Using Abstraction to Interleave Planning and Execution

Illah R. Nourbakhsh
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
illah@cs.cmu.edu

Abstract
Abstraction is a technology that enables *interleaved planning and execution*, the much sought-after quality of a robot architecture that can both plan and act (and plan and act and plan...) in short order. This paper presents a definition for Abstraction and for an architecture composed of a collection of abstractions, or an *Abstraction System*. We span both the theoretical extreme of formally defining abstraction and the empirical extreme of implementing an Abstraction System on a real-world robot and collecting compelling experimental results.

1 What is Abstraction?

Abstraction is a term that has been used by many to denote some strategic form of problem simplification. Particular instances of abstraction can be found in early planning literature (Sacerdoti 1974), while more recent, formal discourse in the planning community has led to crisp definitions of abstraction (Holte et al. 1996).

The most elemental ingredient of abstraction is a representational transformation. An abstraction can perform a transformation on the original search space, yielding a new, less complex search space in which discrimination between (irrelevant) details has been removed. More formally:

> *An **Abstraction** is a mapping from the original search space to an abstract search space, such that every state in the original search space is mapped to exactly one state in the abstract search space.*

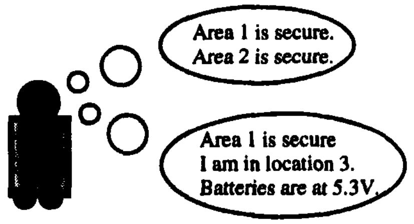

Figure 1: A security robot reasoning at two levels of abstraction

This surjective mapping of states, known as State Abstraction, is fully general. Even in the case of continuous worlds, volume-based reasoning can reduce representation requirements to finite degrees, thereby allowing this definition to function without difficulty (Erdmann 1990).

Consider Figure 1. For the security robot shown, an abstract space can completely ignore the position of the robot, differentiating world state only on the basis of the security of areas on the robot's map. This abstraction will be useful when reasoning strategically about highest-level goals (keep all areas secure). Note, however, that the abstract problem space does not obviate the need for the ground problem space, for the robot must reason about its own position at *some* level.

Not surprisingly, the surjective mapping alone does not suffice for defining useful abstractions. After all, arbitrary partitioning of a set of states is possible, and not every partition proves to be useful during planning. Those partitions that are useful are limited to those that can be *refined*, and so we define a *sound abstraction*:

> *A **Sound Abstraction** is an abstraction from the ground search space G to the abstract search space A such that, for every arc between two states in A there is a conditional plan between corresponding states in G.*

This definition, which places constraints on the arcs of the abstract search space, provably ensures that whenever there is a path between two states in an abstract space, there must be an executable conditional plan from the corresponding initial conditions to the corresponding goal conditions in the ground problem space.

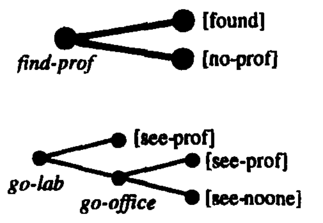

Figure 2: In perceptual abstraction, the abstract percepts *found* and *no-prof* are implemented via a ground conditional plan.

This definition also clarifies an additional, crucial point: the search spaces we discuss can encode *conditional* plans, meaning that we are presenting a formalism for both action and perception.

Perception is formally a guarantee of run-time information gain. Indeed, this information gain can be so important to successful plan execution that some architectures explicitly plan to a perceptual branch, then execute the partial conditional plan early (Olawski et al. 1993). Abstraction results in new problem spaces that also contain perceptual branch points, just like a ground problem space. The difference is that the abstract perceptual branch is, in fact, implemented via an entire conditional plan, consisting of action and perception. In previous literature, this concept has been called *perceptual actions* (Etzioni et al. 1992) and is often wrongly relegated to ground-level planning spaces. Figure 2 demonstrates such an abstract percept and its ground-level implementation.

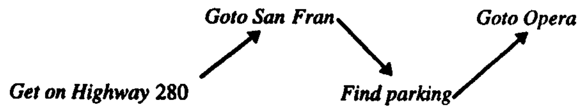

Figure 3: The abstract problem space for an opera-attending robot may contain only states and actions of a coarse-grained nature, ignored such specifics as the location of the car keys, the amount of gas in the car's tank, etc.

