when it has no other role to play. This suggests that the linear ordering of levels in APE, while simple and workable, should give way a more richly interconnected set of levels in future implementations.

9.3.1.2 HomeBot Feels Pain

This example demonstrates the low-level reactive capabilities of the APE system, as implemented in HomeBot's pain reflex.[50] At the start of each run the HomeBot system is initialized with a set of goals. Most of the system's goals are derived by goal-decomposition from the initial conventional level goal to `(achieve '(praise))`, but initial goals are posted at lower levels as well. The pain reflex is initialized by an initial goal to `(abolish '(painful :?))`, which is posted at the perceptual/manual level. This goal triggers the instantiation of the `pain-reflex` operator, shown in Figure 24.

```
(defoperator pain-reflex
  :level :perceptual-manual
  :for (abolishing (painful :?))
  :filters ((eq ?1 :?))
  :steps ((HAND-PAIN-REFLEX
           (doubt `(painful hand))
           (abolish `(painful hand) :priority :urgent
                    :call-back
                    #'(lambda () (finish HAND-PAIN-REFLEX))))
          (HEAD-PAIN-REFLEX
           (doubt `(painful head))
           (abolish `(painful head) :priority :urgent
                    :call-back
                    #'(lambda () (finish HEAD-PAIN-REFLEX)))))
  :initially (HAND-PAIN-REFLEX HEAD-PAIN-REFLEX)
  :transitions ((HAND-PAIN-REFLEX :> HAND-PAIN-REFLEX)
                (HEAD-PAIN-REFLEX :> HEAD-PAIN-REFLEX)))
```

Figure 24. The `pain-reflex` operator.

The `for` argument of the `pain-reflex` operator specifies that the operator will be triggered for any goals of abolishing sentences with two symbols, the first of which is `painful`. The `filters` argument specifies that the operator is only really appropriate

[50] I am not making a metaphysical claim that HomeBot actually <u>feels</u> pain, although related claims <u>have</u> been made and disputed with reference to cognitive models (see, e.g. [Dennett 1978]). Neither is HomeBot's pain reflex intended to model the full behavioral complexity of human or animal pain reactions. HomeBot's pain reflex is a low-level "reflex arc" for self-preservation; analogous reflex arcs in humans are associated with pain, and hence the use of the terms "feel" and "pain."

when the second symbol is the variable `:?`; this ensures that goals for abolishing specific pains like `(painful hand)` will be handled only by specialized operators. The `pain-reflex` operator contains two steps, each of which sets up a specialized pain reflex. The `HAND-PAIN-REFLEX` step first issues a command to doubt the painful status of the hand. This has the effect of setting the epistemic tag of the blackboard item for `(painful hand)` to "?", regardless of its previous state. It then posts a goal of abolishing `(painful hand)` with a priority of `:urgent`, and attaches a demon to the goal that will finish the `HAND-PAIN-REFLEX` step if and when `(painful hand)` is abolished. The `initially` argument specifies that `HAND-PAIN-REFLEX` will begin executing as soon as the operator is instantiated, and the first clause of the `transitions` argument creates a loop in the Petri net that will ensure that `HAND-PAIN-REFLEX` is re-executed each time it finishes (see Figure 25). The `HEAD-PAIN-REFLEX` step is analogous, and may run in parallel with `HAND-PAIN-REFLEX`. The `pain-reflex` operator has no arguments specifying termination actions (for example, `on-success`), because it is intended to run forever.

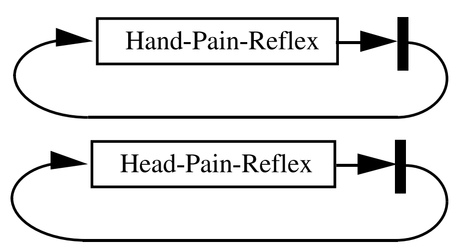

Figure 25. The Petri net of the `pain-reflex` operator.

The goal to `(abolish '(painful hand))`, posted by the `pain-reflex` operator, triggers the instantiation of the `hand-pain-reflex` operator shown in Figure 26. The `hand-pain-reflex` operator has a filter condition complementary to that in `pain-reflex` which ensures that it will <u>not</u> be instantiated for goals containing variables.

The Petri net for hand-pain-reflex, shown graphically in Figure 27, contains eight steps, several of which may run in parallel.

```
(defoperator hand-pain-reflex
  :level :perceptual-manual
  :for (abolishing (painful hand))
  :filters ((neq ?1 :?))
  :steps ((FEEL (discover `(current body-awareness)
                          :call-back
                          #'(lambda () (finish FEEL))))
          (DETECT-HEAT
           (discover `(temperature robot-hand hot)
                     :call-back
                     #'(lambda () (finish DETECT-HEAT))))
          (DETECT-COLD
           (discover `(temperature robot-hand cold)
                     :call-back
                     #'(lambda () (finish DETECT-COLD))))
          (GAIN-CONTROL (start-using :arm-control)
                        (finish GAIN-CONTROL))
          (DROP-CONTENTS (control :hand :open)
                         (finish DROP-CONTENTS))
          (RETRACT (control :arm :retract)
                   (finish RETRACT))
          (YELL (beep) (format t "~%OUCH!, My hand hurts!")
                (finish YELL))
          (DONE (doubt '(current body-awareness))
                (achieve '(current body-awareness)
                         :call-back
                         #'(lambda () (succeed)))))
  :initially (FEEL)
  :transitions ((FEEL :> DETECT-HEAT DETECT-COLD)
                (DETECT-HEAT :> GAIN-CONTROL)
                (DETECT-COLD :> GAIN-CONTROL)
                (GAIN-CONTROL :> DROP-CONTENTS RETRACT YELL)
                (DROP-CONTENTS RETRACT YELL :> DONE))
  :on-success ((reject '(painful hand)))
  :on-termination ((stop-using :arm-control)))
```

Figure 26. The `hand-pain-reflex` operator.

The first step to be executed is `FEEL`, which ensures that the sensor information recorded on the blackboard is current. The notion of “current” at the perceptual/manual level is not explicitly temporal; sensor commands generally finish by asserting that knowledge is current, and control commands generally finish by rejecting or doubting that this is the case. On the other hand, the temporal level periodically rejects `(current body-awareness)`, so the temporal connotations of "current" are sometimes appropriate.

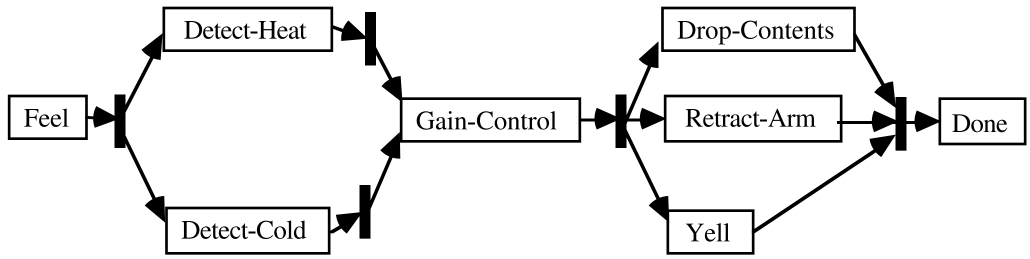

Figure 27. The Petri net of the `hand-pain-reflex` operator.

When the sensor knowledge is deemed to be current two steps, `DETECT-HEAT` and `DETECT-COLD` are both queued for execution. Each of these posts a knowledge goal (with a demon) on the blackboard to discover if a certain hand temperature is detected. The detection of <u>either</u> a hot or cold hand is sufficient to fire a transition in the operator that queues `GAIN-CONTROL` for execution.

The `GAIN-CONTROL` step asks the resource arbitrator to seize control of the `:arm-control` resource. Since the `(abolish '(painful hand))` goal was posted with a priority of `:urgent`, the `hand-pain-reflex` operator also has a priority of `:urgent`. This means that the resource arbitrator will suspend any other operator currently using `:arm-control`, unless that operator <u>also</u> has a priority of `:urgent` (in which case the tie is broken arbitrarily). Once control of the arm has been seized, a transition can fire that will queue the code from steps `DROP-CONTENTS`, `RETRACT` and `YELL` for execution.

`DROP-CONTENTS` calls a primitive control function to open the hand, immediately releasing its contents. `RETRACT` calls a primitive control function to retract the arm, a move intended to maximize the distance between the hand and the offending object.

YELL simply makes a sound and prints a message. These three reactions may occur in any order, and in a true parallel environment they could occur simultaneously. It might be more rational to require that DROP-CONTENTS occur before RETRACT; such a constraint could easily be added. Once all three reactions have been completed, the final transition may fire, queuing the DONE step for execution. The DONE step asserts doubt of the correctness of the current sensory knowledge (since it may not have been updated since the opening of the hand and the retraction of the arm), and declares success once that knowledge has been updated. This triggers an assertion that the pain has been alleviated (via the on-success argument) and releases control of the arm (via the on-termination argument). The hand-pain-reflex operator then terminates, but the loop in pain-reflex ensures that a new instance will be created shortly.

The remainder of this section describes trace output produced by APE during a demonstration the pain reflex. In the test run, a hot, dirty sock[51] is placed a few steps in front of HomeBot, leading via subgoaling from earn-praise, do-laundry, etc., to goals for picking up the sock.[52] The pain reaction ensues when the hot sock is in the hand and a tactile sensing operation is performed. The numbers in the first columns of the listings indicate sequence; the number "1" indicates the first message in the run of the system. Note that there are gaps, particularly in the later listings—the listings have been edited to highlight the portions most relevant to the pain reaction.[53] Gaps in the sequence are highlighted by an initial "*" on the line following the gap. The second column indicates the level at which the described action is taking place, with

[51]HomeBot is fortunate in not having olfactory sensors.

[52]For the sake of this test run the higher levels were "primed" with knowledge of the dirty sock—this speeds the onset of the initial actions.

[53]The full listing for this example, from the beginning of the run to the completion of the hand-pain-reflex operator, is 895 lines.

translation levels indicated by a level abbreviation preceded by "B" (for "Below"). The third column reports the name of the relevant operator or translator, or "Checking for matches" for the operator/translator triggering function. The remaining columns describe the action itself. When the action is the execution of an operator's `step`, the name of the `step` is printed. When the action is the instantiation of an operator, the sentence of the triggering goal is printed.

The following listing shows the first twenty output-producing events of the run:

```
 1 BConv Check for matches
 2 Conv  Check for matches
 3 Conv  EARN-PRAISE              INSTANTIATE  (PRAISE)
 4 Temp  Check for matches
 5 Temp  PROJECT                  INSTANTIATE  (PROJECTIONS
                                                PROJECTED)
 6 Temp  MAINTAIN-BODY-AWARENESS INSTANTIATE  (BODY-AWARENESS)
 7 Temp  TRACK-OBJECT-LOCATIONS  INSTANTIATE  (OBJECT-LOCATIONS)
 8 Caus  Check for matches
 9 BSpat Check for matches
10 Pr/Mn Check for matches
11 Pr/Mn PAIN-REFLEX              INSTANTIATE  (PAINFUL ?)
12 Pr/Mn FEEL                     INSTANTIATE  (CURRENT
                                                BODY-AWARENESS)
13 Pr/Mn LOOK                     INSTANTIATE  (CURRENT
                                                VISUAL-SCENE)
14 BCaus Check for matches
15 Spat  Check for matches
16 Spat  NAVIGATE                 INSTANTIATE  (MOTION-IMPEDED 1)
17 Pr/Mn LOOK                     STEP         BELIEVE-YOUR-EYES
18 Pr/Mn LOOK                     SUCCEED
19 Pr/Mn LOOK                     DISPOSE
20 BTemp Check for matches
```

Much of the activity in this listing corresponds to the instantiation of operators in response to the system's original goals; by the end of the listing, HomeBot has executed the first visual sensing action. Line number 11 shows the instantiation of the `pain-reflex` operator.

The next listing shows the subsequent twenty-two output-producing events, concluding with the posting of the `(abolish '(painful hand))` subgoal by `pain-`

reflex:

```
  21 BSpat Check for matches
  22 Caus  Check for matches
  23 BSpat Check for matches
  24 Spat  NAVIGATE                STEP         LOOK-FOR-PROBLEM
  25 Conv  EARN-PRAISE             STEP         DETECT-HAZARD
  26 BTemp Check for matches
  27 Temp  TRACK-OBJECT-LOCATIONS  STEP         WAIT-TO-SURVEY
  28 Pr/Mn PAIN-REFLEX             STEP         HEAD-PAIN-REFLEX
  29 BCaus Check for matches
  30 Conv  EARN-PRAISE             STEP         DETECT-IMPROPRIETY
  31 Temp  PROJECT                 STEP         CHECK-FOR-NEW
  32 BConv Check for matches
  33 Caus  Check for matches
  34 Spat  NAVIGATE                STEP         (spinning)
  35 BSpat Check for matches
  36 BSpat MOTION-BLOCKAGE         INSTANTIATE
  37 Pr/Mn FEEL                    STEP         GET-DATA
  38 Pr/Mn FEEL                    STEP         BELIEVE-YOUR-FEELINGS
  39 Pr/Mn FEEL                    SUCCEED
  40 Pr/Mn FEEL                    DISPOSE
  41 BSpat MOTION-BLOCKAGE         RUN
  42 Pr/Mn PAIN-REFLEX             STEP         HAND-PAIN-REFLEX
```

The HAND-PAIN-REFLEX step of the pain-reflex operator (line 42) posts a goal that triggers the instantiation of the hand-pain-reflex operator. The next listing shows the instantiation of the hand-pain-reflex operator, and the execution of its demon-posting steps, DETECT-HEAT and DETECT-COLD:

```
* 53 Pr/Mn HAND-PAIN-REFLEX        INSTANTIATE  (PAINFUL HAND)
  54 BTemp Check for matches
  55 Pr/Mn Check for matches
  56 Pr/Mn HAND-PAIN-REFLEX        STEP         FEEL
  57 BSpat MOTION-BLOCKAGE         RUN
  58 Conv  Check for matches
  59 Temp  MAINTAIN-BODY-AWARENESS STEP         (spinning)
* 64 Pr/Mn PAIN-REFLEX             STEP         (spinning)
  65 Pr/Mn Check for matches
  66 BCaus Check for matches
  67 Pr/Mn HAND-PAIN-REFLEX        STEP         DETECT-HEAT
* 76 Pr/Mn HAND-PAIN-REFLEX        STEP         DETECT-COLD
```

At this point the `hand-pain-reflex` operator instance has completed the preparation of the hand-pain reflex. When the `hand-pain-reflex` operator next receives processor time it will only "spin," since none of its transitions are enabled and it has no code queued for execution:

```
* 86 Pr/Mn HAND-PAIN-REFLEX        STEP         (spinning)
```

This causes the system scheduler to "idle" the operator instance until one of its demons fires.

The next listings show some of the key events leading to the grabbing of the dirty sock. Line 142 shows the execution of the `REMOVE-IMPROPRIETY` step of the conventional-level `earn-praise` operator. (The "impropriety" is the dirty sock on the floor.) The `cleaning` translator, responding to this goal, posts a goal of `(abolishing (dirty-clothing :?))` at the causal level:

```
*142 Conv  EARN-PRAISE             STEP         REMOVE-IMPROPRIETY
*165 BConv CLEANING                INSTANTIATE
```

The subsequent lines show the decomposition of the sock-related goals through the levels of the system. The causal-level `do-laundry` operator posts a goal that triggers the spatial-level `put-on` operator, which in turn posts a goal that triggers the perceptual/manual-level `grab` operator:

```
*180 Caus  DO-LAUNDRY              INSTANTIATE  (CLEAN DIRTY-SOCK)
*194 Caus  DO-LAUNDRY              STEP         TAKE-TO-MACHINE
*209 BCaus CAUSAL-STACKING         INSTANTIATE
*246 BTemp TEMPORAL-STACKING       INSTANTIATE
*263 Spat  PUT-ON                  INSTANTIATE  (ON DIRTY-SOCK
                                                    LAUNDRY-MACHINE)
*298 Spat  PUT-ON                  STEP         GRAB-TOP
*301 BSpat HAND-CONTENTS           INSTANTIATE
*323 Spat  PUT-ON                  STEP         MONITOR-BASE
*338 Pr/Mn GRAB                    INSTANTIATE  (HOLDING DIRTY-SOCK)
```

The `grab` operator posts a goal that triggers a `move-hand` operator, eventually triggering an instances of, `roll`, `move-forward`, and other low-level movement operators. By the end of the following listing HomeBot's hand has closed around the dirty sock:

```
*352 Pr/Mn GRAB                   SEIZE        ARM-CONTROL
 353 Pr/Mn GRAB                   STEP         GAIN-ARM-CONTROL
*357 Pr/Mn GRAB                   STEP         OPEN-HAND
 358 Pr/Mn GRAB                   SEIZE        BODY-MOTION
 359 Pr/Mn GRAB                   SEIZE        BODY-ORIENTATION
 360 Pr/Mn GRAB                   STEP         GAIN-BODY-CONTROL
*366 Pr/Mn GRAB                   STEP         MOVE-HAND
*382 Pr/Mn MOVE-HAND              INSTANTIATE  (HAND-AT 1 14 1)
*423 Pr/Mn ROLL                   INSTANTIATE  (AT 1 15 ?)
*482 Pr/Mn ROLL                   STEP         TAKE-A-STEP
*548 Pr/Mn MOVE-FORWARD           SUCCEED
 549 Pr/Mn MOVE-FORWARD           DISPOSE
 550 Pr/Mn ROLL                   STEP         RE-LOCATE
*705 Pr/Mn GRAB                   STEP         CLOSE-HAND
*717 Pr/Mn CLOSE-HAND             STEP         DO-IT
```