Figure 3 demonstrates a simple example of an abstraction that can aid in a realistic planning scenario. The abstract problem space ignores fine-grained motion as well as a host of temporally short actions. Planning in the abstract space can be done quickly all the way to the goal, which is to reach the Opera. Then, the first step of this abstract conditional plan can be refined, providing a provably correct subgoal to the ground problem space and thereby making the ground-level planning problem tractable.

It is important to note that this simple example fits the formally crisp concept of *sound abstraction* defined above. This intuitive approach to using abstraction is grounded in sufficient formal detail to enable provable properties for the system as a whole (Nourbakhsh 1997).

2 Abstraction Systems

A single level of abstraction usually does not suffice for any but the most trivial of problems. A popular conclusion is to generalize to the *Abstraction Hierarchy*, which simply defines abstractions recursively to create a linearly ordered set of problem spaces ranging from the most abstract to the ground problem space (Sacerdoti 1974), (Knoblock 1994).

The computational gains of doing abstraction can be shown to carry over to this recursive case of Abstraction Hierarchies; however, one important shortcoming has led us to an alternative.

Abstraction is about strategic simplification. Given a particular set of initial conditions and goal conditions (i.e. a *context*) simplification is often able to eliminate factors from consideration that are irrelevant to the problem at hand. But such factors change from problem to problem, even within one problem domain.

For instance, the location of the robot's raincoat may be irrelevant on a sunny day but crucial if rain is in the forecast and the robot's tasks take it out of doors. Therefore, it makes sense for there to be alternative abstractions of a ground problem space, since each alternative will be applicable in a limited set of contexts.

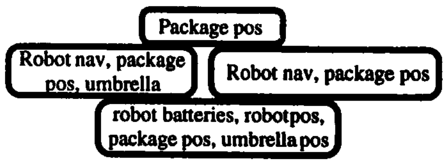
Package pos
Robot nav, package pos, umbrella
Robot nav, package pos
robot batteries, robotpos, package pos, umbrella pos

Figure 4: An abstraction system for a delivery robot that operates in the rain sometimes can have *parallel problem spaces* in which rain and umbrellas are either dealt with or not handled. The position of the umbrella becomes an irrelevant feature—but only in dry weather.

We formalize this concept by defining an *Abstraction System* as a partial order of abstractions:

> *An **Abstraction System** is a set of search spaces that are partially ordered by the abstraction operator. The partial order must have a unique minimal element, called the ground problem space.*

Figure 4 depicts an abstraction system for a delivery robot that is sometimes exposed to rainy weather. In this and in other cases, it is important to note that it is not possible to transform this partial order into any desirable total order—the problem is truly a partial-order one[1].

Creating sets of models with partial-order relationships between them is not new. For example, (Nayak et al. 1992) used partial orders of models in the context of model-based reasoning and diagnosis. Here, we apply

[1] There is added complexity that we will not describe here: in partial-order systems it is important to guarantee that consistent state-mapping occurs via all paths through the abstract spaces. This *self-consistency* property can be formalized and is necessary if soundness and completeness is to be proven. For details, see (Nourbakhsh 1997).

similar techniques to defining an architecture for abstraction, which we shall then use to interleave planning and execution on a robot platform.

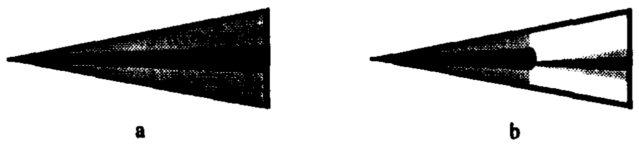

Figure 5: A standard exponential search (a) is transformed by a subgoal in (b), denoted by a black oval, into an initial search out to the subgoal, execution of that partial plan, then resumption of planning to the final goal.

To enable interleaving planning and execution, we must devise a control algorithm that makes use of opportunistic execution during the planning process. Figure 5a depicts a standard search space at the ground problem space.

Figure 5b depicts the same search space when a higher, abstract search space provides a subgoal. The subgoal enables planning and execution at the ground problem space level to be interleaved with real-world execution. This interleaving *and* the subgoaling provided by abstraction both result in search space savings, denoted by the white, unsearched area of the ground problem space.