The next listings show the relevant events from the closing of the hand through the activation of the pain reflex. Lines 735–754 show the instantiation and success of the first tactile sensing operation subsequent to the closing of the hand:

```
 727 Pr/Mn CLOSE-HAND             SUCCEED
 728 Pr/Mn CLOSE-HAND             DISPOSE
*735 Pr/Mn FEEL                   INSTANTIATE  (CURRENT
                                                BODY-AWARENESS)
*741 Pr/Mn FEEL                   STEP         GET-DATA
*752 Pr/Mn FEEL                   STEP         BELIEVE-YOUR-FEELINGS
 753 Pr/Mn FEEL                   SUCCEED
 754 Pr/Mn FEEL                   DISPOSE
```

Because the sock is hot, this operation automatically fires the `DETECT-HEAT` demon posted by `hand-pain-reflex`, enabling the transition in `hand-pain-reflex` that allows the pain-reaction to proceed. Lines 759–764 show the execution of the `GAIN-CONTROL` step of `hand-pain-reflex`.[54] Since `hand-pain-reflex` has a priority

[54]The system prints the "step" message at the end of the execution of a step. The messages

of `:urgent`, all other operators using the `:arm-control` resource are suspended:

```
*759 Pr/Mn HAND-PAIN-REFLEX         SEIZE        ARM-CONTROL
 760 Pr/Mn GRAB                     SUSPEND
 761 Pr/Mn MOVE-HAND                SUSPEND
 762 Pr/Mn GET-WITHIN-REACH         SUSPEND
 763 Pr/Mn FIGURE-DIRECTION         SUSPEND
 764 Pr/Mn HAND-PAIN-REFLEX         STEP         GAIN-CONTROL
```

Lines 811 and 812 show a subsequent attempt by `move-hand` to regain the `:arm-control` resource—the attempt fails because `hand-pain-reflex` still controls it. Lines 785, 825, and 848 show the execution of the `DROP-CONTENTS`, `RETRACT`, and `YELL` steps of `hand-pain-reflex`:

```
*785 Pr/Mn HAND-PAIN-REFLEX         STEP         DROP-CONTENTS
*811 Pr/Mn MOVE-HAND                Can't seize  ARM-CONTROL
 812 Pr/Mn MOVE-HAND                SUSPEND
*825 Pr/Mn HAND-PAIN-REFLEX         STEP         RETRACT
*848 Pr/Mn HAND-PAIN-REFLEX         STEP         YELL
```

After the completion of these steps a new goal for tactile sensation is posted by the `DONE` step of `hand-pain-reflex`:

```
*861 Pr/Mn HAND-PAIN-REFLEX         STEP         DONE
```

This triggers an instance of the `feel` operator; when it is finished, the `hand-pain-reflex` operator succeeds, releasing the `:arm-control` resource and allowing the resumption of the previously suspended operator instances:

---

printed in lines 759–763 were all generated <u>during</u> the execution of the `GAIN-CONTROL` step that is reported in line 764.

```
*886 Pr/Mn FEEL                    STEP         BELIEVE-YOUR-FEELINGS
 887 Pr/Mn FEEL                    SUCCEED
 888 Pr/Mn FEEL                    DISPOSE
*893 Pr/Mn HAND-PAIN-REFLEX        SUCCEED
 894 Pr/Mn HAND-PAIN-REFLEX        DISPOSE
 895 Pr/Mn HAND-PAIN-REFLEX        RELEASE      ARM-CONTROL
 896 Pr/Mn GRAB                    RESUME
 897 Pr/Mn MOVE-HAND               RESUME
 898 Pr/Mn GET-WITHIN-REACH        RESUME
 899 Pr/Mn FIGURE-DIRECTION        RESUME
```

Many of the lines of output <u>not</u> shown in the above listings pertain to other, unrelated processes occurring within HomeBot during the same time interval. The activity of the perceptual-manual level is disrupted by the pain-reaction, but reasoning at other levels (for example, back-chaining on causal rules) continues without interruption.

9.3.1.3 HomeBot Navigates

HomeBot's perceptual/manual level is capable of generating only straight-line motion in the world. The spatial level implements a simple form of path planning that allows HomeBot to circumvent obstacles that are placed in its way. HomeBot's paths are simply intermediate destination points; if the robot is blocked in moving from point *a* to point *b*, then spatial level operators will attempt to find a reasonable point *c* such that the paths from *a* to *c* and from *c* to *b* are both clear. The procedure can also be applied recursively. This is not a sophisticated form of navigation, but it serves to illustrate several points about the supervenience architecture.

The spatial level is not involved in low-level movement tasks unless problems are encountered that require spatial reasoning. Aside from the processes that monitor for such problems, the computational resources of the spatial level are free for other uses during motion in the world. When spatial level reasoning and intervention become necessary, unaffected components of the perceptual/manual level continue to operate. This allows for progress to be made on other perceptual/manual tasks, and it permits the continuation of processes that may determine that the previously detected problem has been serendipitously alleviated. This demonstrates how the supervenience architecture allows for the application of complex reasoning procedures without interfering unnecessarily with the reactive behavior of the lower levels.

The `roll` operator is HomeBot's perceptual/manual operator most directly responsible for moving to various locations in the world. The `roll` operator generates approximately straight-line motion; its Petri net is shown in Figure 28. For each increment of movement the operator figures the direction (north, east, south or west) in which HomeBot must move to approach the target. It then figures which type of movement (left, right, forwards or backwards) is required and posts a goal for one step of that type. If the movement is accomplished successfully (determined by relocating and comparing initial and final locations) then the process is repeated until the target location is reached. Failure may occur either by a lack of progress determined by the `check-progress` step, or by a detection of blockage by the `watch-for-blockage` step. The `watch-for-blockage` step posts a demon that will be triggered upon failure of the subgoaling primitive motion operator (`move-forward`, `move-left`, etc.). Whenever failure occurs the operator posts a belief in an obstacle, possibly of unknown identity, in the appropriate place. Failure also triggers posting of a statement that the path to the target location is "impassable."

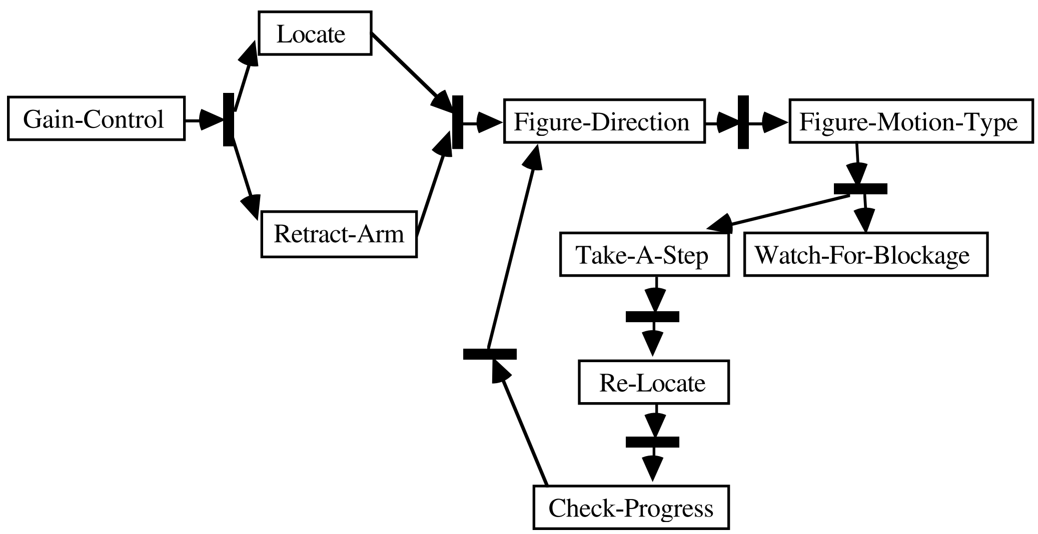

Figure 28. The Petri net of the `roll` operator.

An initial goal at the spatial level is to `(abolish '(motion-impeded 1))`.[55] This goal triggers the `navigate` operator, the Petri net of which is shown in Figure 29. The `look-for-problem` step posts a goal that, via the `motion-blockage` translator, detects failed `roll` operations at the perceptual/manual level. The `motion-blockage` translator, shown in Figure 30, generates sentences about blockages at the spatial

[55]The "1" in this goal sentence is a counter for the number of navigators that have been instantiated; it is used to allow for multiple simultaneous navigational processes. This is necessary because APE allows only one instantiation of a given operator for a given goal at a time.

level based on conjoined knowledge about discovered impassable paths and about current achievement goals for movement of the robot. The `monitor-goal` step posts a goal that triggers a similar translator, called `relieved-blockage`, that monitors the perceptual/manual level for removal of the obstacle. The `find-path` step posts a goal that triggers the `plan-path` operator in order to find a path around the obstacle. The `broadcast` step is executed as soon as <u>either</u> a path is found or the obstacle is noticed to have been removed; it then posts the new path and triggers the `path` translator to send the path to the perceptual/manual level. If the obstacle has been removed then the path-planning operation is cancelled (via a call to `ignore` made by the `monitor-goal` step) and the original path is passed back down.

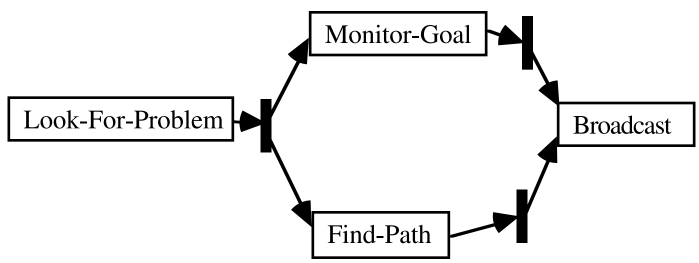

Figure 29. The Petri net of the `navigate` operator.

```
(deftranslator motion-blockage
  :xlevel :below-spatial
  :demand (discovering (blocked-to :? :? :? :?))
          ;the above variables are for x, y, obstacle-x, obstacle-y
  :supplies ((blocked-place ; places that it's impassable to
               (get-all-believed
                `(impassable-to ,?1 ,?2 :? ,?3 ,?4 :?)))
             (destination   ; places HomeBot's trying to get to
               (get-all-achieving `(at ,?1 ,?2 :?))))
  :test (and (eql (nth 1 (sentence blocked-place)) ; same x coords
                  (nth 1 (sentence destination)))
             (eql (nth 2 (sentence blocked-place)) ; same y coords
                  (nth 2 (sentence destination))))
  :products
    ((believe `(blocked-to ,(nth 1 (sentence blocked-place))
                           ,(nth 2 (sentence blocked-place))
                           ,(nth 4 (sentence blocked-place))
                           ,(nth 5 (sentence blocked-place))))))
```

Figure 30. The `motion-blockage` translator.

```
(defoperator roll-on-path
  :level :perceptual-manual
  :for (achieving (at :? :? :?))
  :filters ((believed `(path-to ,?1 ,?2 :? :?)))
  :variables (path-x path-y)
  :steps ((GET-PATH-PLACE
           (let ((s (sentence (get-believed-ground
                               `(path-to ,?1 ,?2 :? :?)))))
            (setq path-x (nth 3 s))
            (setq path-y (nth 4 s)))
           (finish GET-PATH-PLACE))
          (GET-TO-PATH-PLACE
           (doubt `(impassable-to ,path-x ,path-y :? :? :? :?))
           (achieve `(at ,path-x ,path-y :?)
                    :call-back
                    #'(lambda () (finish GET-TO-PATH-PLACE))))
          (GET-TO-DEST
           (doubt `(impassable-to ,?1 ,?2 :? :? :? :?))
           (achieve `(at ,?1 ,?2 :?)
                    :call-back #'(lambda () (succeed)))))
  :initially (GET-PATH-PLACE)
  :transitions ((GET-PATH-PLACE :> GET-TO-PATH-PLACE)
                (GET-TO-PATH-PLACE :> GET-TO-DEST)))
```

Figure 31. The `roll-on-path` operator.

When a path is present at the perceptual/manual level the `roll-on-path` operator becomes applicable in place of the `roll` operator. The `roll-on-path` operator, shown in Figure 31, sequentially posts goals for reaching the intermediate and destination targets. These goals trigger new instantiations of the `roll` operator. If obstacles are encountered along the new path then the navigation process continues recursively.

Figure 32 shows an operator activation graph, generated from trace output, for a span of time during which navigation and path planning were required. The graph shows the activity of operators with short vertical lines; time is represented by the horizontal axis, with later events to the right. The double horizontal lines separate levels; from top to bottom are the conventional, causal, temporal, spatial, and perceptual/manual levels respectively.

No perceptual/manual activity is shown during path-planning in Figure 32, because the only active processes at that level are unable to proceed without a resolution of the motion blockage. However, if the object in the robot's hand were to suddenly become hot then the pain reaction of the previous example would occur—the pain reflex functions in the same way regardless of processes occurring at the spatial level. If the obstacle is removed then the path-planning process is "short circuited," and progress continues to the destination.

Figure 33 shows an example of short-circuited path-planning. The square indicates the last movement before an obstacle was encountered, and the oval indicates the subsequent navigational and path-planning processes at the spatial level. The obstacle was removed shortly after it was encountered—this is recorded on the perceptual/manual blackboard upon completion of the `look` operator (indicated with a circle). Movement resumes almost immediately (indicated with the rectangle).

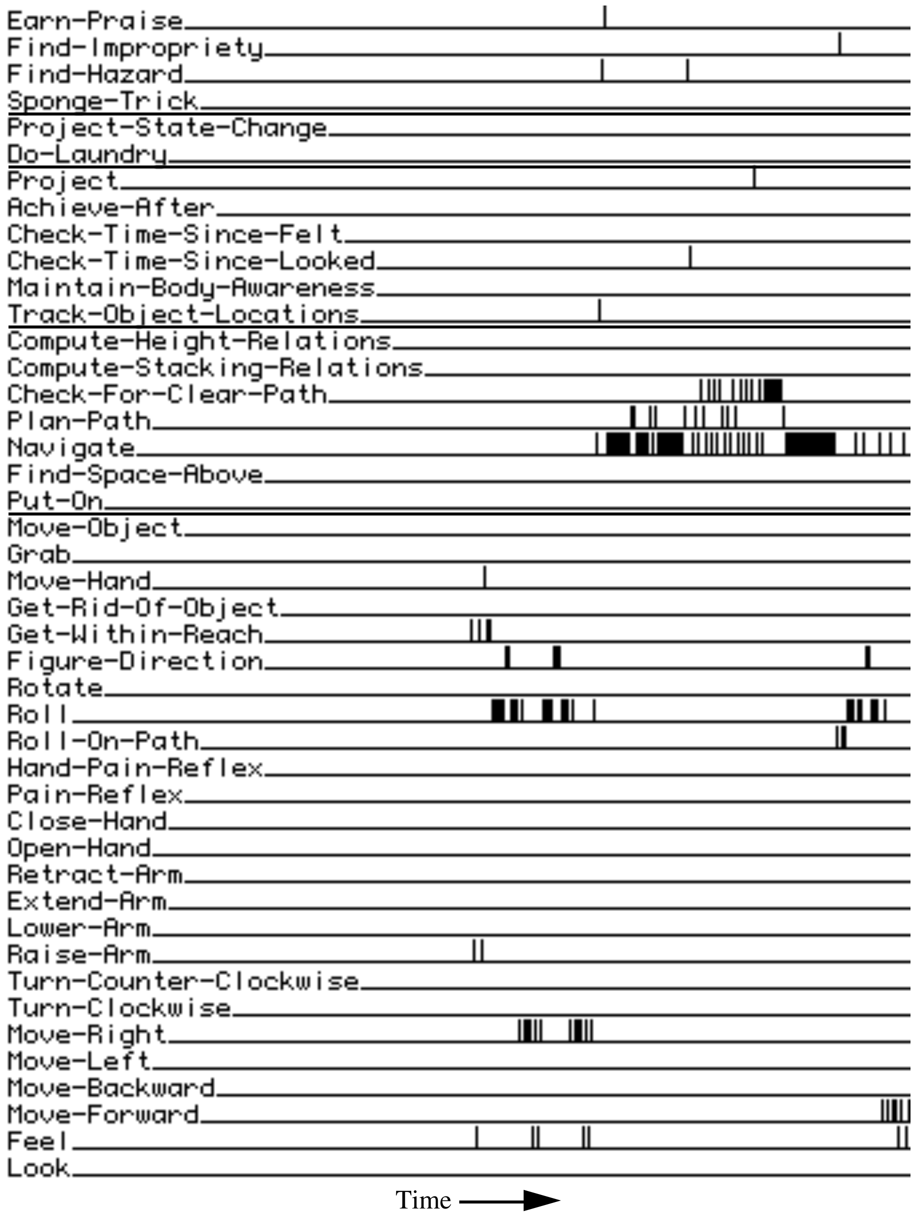

Figure 32. Operator activations during navigation.