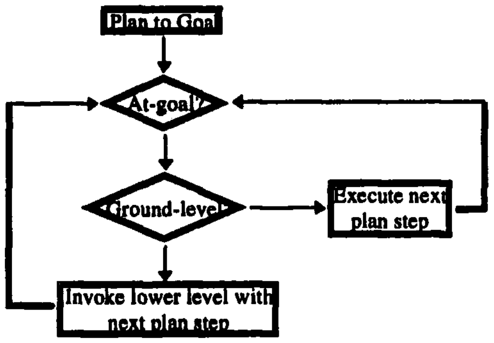

Figure 6: A flowchart description of the control algorithm used in every problem space in the abstraction system.

In order to implement the aforementioned flow of control between abstraction levels in an abstraction system, each problem space must opportunistically refine plan steps when they are sure to be on a path to the goal, given subgoals from higher-level abstract problem spaces. Figure 6 is a flowchart that depicts the algorithm that must be implemented in each problem space. Note that the concept of opportunistic refinement applies to every level except the ground problem space; at this level, refinement equals real-world execution.

An obvious question during the refinement process is: which lower problem space should be chosen, since a partial order may offer more than one possibility? The short answer is that one approach that certainly succeeds is to call all qualified problem spaces in parallel, terminating the ongoing planning processes when one of them succeeds.

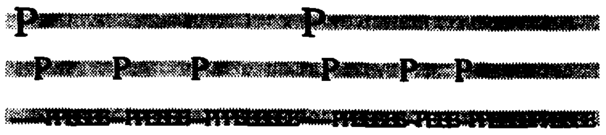

Figure 7: An example of a Behavioral Trace. P indicates planning and E indicates execution. The lowest row is the ground problem space, which is why it includes execution units. The next two higher rows are abstract search spaces.

Figure 7 is an example of a *Behavioral Trace*, a very useful tool for describing the overall, observable behavior of the autonomous system (Genesereth 1993). At the ground level, an observer would see the autonomous system executing actions in groups, with "noop's" in between[2]. As one examines higher levels of abstraction, a waterfall pattern of execution becomes clear; high-level planning always results in a series of refinements all the way to the lowest level, with execution thereafter. Then, planning continues (to the next subgoal) at a higher level and the same cycle ensues.

This behavioral trace demonstrates quite clearly that abstraction is enabling interleaving by providing both subgoals and a formally crisp relationship between problem spaces, so that the mapping of that subgoal is well-defined.

As we discuss in the next section, this is quite literally how we have implemented abstraction systems: by formally specifying, in Lisp, the mapping between states and state sets in neighboring abstraction layers. Then, by simply attaching planners to each problem space in the abstraction system, we have demonstrated highly interleaved solutions to purposefully overcomplicated problem scenarios.

3 Implementation: Balin of Smith

In recent years, a popular question has been whether abstraction can, in the final analysis, enable computational savings, or whether the overhead of using multiple search spaces simultaneously offsets any potential search space savings.

[2] Noop is the operator our robot executes when planning ensues. Physically, the robot stands still during planning, and this is precisely the behavioral indication that interleaving is taking place. As it turns out, Balin is at a standstill for such a short time that it is unnoticeable by human senses.

We set out to address this in several ways, one of which was to create a real-world abstraction system that demonstrated measurable cost savings. To stress the need for abstraction, we invented a highly complex scenario in which a robot must satisfy a set of disparate goals and must reason about a large set of environmental properties.

Table 1: Tasks and feature implemented byBalin

Tasks	State Features
navigation	position, obstacles, world map
prof singing	location of profs, songs
hourly chime	temporal indexing
working in rain	umbrella position, manip.

Table 1 summarizes the result of this design effort. Considerations would range from position and obstacle detection to temporal reasoning in order to achieve the real-time goal of hourly chiming. The primary task of the robot would involve Professor happiness—its task was to sing regularly to a set of professors in order to keep them in good spirits. Balin's goal would be to accomplish overall Professor happiness while meeting its other constraints, including hourly chimes for the author and careful reasoning to either stay clear of "rainy" hallways or retrieve its virtual umbrella[3].

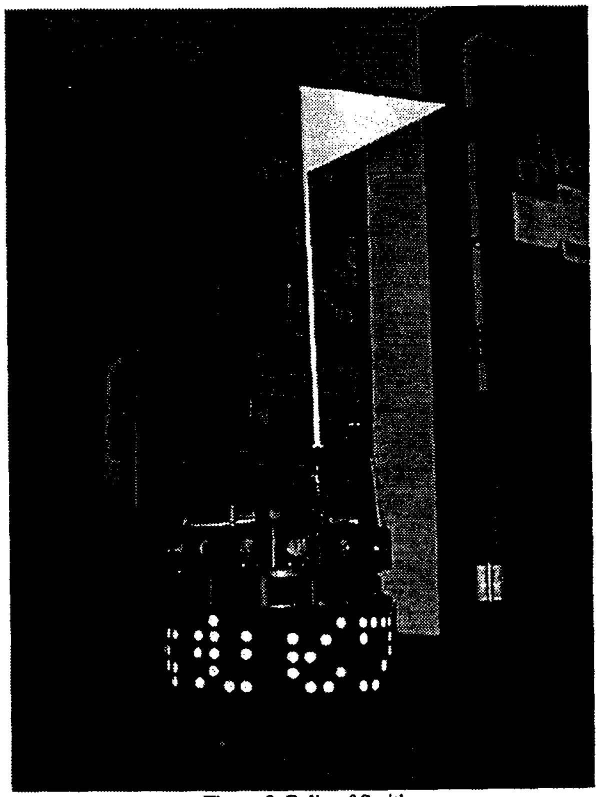
Figure 8: Balin of Smith

[3] It does not really rain in Smith Hall, where Balin operates. This part of the scenario is altogether artificial.

Figure 8 is a picture of Balin in its incarnation. Balin is a Nomad-150 mobile robot that wanders the halls of Smith Hall at Carnegie Mellon's Robotics Institute. The processor is a Macintosh Powerbook 170 Dinosaur running Macintosh Common Lisp 2.01. This MCL image implements *every* problem space of the abstraction system, from the lowest-level ***ground problem space*** to the highest problem space that is three levels of abstraction away.

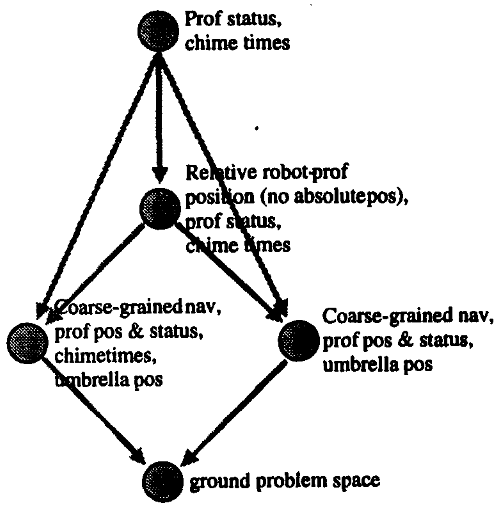
Prof status, chime times
Relative robot-prof position (no absolutepos), prof status, chime times
Coarse-grained nav, prof pos & status, chimetimes, umbrella pos
Coarse-grained nav, prof pos & status, umbrella pos
ground problem space

Figure 9: Balin's abstraction system

Figure 9 depicts the abstraction system we chose for Balin. The ground problem space captures *all* relevant details of the scenario. The scenario is quite complex, resulting in more than 10^9 possible states at this level, 8 possible low-level actions and 32 possible perceptual input vectors, or percepts (Nourbakhsh 1997). Clearly, we succeeded in creating an unwieldy ground problem space in the hopes that abstraction would mitigate the complexity of searching this space.

As one travels up the 4-deep abstraction system, the complexity of the search spaces is diminished rapidly. At level 1, two alternative search spaces distinguish between rainy and dry contexts, in the former case reasoning explicitly about actions and events that involve picking up the umbrella while dropping all `pickup-umbrella` arcs and ignoring the umbrella's position in the latter.

At level 2, the problem space distinguishes only between states sets based on changes in the *relative positions* of the robot and the professors (and the author), thereby ignoring singular information about the position of the robot. Note that this is a valid form of irrelevance at this reasoning level, although it is crucial to reason about one's own position at lower levels of reasoning in order to be able to navigate.

Finally, level 3 incorporates a representation that captures only state distinctions involving the happiness levels of professors and the timeliness of the robot's chimes. These, after all, comprise the top-level goals of the robot, and so this highest-level space acts as a *task scheduler*, sequencing and interleaving satisfaction of these top-level goals. Complete planning to the goal at this level is trivial since the state set size is merely 16 and the branching factor turns out to be, on average, 4.