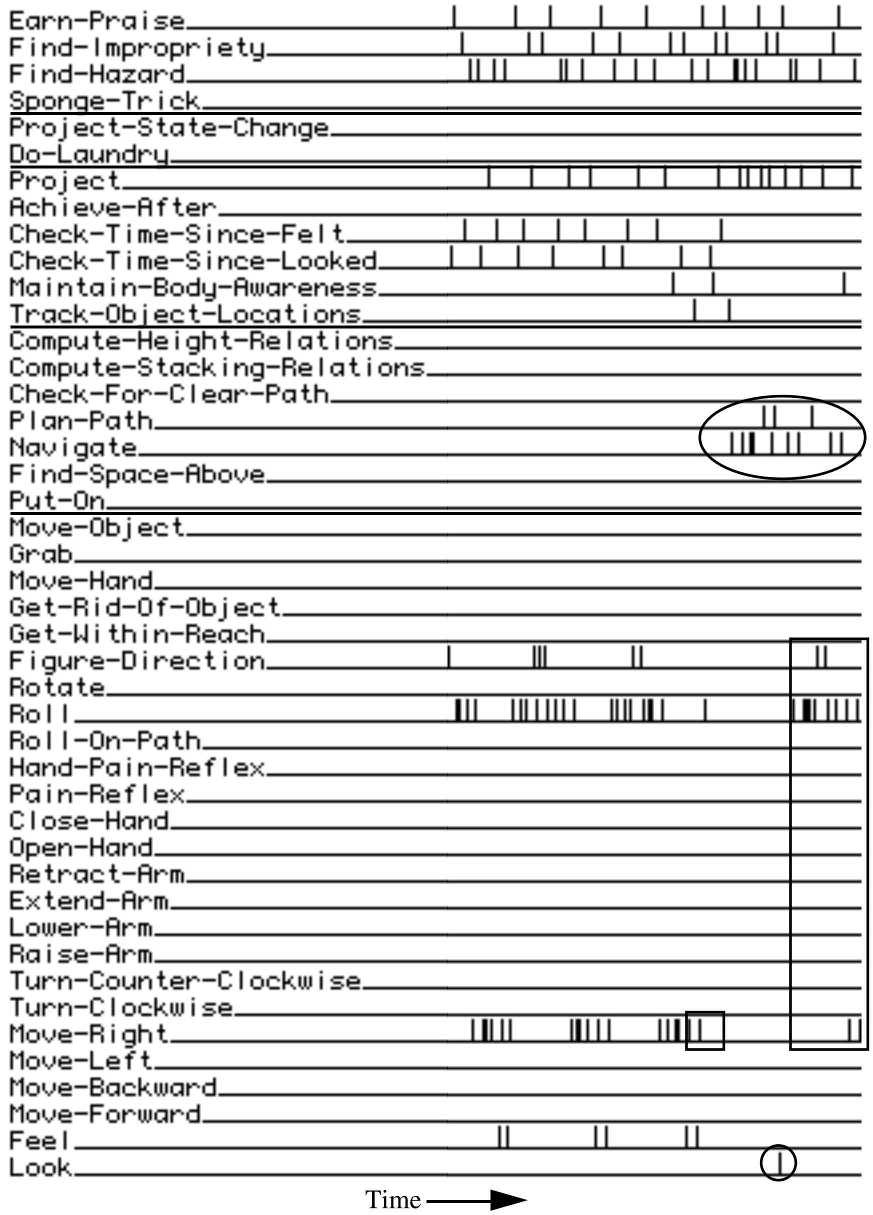

Figure 33. Short-circuited path planning resulting from the removal of an obstacle.

9.3.1.4 HomeBot and the Ice Cube

The Ice Cube problem, described previously in Section 2.3, provides an example of the need for <u>integration</u> of reactive and deliberative components within dynamic-world planning systems. In this problem we suppose that HomeBot is in the midst of a chore when an ice cube is seen in the middle of the floor. We stipulate that there is no built-in low-level “ice cube reaction rule,” and we further suppose that the ice cube itself is not a hazard, but that a puddle on the floor is. This latter supposition serves to make more obvious the demand for high-level reasoning; not only must a causal inference of a hazard take place, but also a causal inference about melting and a temporal inference about the time taken by such a melting.

The appropriate behavior in the early stages of the Ice Cube problem is for HomeBot to continue the previous chore while reasoning about the ice cube. Once a hazard is inferred the previous chore should be suspended, and the hazard should be removed. After the hazard is removed work should resume on the previous chore with as little replanning as possible.

As in previous examples, high-level operators perform sufficient reasoning to set up monitors for “interesting” conditions with respect to their levels of expertise. In this case the conventional-level `earn-praise` operator, triggered by the initial goal to `(achieve '(praise))`, monitors for “improper” or “hazardous” conditions. When either is found a goal is posted for its abolishment. Goals for the removal of improprieties are given `:normal` priority, while goals for the removal of hazards are given `:urgent` priority. This prioritization can be thought of as a simple piece of conventional knowledge encoded into the `earn-praise` operator.

Normal chore-oriented behavior is generated by subgoaling on the goals for discovering and abolishing improper conditions. In the test run described below, the improper condition is a dirty piece of clothing (a sock) on the floor. A representation of the sock propagates to the causal level, where the `do-laundry` operator generates a plan to clean the sock via a series of goals that will decompose into temporal, spatial, and perceptual/manual tasks. Figure 34 shows an operator activation graph, generated from trace output, for a span of time starting at the beginning of a test run and ending with HomeBot moving to pick up the dirty sock. In this test run the higher levels were primed with knowledge of the existence of the dirty sock, as only the later events are of interest.

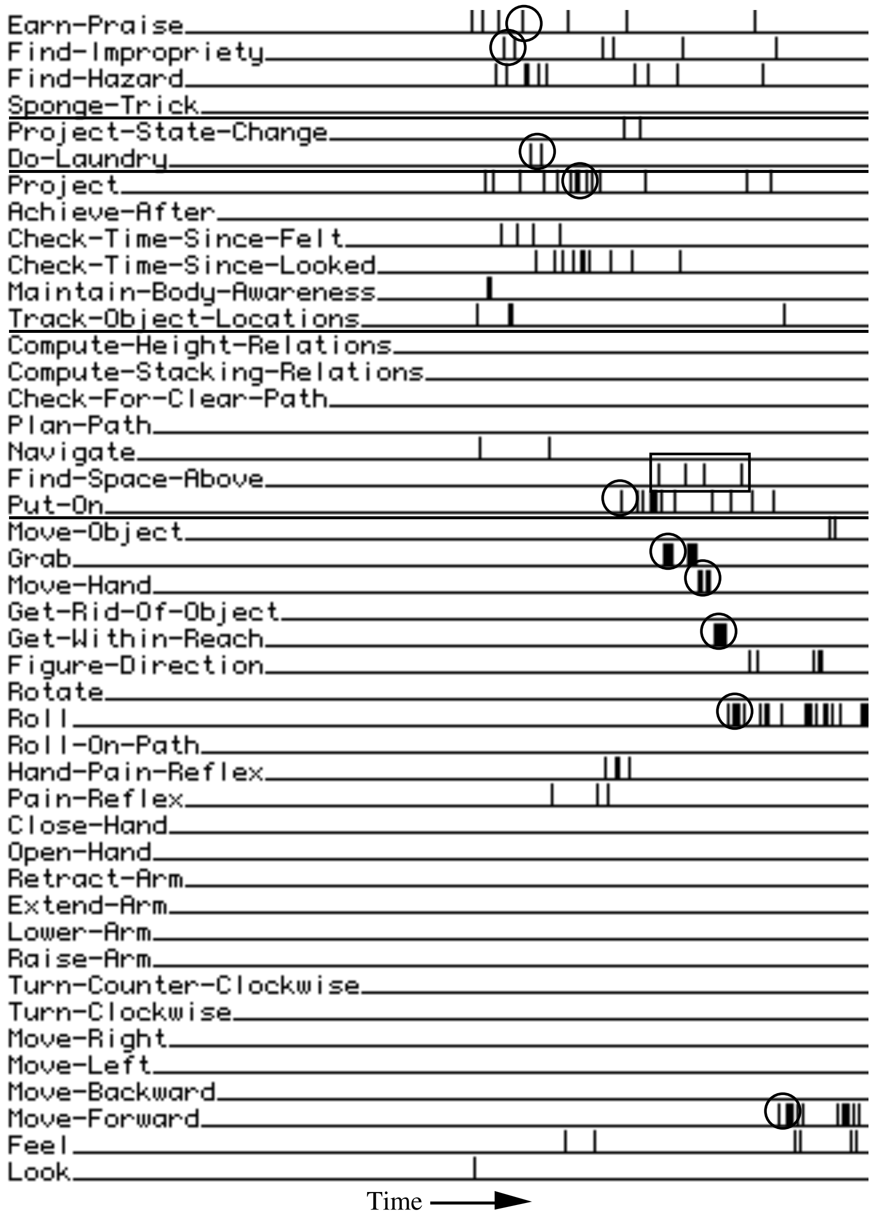

Figure 34. Operator activations leading to initial steps.

The circles in Figure 34 show the flow of activity from the conventional level, at which a decision is made to alleviate the improper condition, to the perceptual/manual level, at which commands are issued for moving the robot. By the end of the shown trace HomeBot has made two steps toward the dirty sock, each followed by a re-orienting `feel` operation. Note that activity continues at various levels while the motion commands are being executed. For example, the spatial level begins to reason about the destination of the dirty sock (with the `find-space-above` operator, shown with a rectangle) as soon as the `put-on` operator has made sufficient progress.

Figure 35 shows a later segment of the same run, during which HomeBot is simultaneously picking up the sock and noticing the ice cube. The large square indicates the actions of picking up the sock (extending the arm, closing the hand, and then raising and retracting the arm), while the circles show the flow of activity relating to the ice cube. The left-most, lower-most circle indicates the `look` operation during which the ice cube is first seen. The subsequent circles show the propagation of the representation of the ice cube to the causal level, at which a state change (melting) is projected, and to the conventional level, at which the change is recognized as a hazard. The `sponge-trick` operator contains a remedy for ice cube hazards: put a sponge on the cube to absorb the water as the ice melts. Although this solution is contrived it is not “hard wired”—once the goal of preventing the puddle is posted, more realistic operators could also be employed to discover a solution. The more important point, however, is that the solution is not hard wired <u>to the conditions in the world</u>. The fact that `sponge-trick` is applicable requires inference of the hazard, and the inference processes can progress in parallel with action.

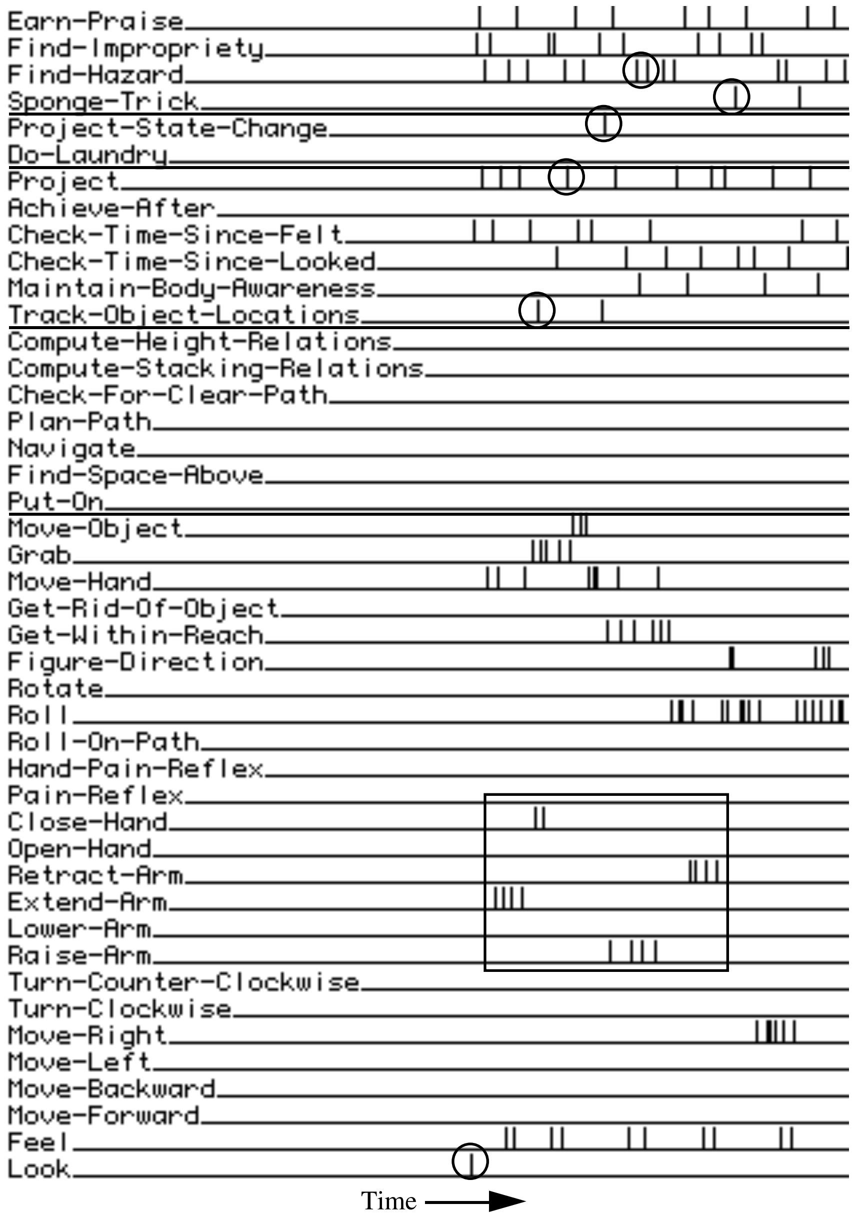

Figure 35. Operator activations while grabbing the dirty sock.

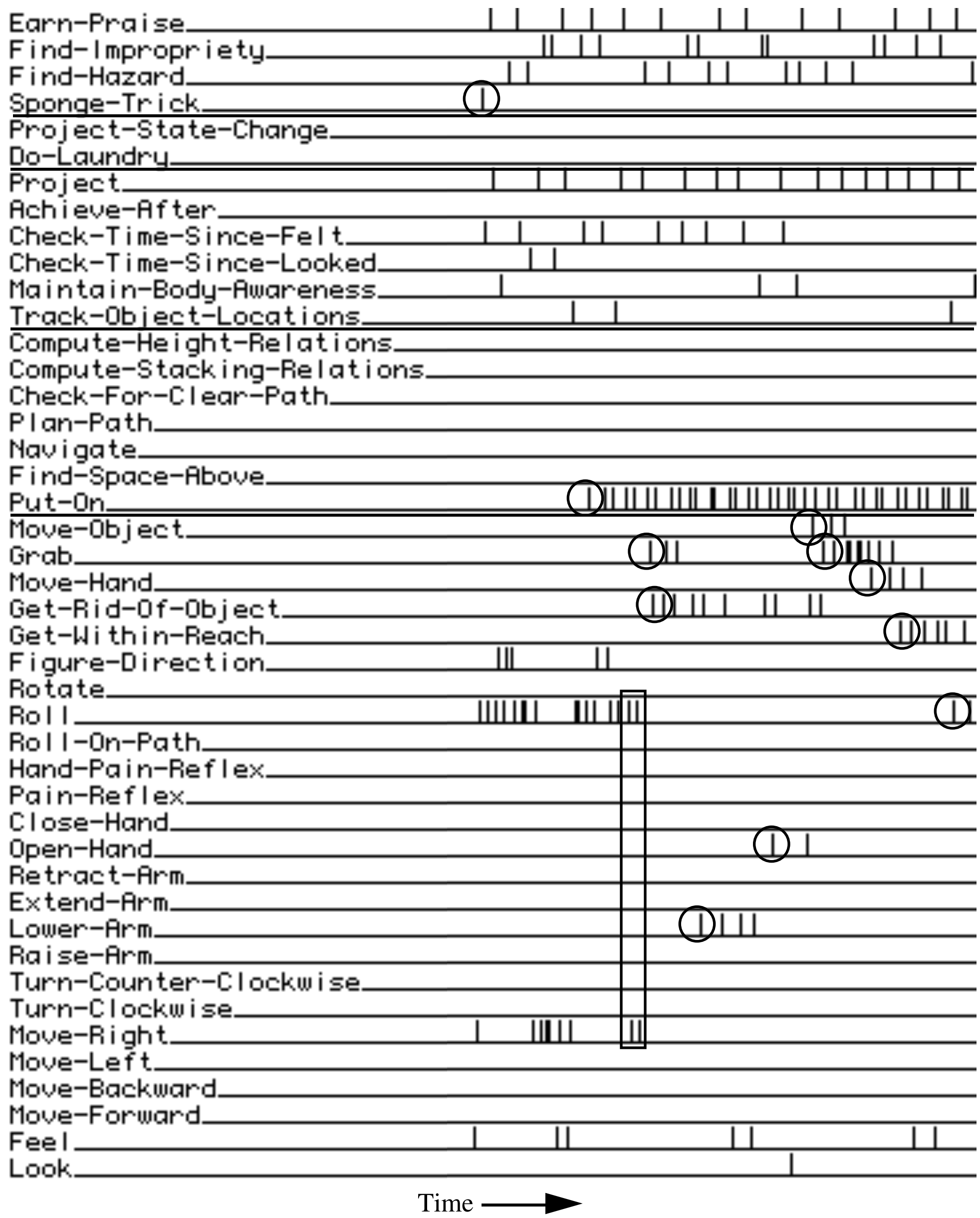

Figure 36. Operator activations during the transition between tasks.