The goal was to create a complex environment for a robot, so that an abstraction system would be not only feasible but useful. Balin demonstrated both of these points easily, as the results in the following section summarize.

4 Three Results

In this section, we present three types of results: empirical results on Balin demonstrate measurable performance improvement, as something of an existence proof that abstraction can indeed help. A brief computational result demonstrates that abstraction saves at the exponent-level. Finally, we summarize formal results that enable architectural completeness and soundness to be proven for the model of abstraction systems.

Empirical Results

Over the course of approximately one week, we exercised Balin's abstraction system in Smith Hall (Carnegie Mellon's Robotics Institute) and collected planning-cost data. Execution runs lasted approximately one hour on these runs.

Table 2: Planning time results forBalin experiments

	Planning time (sec)	No abstraction
Level 3	0.09	0.09
Level 2	0.19	7.0
Level 1	0.31	>6h
Level 0	2.13	--

Table 2 summarizes the total planning time for Balin at each level of abstraction. The rightmost column indicates the total planning time at that level of abstraction if the abstraction system is not used. Note that the top-most level suffers no computational setback in this case, since it receives no subgoal information from a higher level anyway, even when an abstraction system is in place.

As is indicated by the table, ground level planning with no abstraction was impossible. Planning at the first level of abstraction is tractable, but actually takes longer than the entire execution-time!

A final note about this table regards the relative sizes of planning times in each space. Note that the total planning time in successively higher problem spaces diminishes quickly. This is the computational overhead of abstraction, and the fact that the total overhead is much smaller than the total planning time at the ground level demonstrates that it is quite possible for the computational gains of abstraction to handily outweigh its overhead.

Computational Savings

Planning is traditionally a search performed on the exponential search space. If there are a actions and p percepts in this search space, then planning is exponential in the length of the plan: $p^k a^k$.

Conventional means of abstraction offer search space saving because the abstract problem-solving process informs the ground search space with subgoals, reducing the depth to which the planning system must blindly search. If the original solution is of length k and the abstract subgoal 'cuts' at the midpoint of the solution, then we transform the search space size from $p^k a^k$ to $p^{k/2}a^{k/2} + p^{k/2}(p^{k/2}a^{k/2}) = p^{k/2}a^{k/2}(1+p^{k/2})$.

Now consider interleaving planning and execution. Suppose that the subgoals provided by the abstraction are guaranteed to be correct (a *sound abstraction*). In that case, ground level planning can stop after a conditional solution to the next unachieved subgoal is in hand. Once the system executes that solution, reaching a particular node at the fringe of the plan, planning resumes.

For our previous formula, this further reduces the search space to $2p^{k/2}a^{k/2}$. In the general case, when abstraction yields l subgoals for a k-step problem, interleaving abstraction and execution yields the following savings at the ground level:

(1) $p^k a^k$ (no abstraction)
(2) bounded by $p^{k/l}a^{k/l}(lp^k)$ (with abstraction)
(3) bounded by $lp^{k/l}a^{k/l}$ (interleaving + abstraction)

The intuition behind these savings, which are indeed occurring in the exponent, is that we make maximal use of run-time information gain (in the form of sensory feedback) to reduce perceptual branching *in addition* to the well-known reduction in effectory branching.

Formal Properties

A further result stems from the fact that our definition of abstraction and abstraction systems fits formally within the mathematical framework of state and property representations. By leaning on the mathematical and graph-theoretic foundations of this formal work and appealing to formal definitions of soundness and completeness for *robot architectures* (Nourbakhsh ?), we are able to prove soundness and completeness.

> Theorem 1: *There exists a search algorithm ABS that is sound and complete for all consistent, sound abstraction systems.*

In this paper, we have not delved into sufficient detail to present the ABS algorithm fully; it is, however, a somewhat complex extension of the flowchart shown in Figure 6.

In the context of a robot architecture, *soundness* indicates that the system will never demonstrate a false positive on goal achievement, assuming the ground level problem space is loyal to the real world. *Completeness* indicates that the system will always achieve the goal if there exists a conditional plan in the ground space from the initial conditions to the goal conditions[4].