Figure 36 shows the decomposition and execution of the plan for moving the sponge. The circles indicate the major events in this process, while the rectangle indicates the suspension of the previous task. Note that work on the previous task continues until the hazard-prevention task is decomposed to the point at which a resource conflict occurs. (The `roll` and `get-rid-of-object` operators both use the `:arm-control` resource.) Once the operator instances for the new task have seized control of the robot's arm and body, the sock is dropped (via `get-rid-of-object`, `lower-arm`, and `open-hand`) and actions are initiated for the tasks of retrieving and moving the sponge. By the end of the trace shown in Figure 36 a new instance of the `roll` operator has begun to execute.

Figure 37 shows the eventual completion of the hazard-prevention task and the resumption of the laundry task. The circles show the principal events in the completion of the hazard-prevention task: the dropping of the sponge onto the ice cube, the computation that the proper stacking relation holds, and the propagation of the new knowledge to the operators at the conventional level. The squares in Figure 37 indicate the principal events in the resumption of the laundry task. When the sponge is dropped the `get-rid-of-object` operator releases the `:hand-control` resource, allowing for the resumption of operator instances that had previously been suspended.

The instance of the `move-object` operator that was involved in moving the sock failed during the hazard-prevention task; it had posted a demon to monitor `'(holding dirty-sock)` that executed, triggering failure, shortly after the sock was dropped. The operators working on subgoals of `move-object` (for example, `roll`) were also terminated at that time. The goal of moving the sock remained, however, and a new instance of `move-object` was instantiated, leading also to an instance of `grab`. This instance of `grab` was suspended as soon as it requested the `:arm-control` resource.

When the sponge is dropped and the `:arm-control` resource once again becomes available, the suspended `grab` operator is resumed. This allows for resumption of the laundry task, and by the end of the trace in Figure 37 a step has been made back toward the dirty sock. The resumption of the laundry task is fairly simple because the intervention of the hazard-prevention task caused failures only at the lowest level. None of the components of the laundry task at other levels of representation were affected.

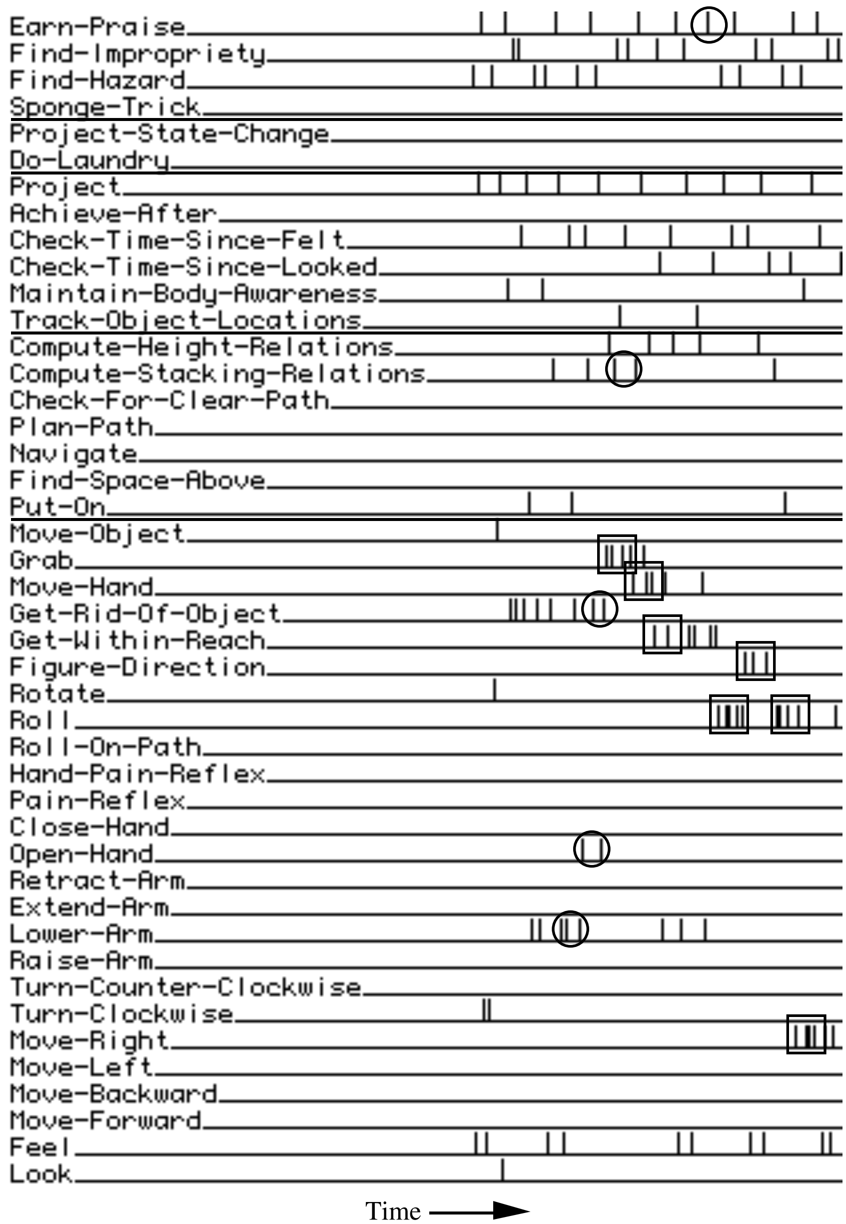

Figure 37. Operator activations during resumption of the laundry task.

9.3.2 Doorbells, Fire, and Overflowing Sinks

Section 9.3.1 described a basic set of operators and translators for HomeBot, and presented examples of HomeBot's behavior using these operators and translators. In the present section I describe three extensions to HomeBot that illustrate how the system's capabilities can be augmented.

The first example, "Doorbells" (9.3.2.1), shows how a new sense modality can be accommodated, and how a new low-level "reflex arc" can be added. The second example, "Fire" (9.3.2.2), shows how new high-level capabilities can build on existing operators and translators. The third example, "Overflowing Sinks" (9.3.2.3), demonstrates a form of plan/act integration not exhibited in previous examples—HomeBot actively seeks situations in which corrective action may be necessary.

9.3.2.1 Doorbells

The purpose of this example is to show the incorporation of a new sense modality and a new low-level "reflex arc" into the HomeBot system. The HomeBot domain simulator was enhanced to provide a simple simulation of the apartment's sonic environment, and a `(sense :audio)` command was added to the interface between the perceptual/manual level and the world simulator.[56] The desired behavior is for HomeBot to respond to a ringing doorbell by suspending any current tasks and opening the door. Ideal door-answering behavior is more complex, and would require mediation by higher, more knowledge-rich levels of representation. For example, the door should not be opened if HomeBot knows that the person at the door is a burglar. These extensions could be made as well, but they are not related to the issues that this example is intended to address.

[56] The doorbell rings for a long time, ensuring that HomeBot's time-slicing will not prevent its detection.

The doorbell-reaction behavior can be implemented with the addition of two new operators at the perceptual/manual level: `hear` (shown in Figure 38), and `door-answering-reflex` (shown in Figure 39). Two additional initial goals, `(abolish '(waiting-at visitor door))` and `(achieve '(current audio-awareness))`, are also added to the system's initialization code. These goal trigger the instantiation of `hear` and `door-answering-reflex` at the beginning of each run.

```
(defoperator hear
  :level :perceptual-manual
  :for (achieving (current audio-awareness))
  :variables (audio-events)
  :steps ((LISTEN (believe '(listening))
                  (setq audio-events (sense :audio))
                  (finish LISTEN))
          (NOTE-SILENCES
           (mapc #'(lambda (audio-event)
                     (reject `(sounding ,audio-event))
                     (believe `(sounded ,audio-event)))
                 (set-difference
                  (mapcar #'cadr
                   (mapcar #'sentence
                    (get-all-believed-ground `(sounding :?))))
                  audio-events))
           (finish NOTE-SILENCES))
          (NOTE-SOUNDS
           (if (null audio-events)
             (fail)
             (progn
               (mapc #'(lambda (audio-event)
                         (believe `(sounding ,audio-event)))
                     audio-events)
               (succeed)))))
  :initially (LISTEN)
  :transitions ((LISTEN :> NOTE-SILENCES)
                (NOTE-SILENCES :> NOTE-SOUNDS))
  :on-success ((reject '(silence)))
  :on-failure ((believe '(silence)))
  :on-termination ((doubt '(listening))))
```

Figure 38. The `hear` operator.

```
(defoperator door-answering-reflex
  :level :perceptual-manual
  :for (abolishing (waiting-at visitor door))
  :steps ((MONITOR-FOR-DOORBELL
           (discover `(sounded doorbell)
                     :call-back
                     #'(lambda () (finish MONITOR-FOR-DOORBELL))))
          (GO-TO-DOOR
           (achieve `(at 3 3 1) :priority :urgent
                    :call-back
                    #'(lambda () (finish GO-TO-DOOR))))
          (ANSWER-DOOR (format t "~%Open Sesame!")
           (finish ANSWER-DOOR))
          (FORGET-DOORBELL
           (reject '(heard doorbell))
           (succeed)))
  :initially (MONITOR-FOR-DOORBELL)
  :transitions ((MONITOR-FOR-DOORBELL :> GO-TO-DOOR)
                (GO-TO-DOOR :> ANSWER-DOOR)
                (ANSWER-DOOR :> FORGET-DOORBELL)))
```

Figure 39. The `door-answering-reflex` operator.

The following listing shows the trace output from the initial portion of a run; lines 11 and 12 show the instantiation of the two operators under discussion:

```
 1 BConv Check for matches
 2 Conv  Check for matches
 3 Conv  EARN-PRAISE             INSTANTIATE  (PRAISE)
 4 Temp  Check for matches
 5 Temp  PROJECT                 INSTANTIATE  (PROJECTIONS
                                               PROJECTED)
 6 Temp  MAINTAIN-BODY-AWARENESS INSTANTIATE  (BODY-AWARENESS)
 7 Temp  TRACK-OBJECT-LOCATIONS  INSTANTIATE  (OBJECT-LOCATIONS)
 8 Caus  Check for matches
 9 BSpat Check for matches
10 Pr/Mn Check for matches
11 Pr/Mn DOOR-ANSWERING-REFLEX   INSTANTIATE  (WAITING-AT
                                               VISITOR DOOR)
12 Pr/Mn HEAR                    INSTANTIATE  (CURRENT
                                               AUDIO-AWARENESS)
```

The `door-answering-reflex` operator sets up a demon-based monitor for the detection of a doorbell sound; when the `hear` operator asserts the occurrence of such a sound the demon will fire, leading to an `:urgent` goal of moving to the door, followed by a command to open the door. Once the door has been opened the `door-answering-reflex` operator succeeds and is disposed of. Note, however, that `door-answering-reflex` does <u>not</u> assert the satisfaction of its triggering goal (`(abolish '(waiting-at visitor door))`); this will result in the creation of a <u>new</u> instance of `door-answering-reflex` shortly after the success of the first.

The `hear` operator executes an audio sensing operation and records the new audio environment on the blackboard, succeeding when something is heard and failing in case of silence. The `NOTE-SILENCES` step changes belief from "`sounding`" to "`sounded`" for sounds that were heard previously, but are no longer in the audio environment. The `NOTE-SOUNDS` step records beliefs about the currently sounding sounds. The `hear` operator, like the `door-answering-reflex` operator, does <u>not</u> assert the satisfaction of its triggering goal; it will therefore "loop" indefinitely via re-instantiation. The looping of the `hear` operator is unconstrained; unlike the `door-answering-reflex` it does not post and wait for any subgoals. Hence the `hear` operator will loop as quickly as allowed by the system's allocation of processor time. This is reasonable, since sounds can occur at any time, and since audio sensing is, in HomeBot, computationally inexpensive.

The following listing shows trace output immediately after the ringing of the doorbell:

```
* 74 Pr/Mn HEAR                    STEP         NOTE-SOUNDS
  75 Pr/Mn HEAR                    SUCCEED
  76 Pr/Mn HEAR                    DISPOSE
* 89 Pr/Mn HEAR                    INSTANTIATE  (CURRENT
                                                 AUDIO-AWARENESS)
*108 Pr/Mn DOOR-ANSWERING-REFLEX   STEP         GO-TO-DOOR
*111 Pr/Mn Check for matches
 112 Pr/Mn ROLL                    INSTANTIATE  (AT 3 3 1)
```

Line 74 shows the step of the `hear` operator that records the ringing of the doorbell on the blackboard. This action immediately fires the demon posted by `door-answering-reflex`, but no action is taken until `door-answering-reflex` receives processor time at line 108. In the meantime, a new instance of the `hear` operator is created (line 89). Line 112 shows the instantiation of the `roll` operator that will take HomeBot to the door.

9.3.2.2 Fire

The basic set of operators and translators described in Section 9.3.1 is capable of detecting only one kind of hazard: a "slip" hazard that is inferred from the existence of a slippery substance or object on the floor. In the present section I will describe the new operators and translators required for handling fire hazards as well.

A sentence at the conventional level, asserted during initialization, indicates that fire also constitutes a hazard: `(believe '(hazard fire))`. This allows the existing `earn-praise` and `find-hazard` operators to handle fires in the same way that they handle slip hazards. The `fight-fire` operator (Figure 40) encodes the conventional solution for abolishing fire conditions; this is analogous to the `sponge-trick` operator for ice cube slip hazards. Additional operators are required for dialing the telephone and for turning on the sprinkler system.

```
(defoperator fight-fire
  :level :conventional
  :for (abolishing (hazardous-condition fire :?))
  :steps ((CALL-FOR-HELP
           (achieve '(telephone-connect 911)
                    :call-back
                    #'(lambda ()
                         (beep) (format t "~%FIRE!!~%")
                         (give-name-and-address)
                         (finish CALL-FOR-HELP))))
          (TURN-ON-SPRINKLERS
           (achieve '(activated sprinkler-system)
                    :call-back
                    #'(lambda () (finish TURN-ON-SPRINKLERS))))
          (WAIT-FOR-SIGNAL
           (discover '(signaled fire-marshal all-clear)
                     :call-back
                     #'(lambda () (succeed)))))
  :initially (CALL-FOR-HELP TURN-ON-SPRINKLERS)
  :transitions ((CALL-FOR-HELP TURN-ON-SPRINKLERS
                 :> WAIT-FOR-SIGNAL))
  :on-success ((reject `(hazardous-condition fire ,?1))))
```

Figure 40. The `fight-fire` operator.

The existence of fire is inferred from the existence of smoke at the causal level by the `infer-fire-from-smoke` operator, shown in Figure 41. The existence of fire is communicated from the causal to the conventional level via the `fire-detection` translator, shown in Figure 42. The causal level is informed of the existence of smoke via the `smoke-detection` translator, shown in Figure 43. The `smoke-detection` translator utilizes temporal-level knowledge about the materials of objects—this knowledge is maintained by the basic operators and translators described in Section 9.3.1.

```
(defoperator infer-fire-from-smoke
  :level :causal
  :for (discovering (projected fire-near :?))
  :variables (smoky-thing)
  :steps ((DETECT-SMOKE
           (discover `(smoke :?)
                     :call-back
                     #'(lambda ()
                         (setq smoky-thing
                               (nth 1 (sentence (get-believed-ground
                                                 '(smoke :?)))))
                         (finish DETECT-SMOKE))))
          (PROJECT (believe `(projected fire-near ,smoky-thing))
                   (succeed)))
  :initially (DETECT-SMOKE)
  :transitions ((DETECT-SMOKE :> PROJECT)))
```

Figure 41. The `infer-fire-from-smoke` operator.

```
(deftranslator fire-detection
  :xlevel :below-conventional
  :demand (discovering (fire :?)) ; variable is sign of fire
  :commands ((discover `(projected fire-near :?)))
  :supplies ((fire-sign (mapcar #'caddr
                         (mapcar #'sentence
                          (get-all-believed-ground
                           '(projected fire-near :?))))))
  :products ((believe `(fire ,fire-sign))))
```

Figure 42. The `fire-detection` translator.

```
(deftranslator smoke-detection
  :xlevel :below-causal
  :demand (discovering (smoke :?)) ; variable is sign of smoke
  :commands ((discover `(object-materials)))
  :supplies ((smoke-thing
              (mapcar #'cadr
                      (mapcar #'sentence
                              (get-all-believed-ground
                               '(material :? :smoke))))))
  :products ((believe `(smoke ,smoke-thing))))
```

Figure 43. The `smoke-detection` translator.

The following listings show trace output from a test run that was initialized with a smoke cloud. Lines 232–260 show the initialization of the causal level processes for inferring the existence of fire:

```
*232 Caus  INFER-FIRE-FROM-SMOKE   INSTANTIATE  (PROJECTED
                                                 FIRE-NEAR ?)
*246 Caus  INFER-FIRE-FROM-SMOKE   STEP         DETECT-SMOKE
*260 Caus  INFER-FIRE-FROM-SMOKE   STEP         (spinning)
```

The `fire-detection` translator runs continuously (for example, at lines 286 and 300) but has nothing to report until the `smoke-detection` translator communicates the existence of smoke to the causal level (lines 303–323). The causal level then infers the existence of fire (lines 329–331), allowing `fire-detection` to relay the information to the conventional level (line 377):

```
*286 BConv FIRE-DETECTION          RUN
*300 BConv FIRE-DETECTION          RUN
*303 BCaus Check for matches
 304 BCaus SMOKE-DETECTION         INSTANTIATE
*323 BCaus SMOKE-DETECTION         RUN
*329 Caus  INFER-FIRE-FROM-SMOKE   STEP         PROJECT
 330 Caus  INFER-FIRE-FROM-SMOKE   SUCCEED
 331 Caus  INFER-FIRE-FROM-SMOKE   DISPOSE
*377 BConv FIRE-DETECTION          RUN
```

This allows for the instantiation and progress of the `fight-fire` operator, shown in the following lines:

```
*437 Conv  FIGHT-FIRE              INSTANTIATE  (HAZARDOUS-CONDITION
                                                 FIRE SMOKE-CLOUD)
*535 Conv  FIGHT-FIRE              STEP         TURN-ON-SPRINKLERS
*552 Conv  ACTIVATE-SPRINKLERS     INSTANTIATE  (ACTIVATED
                                                 SPRINKLER-SYSTEM)
*596 Conv  ACTIVATE-SPRINKLERS     STEP         ACTIVATE
 597 Conv  ACTIVATE-SPRINKLERS     SUCCEED
 598 Conv  ACTIVATE-SPRINKLERS     DISPOSE
*613 Conv  FIGHT-FIRE              STEP         CALL-FOR-HELP
```

9.3.2.3 Overflowing Sinks

One of the "user interventions" built into the HomeBot world simulator allows for the overflowing of the kitchen sink. This leads to several problems (slip hazards, electrocution hazards, wasted water, etc.) and poses interesting questions about elements of solution strategies (turning off the water, getting to the mop without being electrocuted, etc.). However, none of the interesting issues can be addressed if HomeBot <u>never</u> <u>notices</u> that the sink overflowed. To ensure that HomeBot notices such things, it is necessary that visits be made to each room in the apartment "once in a while." This is a form of sense/act integration not demonstrated in the previous examples: HomeBot must act (move) in order to sense (look in various rooms) in order to know if further action (for example, taking care of an overflowing sink) is necessary. Further, the decisions about which rooms to visit may depend on prior reasoning.

The notion of "once in a while" belongs at the temporal level, and may depend on the room in question. Hence, sentences such as `(kitchen-survey-frequency :? cycles)` are asserted at the temporal level. These sentences can be modified by operators when, for example, it is known that someone else is watching a particular room. The `track-object-locations` operator (part of the basic set of operators from Section 9.3.1) contains a step that posts a goal to `(debunk (current room-scenes))`. This goal triggers the `room-visit-recency` operator shown in Figure 44. The `room-visit-recency` operator compares the times since HomeBot has been in each of the rooms to the survey frequencies posted on the temporal blackboard. The knowledge of the times at which HomeBot is in each room is maintained by the `room` and `temporal-room` translators (Figures 45 and 46 respectively). The `room` translator transforms perceptual/manual knowledge about HomeBot's location (expressed in coordinates) into spatial knowledge about the room that HomeBot is in. The `temporal-room` translator transforms the resulting spatial knowledge into temporal knowledge about <u>when</u> HomeBot is in each room.

```
(defoperator room-visit-recency
  :level :temporal
  :for (debunking (current room-scenes))
  :variables (now check-kitchen check-bathroom
                  check-bedroom check-livingroom)
  :steps ((GET-TIME (setq now ; get current time from blackboard
                          (cadr (sentence (get-believed-ground
                                            '(homebot-time :?)))))
                    (finish GET-TIME))
          (KITCHEN ; check if kitchen needs to be checked
           (setq check-kitchen
                 (> (- now   ; compare time since last visit...
                       (apply
                        #'max
                        (mapcar #'caddr
                                (mapcar #'sentence
                                        (get-all-believed-ground
                                         '(in-room kitchen :?))))))
                    (cadr    ; to frequency kitchen should be visited
                     (sentence
                      (get-believed-ground
                       '(kitchen-survey-frequency :? cycles))))))
           (finish KITCHEN))
          (BATHROOM <similar code deleted>...)
          (BEDROOM <similar code deleted>...)
          (LIVINGROOM <similar code deleted>...)
          (DONE
           (if (not (or check-kitchen check-bathroom
                        check-bedroom check-livingroom))
             (fail)
             (progn (if check-kitchen ; post necessity of visit
                       (believe '(old-room-view kitchen)))
                    (if check-bathroom
                       (believe '(old-room-view bathroom)))
                    (if check-bedroom
                       (believe '(old-room-view bedroom)))
                    (if check-livingroom
                       (believe '(old-room-view livingroom)))
                    (succeed)))))
  :initially (GET-TIME)
  :transitions ((GET-TIME :> KITCHEN BATHROOM BEDROOM LIVINGROOM)
                (KITCHEN BATHROOM BEDROOM LIVINGROOM :> DONE))
  :on-success ((reject '(current room-scenes))))
```

Figure 44. The `room-visit-recency` operator (abbreviated).

```
(deftranslator room
  :xlevel :below-spatial
  :demand (discovering (room))
  :supplies ((x (list
                 (cadr (sentence
                        (get-believed-ground '(at :? :? :?))))))
             (y (list
                 (caddr (sentence
                         (get-believed-ground '(at :? :? :?)))))))
  :products ((doubt '(in-room :?))
             (believe `(in-room ,(room-containing x y)))))
```

Figure 45. The `room` translator.

```
(deftranslator temporal-room
  :xlevel :below-temporal
  :demand (discovering (room-when))
  :supplies ((room (list
                    (cadr (sentence
                           (get-believed-ground '(in-room :?)))))))
  :products ((believe `(in-room ,room
                                ,(cadr (sentence
                                        (get-believed-ground
                                         '(homebot-time :?))))))
             (believe `(in-room ,room))))
```

Figure 46. The `temporal-room` translator.

When `room-visit-recency` deems that a room should be visited, it posts messages to the blackboard and succeeds, leading `track-object-locations` to post a goal that triggers the `visit-rooms` operator (Figure 47). The `visit-rooms` operator posts goals for visiting each of the appropriate rooms with priority `:background`; hence HomeBot will visit the rooms as soon as it is not otherwise engaged. The goals to visit the rooms decompose into perceptual/manual movement goals with the aid of the `go-room` and `temporal-go-room` translators (Figures 48 and 49 respectively). These goals trigger `roll` operators with priority `:background` (inherited from `visit-rooms`).

```
(defoperator visit-rooms
  :level :temporal
  :for (achieving (current room-scenes))
  :variables (rooms-to-visit rooms-visited)
  :steps ((GET-ROOM-LIST (setq rooms-to-visit
                               (mapcar #'cadr
                                 (mapcar #'sentence
                                         (get-all-believed-ground
                                          '(old-room-view :?)))))
                         (finish GET-ROOM-LIST))
          (POST-GOALS (doubt '(in-room :?))
                      (setq rooms-visited nil)
                      (mapcar
                       #'(lambda (room)
                           (achieve `(in-room ,room)
                                    :priority :background
                                    :call-back
                                    #'(lambda ()
                                       (push room rooms-visited))))
                               rooms-to-visit)
                      (finish POST-GOALS))
          (WAIT-TILL-DONE
           (if (set-difference rooms-to-visit rooms-visited)
             (finish WAIT-TILL-DONE)
             (succeed))))
  :initially (GET-ROOM-LIST)
  :transitions ((GET-ROOM-LIST :> POST-GOALS)
                (POST-GOALS :> WAIT-TILL-DONE)
                (WAIT-TILL-DONE :> WAIT-TILL-DONE))
  :on-success ((reject '(old-room-view :?))
               (believe '(current room-scenes))))
```

Figure 47. The `visit-rooms` operator.

```
(deftranslator go-room
  :xlevel :below-spatial
  :demand (achieving (in-room :?))
  :commands ((achieve `(at ,@(room-center ?1)))))
```

Figure 48. The `go-room` translator.

```
(deftranslator temporal-go-room
  :xlevel :below-temporal
  :demand (achieving (in-room :?))
  :commands ((achieve `(in-room ,?1))))
```

Figure 49. The `temporal-go-room` translator.

The following queries were executed during a task in the living room:

```
1 > (at-level :temporal
              (get-all-achieving '(in-room :?)))
(<Blackboard Entry:(IN-ROOM KITCHEN) (E:?)(TE:?)(T:+)>
 <Blackboard Entry:(IN-ROOM BATHROOM) (E:?)(TE:?)(T:+)>
 <Blackboard Entry:(IN-ROOM BEDROOM) (E:?)(TE:?)(T:+)>)

1 > (at-level :spatial
              (get-all-achieving '(in-room :?)))
(<Blackboard Entry:(IN-ROOM BATHROOM) (E:?)(TE:?)(T:+)>
 <Blackboard Entry:(IN-ROOM KITCHEN) (E:?)(TE:?)(T:+)>
 <Blackboard Entry:(IN-ROOM BEDROOM) (E:?)(TE:?)(T:+)>)

1 > (at-level :perceptual-manual
              (get-all-achieving '(at :? :? :?)))
(<Blackboard Entry:(AT 8 23 1) (E:-)(TE:?)(T:+)>   ;; BEDROOM
 <Blackboard Entry:(AT 12 5 1) (E:-)(TE:?)(T:+)>   ;; KITCHEN
 <Blackboard Entry:(AT 5 17 ?) (E:-)(TE:?)(T:+)>   ;; (current task)
 <Blackboard Entry:(AT 15 22 1) (E:-)(TE:?)(T:+)>) ;; BATHROOM
```

Goals exist for visiting the other rooms, but all of their corresponding `roll` operators have been suspended pending the completion of the current task (which has priority `:normal`). When the current task is finished, the resource arbitrator will resume one of the suspended `roll` operators.

9.4 Performance

The performance of the HomeBot system depends on the particular set of operators and translators that are in use, the task environment in the simulated world, the allocations specified for the feap control structures described in Section 8.8, and characteristics of the hardware and software environment. This section describes the system's performance in test runs using the "basic" set of operators described in Section 9.3.1, on a problem similar to that described in Section 9.3.1.2. Figure 50 shows the feap allocations used in the test runs. The tests were performed on an Apple Macintosh IIcx computer, which uses a Motorola 68030 processor and a Motorola 68882 floating-point coprocessor, running with a clock frequency of 15.6672 MHz. The tests were run in Macintosh Common Lisp version 2.0b1p3, in a RAM partition of 17 Megabytes, under Macintosh System Software 7.0.

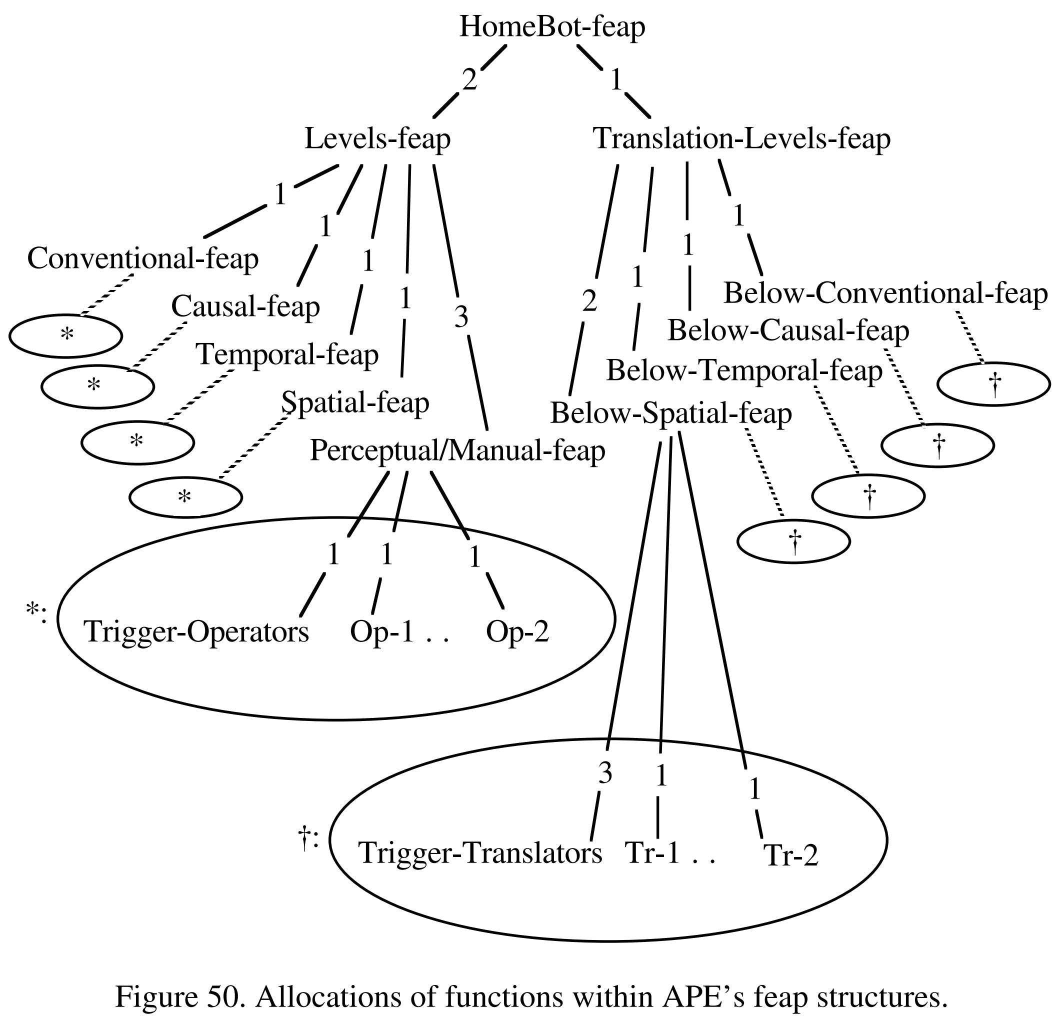

Figure 50. Allocations of functions within APE's feap structures.

As shown in Figure 50, processor time is divided fairly evenly across the system, but extra time is allotted to the lowest levels. In a system that performs significant high-level reasoning one might expect optimal results to be obtained by allocating greater amounts of time to the higher levels (see [Hendler 1990]). But the higher levels defined by the basic operators are quite simple, and HomeBot's sensory and motor commands are slowed by the sluggishness of the world simulator. For example, a visual sensing operation currently requires approximately 1.2 seconds.[57] This is due to the complexity of computing the set of objects in HomeBot's view while accounting for occlusions. The particular allocation numbers were derived from observation and experimentation during system development.

Figure 51 shows a graph of the speed with which APE cycles through its central loop (described in Section 8.8), as a function of the length of the test run. Each iteration of the loop is one call to a single function in the system's top-level feap. A single iteration generally results in a single step of a single operator instance, a single run of a single translator instance, or a call to an operator or translator triggering function. Note, however, that many of these calls may be "fruitless"; an operator instance may "spin," a translator instance may produce no results, and a triggering function may find nothing to trigger. The feap mechanisms themselves also use processor time, and each cycle through the central loop involves calls at all three levels of the feap hierarchy. Additional maintenance procedures (for example, checking for user interrupts) are performed once every 20 iterations.

---

[57]This number varies with the position of the robot, the number of objects in view, etc. Run times as low as 0.9 seconds and as high as 3.6 seconds have been observed.

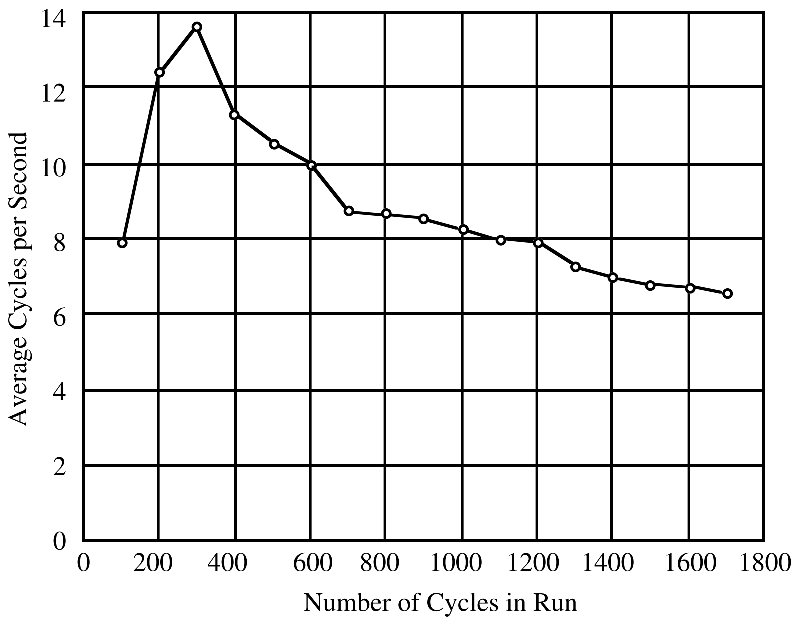

Figure 51. HomeBot cycle speed vs. length of run.

The average cycle speed across all of the test runs was about 8.8 cycles per second. The variation in speed is related to the amount of data accumulated on the system's blackboards. APE makes heavy use of the blackboard access procedures; blackboard access accounts for between one third and one half of the system's total run time. Once an item is put in the blackboard it is never removed; changes in its belief and/or goal status are recorded in the item's tags (see Section 8.4), but the item remains on the blackboard even when all tag values are "?". After 100 main-loop iterations in a test run, a total of 290 sentences had accumulated; after 1700 iterations there were 792 sentences, the growth being approximately linear with the length of the run. Figure 52 illustrates the degradation in the efficiency of the blackboard maintenance procedures in the course of a single run. The graph shows the average run times of a key internal blackboard access function. The brief initial improvement (between 100 and 300 cycles) is consistent with the initial improvement in cycle time shown in Figure 51, although the variability in Figure 52 renders such comparisons speculative.