Some readers will find this formal result to be meaningless, as they care primarily about the utility of the concepts presented. Another group of readers will find this to be the most compelling result, for it shows that a practical, implementable architecture can be sufficiently formal as to attain well-behaved and desirable mathematical properties.

5 Related Work

The majority of research in the area of abstraction has focused on the speedup provided by abstraction, via subgoaling, in the context of planning only (Holte et al. 1996), (Knoblock 1994). Although analysis of such speedup is possible using the abstraction system we have defined here, we have chosen instead to concentrate on producing a methodology that allows abstraction to be leveraged into the interleaving of planning and execution.

Instances of this particular use of abstraction can be found in both the motion planning and manipulation literature. (Donald 1989), (Lazanas & Latombe 1995) serve as two examples of the motion planning community's use of TC actions. The TC, or termination condition action assumes that actions are *durative* and have associated termination criteria. Upon execution, an action is engaged and remains in force until the termination criteria is met.

Lazanas & Latombe effect multistep planning in the TC action domain using an iterative algorithm, nondeterministically choosing a landmark or a set of landmarks as the goal of the next TC action. The formation of landmarks and their use as subgoals is in fact a representation of the initial motion planning problem in a more abstract search space. In this more abstract search space, the action branching factor, the perceptual branching factor and the solution length are all decreased.

In the motion planning community, (Lozano-Perez et al. 1984) demonstrate that a discrete number of directional backprojections can be used during search instead of the examination of an infinite set of continuous backprojections. This is precisely a form of state abstraction, as formalized by our definition of abstraction. Indeed, Bruce Donald (Donald 1989) also demonstrates this form of abstraction through his *critical slice* mechanism, which is a method for discretization while preserving soundness and completeness.

6 Conclusions

We have presented a general, formal framework for representing and reasoning about abstraction systems. Our approach is of particular interest because it has been successfully tested on a real-world robot that was placed in a purposefully "hyper-complex" world. Furthermore, computational and formal results demonstrate that abstraction can provide savings at the exponent-level while preserving the soundness and completeness of the underlying planner.

The partial order concept at the heart of our definition of Abstraction Systems offers a method for reasoning about domain characteristics that are relevant to a particular problem at hand. As such, the partial order representation allows the implementation of very intuitive notions of irrelevance.

Many topics remains to be addressed in this discipline. The automatic generation of abstraction systems and the automatic selection of the right abstract search space during refinement are two such open topics.

Acknowledgments

Michael Genesereth, Daphne Koller and Lise Getoor have all played key roles in the development of the ideas presented in this paper. The Charles Stark Draper Laboratory provided funding for this research. Stanford

[4] Assuming (1) the world is static, and (2) you may have to wait a long, long time.

University's Robotics Laboratory provided the Nomad 150 mobile robot.

References

Donald, B. 1989. Error Detection and Recovery in Robotics. *Lecture Notes in Computer Science*, 336. Springer-Verlag.

Erdmann, M. 1990. On Probabilistic Strategies for Robot Tasks. Technical Report #1155, Massachusetts Institute of Technology.

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N. And Williamson, M. 1992. An Approach to Planning with Incomplete Information. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 115-125.

Genesereth, M. 1993. Discrete Systems. Course notes for *CS 222*. Stanford, CA: Stanford University.

Holte, R., Mkadmi, T., Zimmer, R. and MacDonald, A. 1996. Speeding Up Problem Solving by Abstraction: A Graph Oriented Approach. *Artificial Intelligence*.

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2).

Lazanas, A. and Latombe, J.-C. 1995. Motion planning with uncertainty: a landmark approach. *Artificial Intelligence*, 76(1-2): 287-317.

Lozano-Perez, T., Mason, M. and Taylor, R. 1984. Automatic Synthesis of Fine-Motion Strategies for Robots. *International Journal of Robotics Research*, 3(1):3-24.

Nayak, P., Joskowicz, L, Addanki, S. 1992. Automated Model Selection using Context-Dependent Behaviors. In *Proceedings, Tenth National Conference on Artificial Intelligence*. AAAI Press.

Nourbakhsh, I. 1997. *Interleaving Planning and Execution for Autonomous Robots*. Kluwer Academic Publishers, Boston.

Olawsky, D., Krebsbach, K. And Gini, M. 1993. An Analysis of Sensor-Based Task Planning. Technical Report #93-94. Minneapolis, Minn.: Academic.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115-135.