As mentioned in Section 8.4, the blackboards in APE are implemented as discrimination nets that use the sequence atoms in blackboard sentences as discrimination keys. A reasonable explanation for the performance curves in Figures 51 and 52 is that the initial improvement is due to the “filling out” of an initially “flat” discrimination net structure. The nets reach an optimal structure after a few hundred cycles, after which the lower branches of the discrimination net begin to “flatten” as well, and performance degrades.

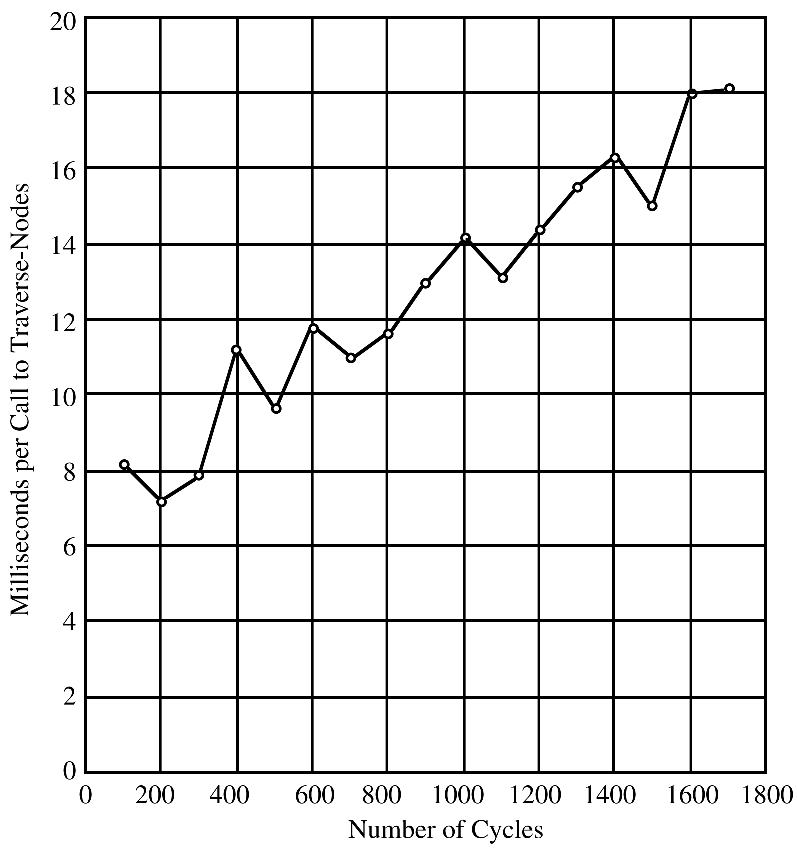

Figure 52. Run time of a key blackboard procedure across a single run of the system.

In the test runs, HomeBot took approximately 15 seconds between “roll” movements in the world. Within 65 seconds of the first such movement, HomeBot had taken the three necessary steps, turned, and performed the arm and hand movements necessary for grasping the object. A “hand pain” reaction, like that described in Section 9.3.1.2, required 15 seconds between the grasping of the hot object and the success of the hand-pain-reflex operator.

HomeBot’s current performance is not adequate for “real-time” behavior. A reasonable response-time for a “pain” reaction, for example, might be an order of magnitude smaller than currently achievable. In considering HomeBot’s performance, however, it is important to bear in mind that APE is a simulation of a parallel architecture on a single processor of modest speed. APE divides its time between a large number of processes that should be running in parallel, and incurs additional overhead in maintaining the related timing structures. Further, the performance statistics for HomeBot include the overhead of maintaining the world simulation; in actual robotic applications such simulation is unnecessary. Nonetheless, there is room for improvement, even in a serial computing environments; some directions for improvement are suggested in Chapter 10.

Part IV

Conclusions

Chapter 10

Summary and Future Directions

This dissertation surveyed a range of concepts of abstraction, levels, and hierarchy in relation to AI systems that must function in complex, dynamic environments. The concept of supervenience was examined, formalized, and compared to forms of abstraction used in AI planning systems. An implemented dynamic-world planning system, based on the concept of supervenience, was presented and shown to solve difficult behavioral problems. The discussion spanned disciplines from psychology to theoretical AI, and further work may be called for in many of these areas.

The field of AI planning systems has evolved from its early focus on small problems in static domains to a concern with the problems that arise in more realistic, complex, and dynamic environments. Dynamic-world planners must be capable of reacting to changes in the world while handling problems of the type solved by early planning systems. They must be capable of integrating deliberation and action in a variety of ways, as illustrated by the discussions of the Open Window and Ice Cube problems in Section 2.3.

A variety of forms of abstraction have been used in AI planning and control systems. Planning systems such as ABSTRIPS use simplification abstraction to reduce the size of problem-solving search-spaces. Control systems designed for dynamic environments often use control hierarchies, in which independent modules handle problems at each level of abstraction. These forms of abstraction have a common core: they both allow for higher levels to abstract away from the world and to reason with respect to an idealized version of reality. The isolation of this core concept of abstraction as “distance from the world” allows for a principled unification of the forms of abstraction most helpful in generating intelligent behavior in complex, dynamic environments.

The concept of supervenience originated in the philosophical literature, and was first applied to problems of ethics and philosophy of mind. Ethical facts are said, on some accounts, to <u>supervene</u> on physical facts; this means that they <u>depend</u> on physical facts, but that they may nonetheless be more than <u>just</u> sets of physical facts. Similar theories have been proffered for the supervenience of the mental on the physical, for the supervenience of the aesthetic on the physical, and for supervenience in a host of other domains, some of which allow for multiple levels of supervenience. Formal-logical accounts of supervenience have been provided, but researchers disagree on the details. The core concept is one of an asymmetric dependency relation between domains of discourse in which the “higher” domains depend on the “lower” domains. The lower domains can be thought of as epistemologically “closer to the world.”

The dissertation formalized the conception of supervenience as distance from the world in the context of nonmonotonic reasoning systems. Supervenience was modeled in a multilevel argument system by limiting the power of higher levels to defeat facts at lower levels; the basic pattern of communication can be described as “assertions up, assumptions down.” The resulting system captures an essential feature of the form of abstraction used in ABSTRIPS-style systems, while allowing for flexibilities found in hierarchical control architectures.

The application of supervenience to dynamic-world planning was facilitated by the development of an architectural model called the <u>supervenience</u> <u>architecture</u>. The supervenience architecture, a multilevel blackboard architecture, posits a set of planning levels ordered by distance from the world. The procedures of each level model the world, solve problems, and initiate actions at their own levels of abstraction. The dependence of higher upon lower levels is implemented by regimenting inter-level communication: knowledge about the world is passed up, and goal-related knowledge is passed down. This "world knowledge up, goals down" style of communication is the implementational analog to the "assertions up, assumptions down" communications regime developed in the formal characterization of supervenience. The supervenience architecture is a generic specification, and many details are left unspecified. It is nonetheless sufficiently elaborated to allow for comparisons to other architectures, and several points of contrast to Brooks's subsumption architecture were noted.

Figure 53 shows the inter-level relations that constitute the principal forms of abstraction covered in the dissertation.

The supervenience architecture was implemented in a program called the <u>Abstraction-Partitioned</u> <u>Evaluator</u> (APE), for which a specific set of five levels was chosen. The five levels—perceptual/manual, spatial, temporal, causal, and conventional—were motivated by philosophical and psychological considerations about the representation of event and action knowledge. APE reasons by means of <u>operators</u> that are specified as Petri nets, allowing for the specification of parallel, asynchronous behaviors. Communication between levels is effected by <u>translators</u> that allow for syntactic manipulation of downward-moving goal knowledge and upward-moving world knowledge. APE simulates parallel, asynchronous execution of many of its components.

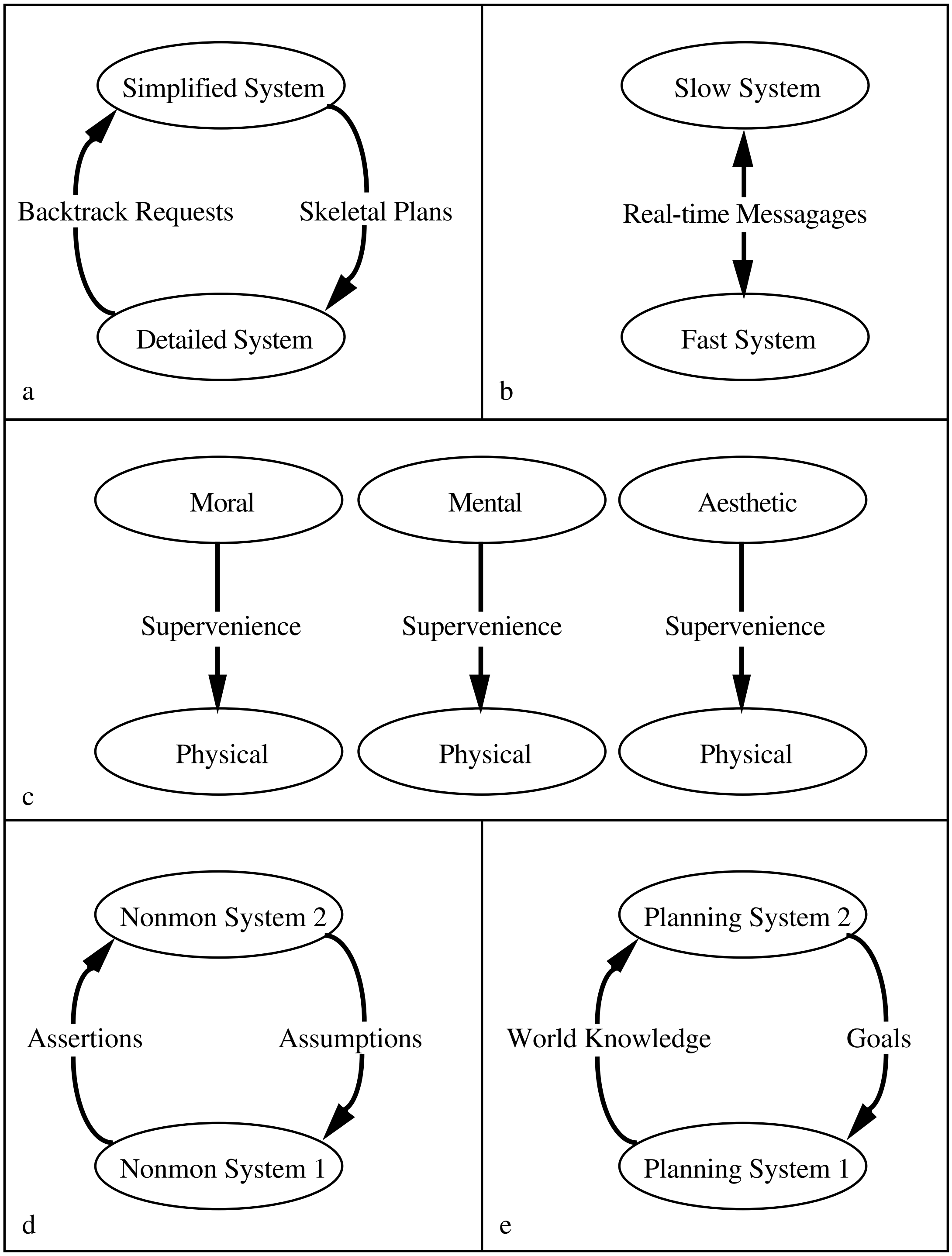

Figure 53. Forms of abstraction: a, simplification abstraction; b, partitioned control abstraction; c, supervenience in philosophy; d, the nonmonotonic formalization of supervenience; e, supervenience in the supervenience architecture.

APE was applied in a simulation environment called the HomeBot domain, in which a household robot is responsible for performing a variety of tasks. Several detailed examples of the HomeBot system were provided, including behavioral problems that cannot be handled by other planning systems. The performance of the HomeBot system is not yet sufficient for "real-time" reactivity, but its performance may be improved by optimization, and by re-implementation in a true parallel environment.

With respect to psychology and neuroscience, several comparative questions may be of interest. The thesis that the mind is organized into levels of supervenience is a cognitive modularity thesis which, if true, would predict certain level-based effects in observed human cognitive function. Parallels to modularity theses of brain structure would also be of interest; one might ask, for example, about the <u>deficits</u> that might be observed in <u>lesioned</u> instances of the supervenience architecture, and about their relation to deficits observed in brain-damaged human subjects. Further comparisons might also be made to levels of <u>development</u>, levels of <u>processing</u>, and <u>integrative levels</u> discussed in the psychological literature (e.g., in [Cermak and Craik 1979], [Greenberg and Tobach 1987]). These topics received some attention in Sections 3.1 and 8.2.1, but more research, coupled with psychological experimentation, may confirm the importance of supervenience in cognitive modeling.

Further comparisons with the philosophical literature would also be of interest. The history of philosophy is replete with theories of levels, from the taxonomies of Aristotle to Hegel's dialectical hierarchy of Being. Even the anti-systematic, existential philosophy of Kierkegaard propounds a theory of levels—the individual ascends through concentric aesthetic, ethical, and religious "spheres" of existence. Twentieth Century philosophy provides more fertile sources of comparison, as many level-based theories are expressed in formal logical language. For example, the level-generation relation between events, mentioned in Chapter 4, has already been formalized in the context of AI systems, and bears at least prima facie similarity to the supervenience relation. Philosophical modularity theories, many inspired by Fodor's The Modularity of Mind [Fodor 1983] and closely allied to work in psychology and linguistics, have recently received renewed interest (see, e.g., [Garfield 1987]).

The formal characterization of supervenience has many relatives in recent work on nonmonotonic reasoning systems, and the connections between the various systems have yet to be drawn. In addition to the theories mentioned in Chapter 5, work on preferred subtheories may provide fruitful comparisons; for example, [Brewka 1991] has provided a formal theory of "levels of reliability." Rathmann has recently completed a dissertation entitled Nonmonotonic Semantics for Partitioned Knowledge Bases [Rathmann 1991], in which he investigates several issues that arise in combining independent nonmonotonic reasoning systems. Considerable work has also been conducted on stratified databases, in which an asymmetric dependency relation is defined over the set of rules in a logic program, helping to define the nonmonotonic semantics of the program (see several papers in [Minker 1988]). Comparative research in this area is hindered by the multiplicity of formalisms in which the theories are expressed; for example, Brewka's theory is expressed as an extension of default logic, Rathmann's work is expressed in the framework of circumscription, research on stratified databases is expressed in the context of logic programming, and the formal characterization of supervenience is expressed in an extended argument system. Nonetheless, it appears that these theories have important affinities that should be explored.

The formal characterization of supervenience can also be compared to previous formalizations of supervenience from the philosophical literature, most of which discuss supervenience in terms of equality or indiscernability relations that hold between entities at different levels.[58] An analog in the AI literature may be found in Hobbs's discussions of levels of granularity, in which "abstraction," "simplification," and "idealization" are defined in terms of logical indistinguishability relations [Hobbs 1990]. Hobbs also describes the relation between granularity and ABSTRIPS-style abstraction; his discussions may provide a convenient bridge between the equality-based and nonmonotonic formulations of supervenience.

Philosophical formal theories of supervenience are often expressed in modal logic; this suggests additional avenues for comparative research. A promising route may be to recast nonmonotonic supervenience in default logic. This would allow for comparison to pervious theories via the relations between modal, multi-valued, and default logics that have been studied by others (e.g., [Ginsberg 1986], [Horty 1992], [Konolige 1987]).

Less formal comparisons to the philosophical literature also suggest themselves. Teller summarized several theories of supervenience with the following "Generalized Supervenience-Determination" thesis:

> *Truths of kind S supervene on/are determined by truths of kind P* if and only if any two cases which agree as to truths of kind P also agree as to truths of kind S. [Teller 1984, 145]

Comparison with the account of supervenience developed in this dissertation hinges on the interpretation of "cases" and "agree." On some interpretations there is considerable divergence; two supervenient planning hierarchies with identical bottom levels could differ at higher levels due to variations in high-level goals or initial knowledge. Formulations such as the following, however, hold in general for supervenient planning hierarchies:

[58] See, for example, [Hellman 1992] and the references cited therein.

> If truths at level *n* supervene on/are determined by truths at level *m* then any two cases that agree about truths at level *m* also agree about those truths at level *n* that are obtained from level *m*.

The significance of this and of related formulations, and of their application to areas of philosophy in which supervenience has played a role, are subjects for further study.

The implementation presented in Part III may be extended in a number of ways. Modest improvements to the APE program might improve the performance and utility of the system. A relatively minor revision would allow <u>any</u> set of levels, and any partial order of communication, to be defined for a particular system built using APE. The simulation of parallelism could be enhanced to allow for automatic, dynamic load-balancing amongst virtual processors. While simple mechanisms for "spin control" are part of the current system (see Section 8.8), a more principled approach to the simulation of parallelism might improve performance considerably. Improvements to the knowledge representation systems are also called for, given their impact on current performance (see Section 9.4).

The knowledge representation system could be extended in several ways, allowing for the application of advanced knowledge representation concepts to reasoning in APE. Extensions that could be made with little difficulty include support for nested structures on the blackboard (for example, `(in cat (house dog))`), support for named variables (for example, `(loves :?x :?x)` for something that loves itself), and support for simple forms of property inheritance. More exotic improvements may be indicated for improving efficiency; for example, mechanisms for "forgetting" sentences that are no longer relevant, or for "focusing" on small subsets of the blackboard. Such mechanisms would also invite further comparisons to psychological theory. The current system's semantics are specified procedurally; a formal specification might be more appropriate as the system becomes more complex.

A more challenging extension is the implementation of APE on true parallel hardware. The simulation of parallelism in APE is not complete; certain aspects of true parallel environments (for example, asynchrony) are simulated, while others (for example, read/write conflicts on the blackboard) are not. Hence it is possible that the implementation of APE on true parallel hardware will lead to unexpected interactions, and to revisions of the architectural model.

The use of true parallelism eliminates only one of the system's simulations; HomeBot also runs in a simulated world. An important follow-up to this research is the use of the supervenience architecture for controlling real robots. Again, the HomeBot world simulator simulates some, but not all aspects of actual robotic domains. For example, the possibility of arbitrary change in the world is simulated, but "ambiguous" sensor readings are not.

The use of real robots raises an interesting issue about the application of the supervenience architecture in general. Roboticists must usually contend with noisy, unreliable sensors and effectors, and hence there is a temptation to say that the lower levels are less certain and less reliable than higher levels, at least at levels close to robotic hardware. This might be interpreted to mean that the supervenience relation is "upside down" at the bottom of the control hierarchy—that higher levels "fill in" or correct low level data. The proper approach in such cases, according to the principles outlined in this dissertation, is for higher levels to re-interpret or ignore the low level data, but not to defeat it. For example, an "image region" level might ignore a white pixel in a black field, but it shouldn't defeat the knowledge that a white pixel was sensed at the "pixel sensing" level. Responding to similar claims of "filling in" in human cognition, Dennett writes:

> The fundamental flaw in the idea of "filling in" is that it suggests that the

> brain is providing something when in fact the brain is ignoring something." [Dennett 1991, 356]

In applying the supervenience architecture to real robots, the validity of Dennett's opinion, and of fundamental assumptions behind the supervenience architecture, will be tested. If system designers are compelled to allow higher levels to defeat, and not just to re-interpret, low level knowledge, then the claims of this dissertation will have to be re-evaluated.[59]

The supervenience architecture should also be of use for a range of cognitive modeling tasks quite different from the robotic tasks demonstrated with HomeBot. The idea behind the supervenience architecture is that dividing a system into levels of "distance from the world" is useful for integrating complex cognitive processes with the flow of events in a dynamic environment. Any system that mandates such integration is a candidate for the application of supervenience.

Consider, for example, the performance of improvised music in an ensemble setting. Individual performers integrate the actions of the group into complex, conventionalized models of the musical context, responding to dynamic changes while planning future activities. Although the types of knowledge and actions in this domain are quite different from those in HomeBot, initial support for the applicability of the supervenience architecture can be drawn from a wide range of sources. These sources include Levinson's discussions of <u>aesthetic</u> supervenience [Levinson 1984], Arnheim's discussions of abstraction in the arts [Arnheim 1969], Lerdahl and Jackendoff's multilevel model of music representation [Lerdahl and Jackendoff 1983], and several recent studies of hierarchy in the cognition of music (e.g., [Serafine 1988]).

As recently as 1991 Tenenberg wrote, "The use of abstraction for problem solving

[59]Thanks to Kurt Konolige for pointing out this connection to Dennett's work.

is too little understood yet to be doctrinaire about which types of abstraction, in general, will be best" [Tenenberg 1991, 273]. On the other hand, concepts of "abstraction" are so prevalent in AI that there is a danger that the word will cease to have any significance. Several forms of abstraction have been used for planning in general, and for dynamic-world planning in particular. The concept of supervenience is important because it captures the essential features of these notions of abstraction insofar as they are important for systems that must function in complex, dynamic environments.

This dissertation addressed a problem that has been a catalyst for speculation in cognitive science from Heraclitus to Fodor: the integration of slow, complex, cognitive processes into the rapid flow of events that constitutes our world. In the nuts-and-bolts cognitive science of AI the problem is best handled through the use of hierarchy— different approaches are appropriate at different levels of abstraction. The technical problem then becomes one of integrating and coordinating the resulting multiplicity of approaches. The aim of this dissertation was to highlight the importance of such integrative and coordinative concerns, and to present concrete theoretical and implementational solutions for some of the related problems.

While intelligent, reactive robots are not yet rolling out of our labs, it is clear that such robots will have high-level mental representations that <u>supervene</u> on the low-level perceptual/manual data of current robotic systems. This supervenience can be studied from philosophical, psychological, neuroscientific, and computational perspectives. The construction of systems such as APE/HomeBot allows for experimentation with the principles of cognitive architecture that result from such studies.

Bibliography

Agre, Phillip E., and David Chapman. 1987. Pengi: An Implementation of a Theory of Activity. In Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-87, 268–272.

Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman. 1983. Data Structures and Algorithms. Reading, Massachusetts: Addison-Wesley Publishing Company.

Albus, James S. 1991. Outline for a Theory of Intelligence. IEEE Transactions on Systems, Man, and Cybernetics 21 (May/June): 473–509.

Allen, James F. 1984. Towards a General Theory of Action and Time. Artificial Intelligence 23: 123-154.

Allen, James, James Hendler, and Austin Tate, eds. 1990. Readings in Planning. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Arkin, Ronald C. 1989. Towards the Unification of Navigational Planning and Reactive Control. In AAAI Spring Symposium on Robot Navigation.

Arkin, Ronald C. 1990. Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation. Robotics and Autonomous Systems 6: 105–122.

Arnheim, Rudolf. 1969. Visual Thinking. London: Faber and Faber Limited.

Austin, J. L. 1975. How to Do Things with Words. Cambridge, Massachusetts: Harvard University Press.

Bacchus, Fahiem, and Qiang Yang. 1991. The Downward Refinement Property. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, 286–292.

Bisiani, Roberto, and A. Forin. 1989. Parallelization of Blackboard Architectures and the Agora System. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 137–152. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Blackburn, Simon. 1988. Supervenience Revisited. In Essays on Moral Realism, edited by Geoffrey Sayre-McCord, 59–75. Ithaca and London: Cornell University Press.

Brachman, Ronald J., and Hector J. Levesque, eds. 1985. Readings in Knowledge Representation. Los Altos, California: Morgan Kaufmann Publishers, Inc.

Bresina, John, and Mark Drummond. 1990. Integrating Planning and Reaction: A Preliminary Report. In Planning in Uncertain, Unpredictable, or Changing Environments, edited by James Hendler. SRC TR 90-45, University of Maryland Systems Research Center.

Brewka, Gerhard. 1991. Nonmonotonic Reasoning: Logical Foundations of Commonsense. Cambridge: Cambridge University Press.

Brooks, Rodney A. 1990. A Robust Layered Control System for a Mobile Robot. In Artificial Intelligence at MIT, Volume 2, edited by Patrick Henry Winston, and Sarah Alexandra Shellard, 2–27. Cambridge, Massachusetts: The MIT Press.

Brooks, Rodney A. 1991. Intelligence without Reason. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, San Mateo, California: Morgan Kaufmann Publishers, Inc., 569–595. (Also available as MIT A.I. Memo No. 1293.)

Bunge, Mario. 1960. Levels: A Semantical Preliminary. The Review of Metaphysics XIII (March): 396–406.

Bylander, Tom. 1991. Complexity Results for Planning. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, 274–279.

Carnap, Rudolph. 1960. Elementary and Abstract Terms. In Philosophy of Science, edited by Arthur Danto, and Ernest Nagel, 150–158. New York: World Publishing Company.

Cermak, Laird S., and Fergus I. M. Craik, eds. 1979. Levels of Processing in Human Memory. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Chapman, David. 1987. Planning for Conjunctive Goals. Artificial Intelligence 32: 333–377.

Chapman, David, and Philip Agre. 1986. Abstract Reasoning as Emergent from Concrete Activity. In The 1986 Workshop on Reasoning About Actions and Plans, edited by M. Georgeff, and A. Lansky, 411–424. Los Altos, California: Morgan Kaufmann Publishers, Inc.

Chenoweth, Stephen V. 1991. On the NP-Hardness of Blocks World. In Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI-91, 623–628.

Chrisman, Lonnie, Rich Caruana, and Wayne Carriker. 1991. Intelligent Agent Design Issues: Internal Agent State and Incomplete Perception. In Working Notes of the 1991 Fall Symposium on Sensory Aspects of Robotic Intelligence, American Association for Artificial Intelligence, 18–25.

Christensen, Jens. 1991. Automatic Abstraction in Planning. Report No. STAN-CS-91-1357, Department of Computer Science, Stanford University.

Churchland, Patricia S., and Terrence J. Sejnowski. 1988. Perspectives on Cognitive Neuroscience. Science 242 (November): 741–745.

Cohen, Paul R. 1991. A Survey of the Eighth National Conference on Artificial Intelligence: Pulling Together or Pulling Apart?. AI Magazine 12 (Spring): 16–41.

Cohen, Phillip R., Jerry Morgan, and Martha E. Pollack, eds. 1990. Intentions in Communication. Cambridge, Massachusetts: The MIT Press.

Corkill, Daniel D. 1989. Design Alternatives for Parallel and Distributed Blackboard Systems. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 99–136. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Craig, Iain D. 1989. The Cassandra Acrhitecture: Distributed Control in a Blackboard System. Chichester, England: Ellis Horwood Limited, Publishers.

Cratty, Briant J. 1973. Movement Behavior and Motor Learning. Philadelphia: Lee & Febiger.

Currie, Ken, and Austin Tate. 1991. O-plan: the open planning architecture. Artificial Intelligence 52 (November): 49–86.

Davis, Lawrence. 1979. Theory of Action. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Davis, Philip J., and Reuben Hersh. 1986. Descartes' Dream. Boston: Houghton Mifflin Company.

Davis, Randall. 1991. A Tale of Two Knowledge Servers. AI Magazine 12 (Fall): 118–120.

Dean, Thomas. 1985. Temporal Imagery: An Approach to Reasoning about Time for Plannng and Problem Solving. YALEU/CSD/RR #433, Yale University Department of Computer Science.

Dean, Thomas, and Mark Boddy. 1988. An Analysis of Time-Dependent Planning. In Proceedings of AAAI-88, 49-54.

Dennett, Daniel C. 1978. Why You Can't Make a Computer that Feels Pain. In Brainstorms, by Daniel C. Dennett, 190–229. Montgomery, Vermont: Bradford Books, Publishers, Inc.

Dennett, Daniel C. 1991. Consciousness Explained. Little, Brown and Company.

Dodhiawala, Rajendra T., N. S. Sridharan, and Cynthia Pickering. 1989. A Real-Time Blackboard Architecture. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 219–237. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Doyle, R. J., D. J. Atkinson, and R. S. Doshi. 1986. Generating Perception Requests and Expectations to Verify the Execution of Plans. In Proceedings of AAAI-86, 81–88.

Drummond, Mark E. 1990. Refining and Extending the Procedural Net. In Readings in Planning, edited by James Allen, James Hendler, and Austin Tate, 667–669. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Durfee, Edmund H. 1990. A Cooperative Approach to Planning for Real-Time Control. In Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control, edited by Katia P. Sycara, 277–283. Defense Advanced Research Projects Agency (DARPA).

Elkan, Charles. 1990. Incremental, Approximate Planning. In Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90, 145–150.

Erman, Lee D., and Victor R. Lesser. 1975. A Multi-level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge. In Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75, 483–490.

Erman, Lee D., Frederick Hayes-Roth, Victor R. Lesser, and Raj D. Reddy. 1980. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. Computing Surveys 12 (June): 213–253.

Erol, Kutluhan, Dana S. Nau, and V. S. Subrahmanian. 1991. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. CS-TR-2797, Department of Computer Science, University of Maryland.

Fehling, Michael R., Art M. Altman, and B. Michael Wilber. 1989. The Heuristic Control Virtual Machine: An Implementation of the Schemer Computational Model of Reflective, Real-Time Problem-Solving. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 191–218. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Fikes, Richard E., and Nils Nilsson. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 5 (2): 189–208.

Findeisen, W., F. N. Bailey, M. Brdys, K. Malinowski, P. Tatjewski, and A. Wozniak. 1980. Control and Coordination in Hierarchical Systems. Chichester, England: John Wiley & Sons.

Firby, James R. 1989. Adaptive Execution in Complex Dynamic Worlds. Doctoral Dissertation, Department of Computer Science, Yale University.

Firby, R. James, and Steve Hanks. 1987. A Simulator for Mobile Robot Planning. In DARPA Knowledge-Based Planning Workshop (Austin, Texas), Defense Advanced Research Projects Agency (DARPA), 23-1–23-7.

Fodor, Jerry A. 1983. The Modularity of Mind. Cambridge, Massachusetts: The MIT Press.

Gale, Richard M. 1971. 'Here' and 'Now'. In Basic Issues in the Philosophy of Time, edited by Eugene Freeman, and Wilfrid Sellars, 72–85. La Salle, Illinois: The Open Court Publishing Co.

Gardner, Howard. 1983. Frames of Mind. New York: Basic Books, Inc., Publishers.

Garfield, Jay L., ed. 1987. Modularity in Knowledge Representation and Natural-Language Understanding. Cambridge, Massachusetts: The MIT Press.

Gat, Erann. 1991. Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots. Doctoral Dissertation (revision 1.1), Virginia Polytechnic Institute and State University.

Gellatly, Angus, Don Rogers, and John A. Sloboda, eds. 1989. Cognition and Social Worlds. Oxford: Clarendon Press.

Georgeff, Michael P. 1990. Planning. In Readings in Planning, edited by James Allen, James Hendler, and Austin Tate, 5–25. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Georgeff, Michael P., and François Felix Ingrand. 1989. Decision-Making in an Embedded Reasoning System. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89, 972–978.

Georgeff, M., and A. Lansky, eds. 1986. The 1986 Workshop on Reasoning About Actions and Plans. Los Altos, California: Morgan Kaufmann Publishers, Inc.

Georgeff, Michael P., and Amy L. Lansky. 1990. Reactive Reasoning and Planning. In Readings in Planning, edited by James Allen, James Hendler, and Austin Tate, 729–734. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Gerard, Ralph W. 1969. Hierarchy, Entitation, and Levels. In Hierarchical Structures, edited by Lancelot Law Whyte, Albert G. Wilson, and Donna Wilson, 215–228. New York: American Elsevier Publishing Company, Inc.

Gillespie, Norman Chase. 1984. Supervenient Identities and Supervenient Differences. The Southern Journal of Philosophy, Spindel Conference 1983: Supervenience XXII (Supplement): 111–116.

Ginsberg, Matthew L. 1986. Multi-valued Logics. In Proceedings of the National Conference on Artificial Intelligence, AAAI-86, 243–247.

Ginsberg, Matthew L. 1991. The Computational Value of Nonmonotonic Reasoning. In Princinciples of Knowledge Representation and Reasoning: Proceedings of the Second International Conference, edited by James Allen, Richard Fikes, and Erik Sandewall, 262–268. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Goldman, Alvin I. 1970. A Theory of Human Action. Princeton, New Jersey: Princeton University Press.

Grafman, Jordan. 1989. Plans, Actions, and Mental Sets: Managerial Knowledge Units in the Frontal Lobes. In Integrating Theory and Practice in Clinical Neuropsychology, Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, 93–138.

Greenberg, Gary, and Ethel Tobach, eds. 1987. Cognition, Language and Consciousness: Integrative Levels. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Gupta, Naresh, and Dana S. Nau. 1991. Complexity Results for Blocks-World Planning. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, 629–633.

Hanks, Steve, and R. James Firby. 1990. Issues and Architectures for Planning and Execution. In Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control, edited by Katia P. Sycara, 59–70. Defense Advanced Research Projects Agency (DARPA).

Hanks, Steve, and Drew McDermott. 1990. Nonmonotonic Logic and Temporal Projection. In Readings in Planning, edited by James Allen, James Hendler, and Austin Tate, 624–640. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Hayes, Patrick J. 1985. The Second Naive Physics Manifesto. In Formal Theories of the Commonsense World, edited by J. R. Hobbs, and R. C. Moore, 1–36. Norwood, New Jersey: Ablex Publishing Corporation.

Hayes, Philip J. 1975. A Representation for Robot Plans. In Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75, 181–188.

Hayes-Roth, Barbara, and Frederick Hayes-Roth. 1979. A Cognitive Model Planning. Cognitive Science 3 (4): 275–310.

Hayes-Roth, Barbara. 1985. A Blackboard Architecture for Control. Artificial Intelligence 26: 251-321.

Hayes-Roth, Barbara. 1990. Dynamic Control Planning in Intelligent Agents. In Planning in Uncertain, Unpredictable, or Changing Environments, edited by James Hendler. SRC TR 90-45, University of Maryland Systems Research Center.

Hellman, Geoffrey. 1992. Supervenience/Determination a Two-Way Street? Yes, but One of the Ways is the *Wrong Way!*. The Journal of Philosophy LXXXIX (January): 42–47.

Hendler, James. 1990. Abstraction and Reaction. In Planning in Uncertain, Unpredictable, or Changing Environments, edited by James Hendler. SRC TR 90-45, University of Maryland Systems Research Center.

Hendler, J., and J. Sanborn. 1988. Monitoring and Reacting: Planning in Dynamic Domains. International Journal of AI and Engineering 3 (April).

Hendler, James A., and James Sanborn. 1987. A Model of Reaction for Planning in Complex Environments. In Proceedings of the Knowledge-Based Planning Workshop, Defense Advanced Research Projects Agency (DARPA), 24-1–24-10.

Hendler, James, and V.S. Subrahmanian. 1990. A Formal Model of Abstraction for Planning. UMIACSWW-TR-90-75 and CS-TR-2480, Department of Computer Science, University of Maryland.

Heuer, H., U. Kleinbeck, and K.-H. Schmidt, eds. 1985. Motor Behavior. Berlin: Springer-Verlag.

Hewett, Micheal, and Barbara Hayes-Roth. 1989. Real-Time I/O in Knowledge-Based Systems. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 269–283. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Hobbs, Jerry R. 1990. Granularity. In Readings in Qualitative Reasoning about Physical Systems, edited by Daniel S. Weld, and Johan de Kleer, 452–455. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Holliday, Mark A., and Mary K. Vernon. 1987. A Generalized Timed Petri Net Model for Performance Analysis. IEEE Transactions on Software Engineering SE-13 (December): 1297–1310.

Horgan, Terence, ed. 1984. Spindel Conference 1983: Supervenience; The Southern Journal of Philosophy, Volume XXII, Supplement.

Horn, Werner, ed. 1990. Causal AI Models. New York: Hemisphere Publishing Corporation.

Horty, John F. 1992. Moral Dilemmas and Nonmonotonic Logic. Journal of Philosophical Logic (forthcoming). A preliminary version appeared in Proceedings of the First International Workshop on Deontic Logic in Computer Science (DEON'91).

Jackendoff, Ray. 1987. Consciousness and the Computational Mind. Cambridge, Massachusetts: The MIT Press.

Jagannathan, Vasudevan. 1989. Realizing the Concurrent Blackboard Model. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 85–97. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Jaques, Elliott, R. O. Gibson, and D. J. Isaac, eds. 1978. Levels of Abstraction in Logic and Human Action. London: Heinemann Educational Books Ltd.

Kambhampati, Subbarao. 1990. Mapping and Retrieval During Plan Reuse: A Validation Structure Based Approach. In Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90, 170–175.

Kambhampati, S., and J. Hendler. 1992. A Validation Structure Based Theory of Plan Modification and Reuse. Artificial Intelligence (in press).

Kinney, David N., and Michael P. Georgeff. 1991. Commitment and Effectiveness of Situated Agents. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, 82–88.

Klagge, James C. 1990. Davidson's Troubles with Supervenience. Synthese 85 (November).

Knoblock, Craig Alan. 1991a. Automatically Generating Abstractions for Problem Solving. CMU-CS-91-120, School of Computer Science, Carnegie Mellon University.

Knoblock, Craig A. 1991b. Search Reduction in Hierarchical Problem Solving. In Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI-91, 686–691.

Knoblock, Craig A., Josh D. Tenenberg, and Qiang Yang. 1991. Characterizing Abstraction Hierarchies for Planning. In Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI-91, 692–697.

Konolige, Kurt. 1987. On the Relation Between Default and Autoepistemic Logic. In Readings in Nonmonotonic Reasoning, edited by Matthew L. Ginsberg, 195–226. Los Altos, California: Morgan Kaufmann Publishers, Inc.

Konolige, Kurt. 1988. Hierarchic Autoepistemic Theories for Nonmonotonic Reasoning: Preliminary Report. In Non-Monotonic Reasoning: Proceedings of the 2nd International Workshop, edited by M. Reinfrank, J. de Kleer, M.L. Ginsberg, and E. Sandewall, 42–59. Berlin: Springer-Verlag.

Korf, Richard E. 1987. Planning as Search: A Quantitative Approach. Artificial Intelligence 33: 65–88.

Kraus, Sarit, Madhura Nirkhe, and Donald Perlis. 1990. Toward Fully Deadline-Coupled Planning. In Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control, edited by Katia P. Sycara, 100–108. Defense Advanced Research Projects Agency (DARPA).

Kuokka, Daniel R. 1990. The Deliberative Integration of Planning, Execution, and Learning. CMU-CS-90-135, Carnegie Mellon University School of Computer Science.

Kyburg, Henry E. Jr., Ronald P. Loui, and Greg N. Carlson, eds. 1990. Knowledge Representation and Defeasible Reasoning. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Leith, Philip. 1990. Formalism in AI and Computer Science. New York and London: Ellis Horwood Limited.

Lennon, Kathleen. 1990. Explaining Human Action. La Salle, Illinois: Open Court Publishing Company.

Lerdahl, Fred, and Ray Jackendoff. 1983. A Generative Theory of Tonal Music. Cambridge, Massachusetts: The MIT Press.

Lesser, Victor R., Robert C. Whitehair, Daniel D. Corkill, and Joseph A. Hernandez. 1989. Goal Relationships and Their Use in a Blackboard Architecture. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 9–26. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Lesser, Victor R., Jasmina Pavlin, and Edmund H. Durfee. 1989. Approximate Processing in Real-Time Problem Solving. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 239–268. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Levin, Iris, and Dan Zakay, eds. 1989. Time and Human Cognition. Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Levinson, Jerrold. 1984. A Thousand Entities: Comments on Haugeland's 'Ontological Supervenience'. The Southern Journal of Philosophy, Spindel Conference 1983: Supervenience XXII (Supplement): 13–17.

Liben, Lynn S., Arthur H. Patterson, and Nora Newcombe, eds. 1981. Spatial Representation and Behavior Across the Life Span. New York: Academic Press.

Lin, Fangzhen. 1991. A Study of Nonmonotonic Reasoning. Report No. STAN-CS-91-1385, Department of Computer Science, Stanford University.

Lin, Fangzhen, and Yoav Shoham. 1989. Argument Systems: a uniform basis for nonmonotonic reasoning. In Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, KR-89, San Mateo, California: Morgan Kaufmann Publishers, Inc., 245–255.

Maes, Pattie. 1990. Situated Agents Can Have Goals. In Designing Autonomous Agents, edited by P. Maes, 49–70. Cambridge, Massachusetts: The MIT Press.

Marr, David. 1982. Vision. W. H. New York: Freeman and Company.

McCarthy, John. 1968. Programs with Common Sense. In Semantic Information Processing, edited by M. Minsky, 403–418. Cambridge and London: The MIT Press.

McDermott, D. 1978. Planning and Acting. Cognitive Science 2 (2): 71-109.

McDermott, Drew. 1990. Planning Reactive Behavior: A Progress Report. In Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control, edited by Katia P. Sycara, 450–458. Defense Advanced Research Projects Agency (DARPA).

Mesarovic, M. D., and D. Macko. 1969. Foundations for a Scientific Theory of Hierarchical Systems. In Hierarchical Structures, edited by Lancelot Law Whyte, Albert G. Wilson, and Donna Wilson, 29–50. New York: American Elsevier Publishing Company, Inc.

Meystel, A. 1987. Theoretical Foundations of Planning and Navigation for Autonomous Robots. International Journal of Intelligent Systems II: 73–128.

Miller, Richard B. 1990. Supervenience is a Two-Way Street. The Journal of Philosophy LXXXVII (December): 695-701.

Minker, Jack, ed. 1988. Foundations of Deductive Databases and Logic Programming. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Mitchell, Joseph S. B., David W. Payton, and David M. Keirsey. 1987. Planning and Reasoning for Autonomous Vehicle Control. International Journal of Intelligent Systems II: 129–198.

Moore, Robert C. 1987. Semantical Considerations on Nonmonotonic Logic. In Readings in Nonmonotonic Reasoning, edited by Matthew L. Ginsberg, 127–136. Los Altos, California: Morgan Kaufmann Publishers, Inc.

Morris, William, ed. 1978. The American Heritage Dictionary of the English Language. Boston: Houghton Mifflin Company.

Mukerjee, Amitabha, and Gene Joe. 1990. A Qualitative Model for Space. In Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90, 721–727.

Nilsson, Nils J. 1980. Principles of Artificial Intelligence. Los Altos, California: Morgan Kaufmann Publishers, Inc.

Nilsson, Nils J. 1991. Toward Agent Programs with Circuit Semantics. Artificial Intelligence.

Payton, David W., J. Kenneth Rosenblatt, and David M. Keirsey. 1990. Plan Guided Reaction. IEEE Transactions on Systems, Man, and Cybernetics 20 (November/December): 1370–1382.

Peterson, James Lyle. 1981. Petri Net Theory and the Modeling of Systems. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Piaget, Jean, and Bärbel Inhelder. 1967. The Child’s Conception of Space. New York: W. W. Norton & Company, Inc.

Piaget, Jean, and Bärbel Inhelder. 1969. The Psychology of the Child. New York: Basic Books, Inc.

Pollack, Martha E. 1986. Inferring Domain Plans in Question-Answering. SRI Technical Note #403, Menlo Park, California: SRI International.

Pollack, Martha E., and Marc Ringuette. 1990. Introducing the Tileworld: Experimentally Evaluating Agent Architectures. In Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90, 183–189.

Pollock, John L. 1989. How to Build a Person: A Prolegomenon. Cambridge, Massachusetts: The MIT Press.

Post, John F. 1984. Comment on Teller. The Southern Journal of Philosophy, Spindel Conference 1983: Supervenience XXII (Supplement): 163–167.

Potegal, Michael, ed. 1982. Spatial Abilities. New York: Academic Press.

Plaisted, David A. 1981. Theorem Proving with Abstraction. Artificial Intelligence 16: 47–108.

Pylyshyn, Zenon W., ed. 1987. The Robot's Dilemma : The Frame Problem in Artificial Intelligence. Norwood, New Jersey: Ablex.

Rathmann, Peter. 1991. Nonmonotonic Semantics for Partitioned Knowledge Bases. Report No. STAN-CS-91-1371, Department of Computer Science, Stanford University.

Raulefs, Peter. 1989. Toward a Blackboard Architecture for Real-Time Interactions with Dynamic Systems. In Blackboard Architectures and Applications, edited by V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, 285–299. Boston: Academic Press, Inc., Harcourt Brace Jovanovich, Publishers.

Reichenbach, Hans. 1958. The Philosophy of Space & Time. Translated by Maria Reichenbach and John Freund. New York: Dover Publications, Inc.

Rich, Charles. 1985. The Layered Architecture of a System for Reasoning about Programs. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, IJCAI-85, 540–546.

Roitblat, H. L. 1991. Cognitive Action Theory as a Control Architecture. In From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior, edited by Jean-Arcady Meyer, and Stewart W. Wilson, 444–450. Cambridge, Massachusetts: The MIT Press.

Rosenschein, S. J., and L. P. Kaelbling. 1986. The Synthesis of Machines with Provable Epistemic Properties. In Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge, edited by J. F. Halpern, 83–98. Los Altos, California: Morgan Kaufmann Publishers, Inc.

Sacerdoti, Earl D. 1974. Planning in a Hierarchy of Abstraction Spaces. Artificial Intelligence 5 (2): 115–135.

Sacerdoti, Earl D. 1975. The Nonlinear Nature of Plans. In Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75, 206–214.

Sacerdoti, Earl D. 1977. A Structure for Plans and Behavior. New York: Elsevier North-Holland, Inc.

Sanborn, James C. 1989. Dynamic Reaction: Controlling Behavior in Dynamic Domains. CS-TR-2184, Department of Computer Science, University of Maryland.

Sayre-McCord, Geoffrey. 1988. Moral Theory and Explanatory Impotence. In Essays on Moral Realism, edited by Geoffrey Sayre-McCord, 256–281. Ithaca and London: Cornell University Press.

Schank, Roger C. 1982. Dynamic Memory. Cambridge: Cambridge University Press.

Schoppers, M. J. 1987. Universal Plans for Reactive Robots in Unpredictable Environments. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence, IJCAI-87, 1039–1046.

Schoppers, Marcel, and Ted Linden. 1990. The Dimensions of Knowledge Based Control Systems and the Significance of Metalevels. In Planning in Uncertain, Unpredictable, or Changing Environments, edited by James Hendler. SRC TR 90-45, University of Maryland Systems Research Center.

Schoppers, Marcel, and Richard Shu. 1990. An Implementation of Indexical/Functional Reference for Embedded Execution of Symbolic Plans. In Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control, San Mateo, California: Morgan Kaufmann Publishers, Inc., 490–496.

Serafine, Mary Louise. 1988. Music as Cognition. New York: Columbia University Press.

Shallice, Tim. 1991. Précis of *From Neuropsychology to Mental Structure.* Behavioral and Brain Sciences 14 (September): 429–469.

Shoham, Yoav. 1988. Reasoning about Change. Cambridge, Massachusetts: The MIT Press.

Shwayder, D. S. 1965. The Stratification of Behaviour. London: Routledge & Kegan Paul, New York: The Humanities Press.

Simon, Herbert A. 1969. The Architecture of Complexity. In The Sciences of the Artificial, by Herbert A. Simon, 84–118. Cambridge Massachusetts: The MIT Press.

Spector, Lee, and James Hendler. 1990a. An Abstraction-Partitioned Model for Reactive Planning. In Proceedings of the Fifth Rocky Mountain Conference on Artificial Intelligence (RMCAI-90), Las Cruces, New Mexico: New Mexico State University, 155–160.

Spector, Lee, and James Hendler. 1990b. Knowledge Strata: Reactive Planning with a Multi-level Architecture. UMIACS-TR-90-140, CS-TR-2564, Department of Computer Science, University of Maryland.

Spector, Lee, and James Hendler. 1991a. The Supervenience Architecture. In The Proceedings of the IJCAI-91 Workshop on Theoretical and Practical Design of Rational Agents, Sydney, Australia.

Spector, Lee, and James Hendler. 1991b. The Supervenience Architecture. In Proceedings of the AAAI Fall Symposium on Sensory Aspects of Robotic Intelligence, Asilomar, California, 93–100.

Steele, Guy. 1990. Common Lisp. Digital Press, Digital Equipment Corporation.

Stefik, Mark. 1981. Planning with Constraints (MOLGEN: Part 1). Artificial Intelligence 16 (2): 111–140.

Steiner, Mark. 1979. Quine and Mathematical Reduction. In Essays on the Philosophy of W. V. Quine, edited by Robert W. Shahan, and Chris Swoyer, 133–143. Norman: University of Oklahoma Press.

Suchman, Lucy A. 1987. Plans and Situated Actions. Cambridge: Cambridge University Press.

Sycara, Katia P., ed. 1990. Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control. Defense Advanced Research Projects Agency (DARPA).

Tate, Austin. 1976. Project Planning Using a Hierarchic Non-Linear Planner. Department of Artificial Intelligence Research Report No. 25, Edinburgh: University of Edinburgh.

Tate, Austin. 1977. Generating Project Networks. In Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-77, 888–893.

Teller, Paul. 1984. A Poor Man's Guide to Supervenience and Determination. The Southern Journal of Philosophy, Spindel Conference 1983: Supervenience XXII (Supplement): 137–162.

Tenenberg, Josh D. 1987. Preserving Consistency across Abstraction Mappings. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence, IJCAI-87, 1011–1014.

Tenenberg, Josh D. 1991. Abstraction in Planning. In Reasoning About Plans, by James F. Allen, Henry A. Kautz, Richard N. Pelavin, and Josh D. Tennenberg, 213–283. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Thalberg, Irving. 1977. Perception, Emotion & Action. New Haven: Yale University Press.

Vere, Steven, and Timothy Bickmore. 1990. A Basic Agent. Computational Intelligence 6: 41-60.

Washington, Richard, and Barbara Hayes-Roth. 1990. Abstraction Planning in Real-Time. In Planning in Uncertain, Unpredictable, or Changing Environments, edited by James Hendler. SRC TR 90-45, University of Maryland Systems Research Center.

Weinberg, Julius. 1973. Abstraction in the Formation of Concepts. In Dictionary of the History of Ideas, edited by Philip P. Wiener, 2–9. New York: Charles Scribner's Sons.

Weld, Daniel S., and Johan de Kleer, eds. 1990. Readings in Qualitative Reasoning about Physical Systems. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Whyte, Lancelot Law. 1969. Structural Hierarchies: A Challenging Class of Physical and Biological Problems. In Hierarchical Structures, edited by Lancelot Law Whyte, Albert G. Wilson, and Donna Wilson, 3–16. New York: American Elsevier Publishing Company, Inc.

Whyte, Lancelot Law, Albert G. Wilson, and Donna Wilson, eds. 1969. Hierarchical Structures. New York: American Elsevier Publishing Company, Inc.

Wilensky, Robert. 1983. Planning and Understanding. Reading, Massachusetts: Addison-Wesley Publishing Company.

Wilkins, David E. 1988. Practical Planning: Extending the Classical AI Planning Paradigm. San Mateo, California: Morgan Kaufmann Publishers, Inc.

Williams, Theodore J., ed. 1985. Analysis and Design of Hierarchical Control Systems. Amsterdam: Elsevier Science Publishers, B.V.

Yang, Qiang, and Josh D. Tenenberg. 1990. ABTWEAK: Abstracting a Nonlinear, Least Commitment Planner. In Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90, 204-209.