Using Abstraction to Interleave Planning and Execution

Illah R. Nourbakhsh[1]
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
illah@cs.cmu.edu

ABSTRACT

Interleaving planning and execution is a much sought after quality of a robot architecture. It allows the robot to plan and act in a timely manner. In this paper, we present a robot architecture that uses abstraction to enable interleaved planning and execution. We define an *Abstraction System* as a partially ordered collection of abstract spaces. We also describe the implementation of an Abstraction System on a real-world robot, leading to compelling experimental results in addition to analytical results.

KEYWORDS: abstraction, interleaving, planning

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 [1], while more recent discourse in the planning and search communities has produced a number of formal definitions for abstraction [2,3].

Although many of these works disagree in detail, the common thread throughout is that abstraction involves a representational transformation. An abstraction performs a transformation on the original search space, yielding a new, less complex search space in which discrimination between (irrelevant) details has been mitigated.

Two fairly distinct approaches to abstraction appear in the literature. In one case, researchers propose to create abstract reasoning spaces by performing transformational operations on *arcs* in the state space; in the other case, abstraction is effected by first redefining the *nodes* of the search space. We will refer to these two approaches respectively as *action-based* and *state-based abstraction.*

Action-based abstraction often strives to combine similar actions or to group actions together to form macro operators [4]. The refinement process following action-based abstraction can be extremely efficient, as the refinement of a macro-operator may require little or no search.

State-based abstraction is the less popular technique in which an abstract space represents world state at a coarser level of detail. In state-based abstraction, refinement may involve a significant computational effort since the abstract solution provides

[1] Funding provided by The Charles Stark Draper Research Laboratory, #706.

subgoals for the refinement process but does not necessarily aid in the discovery of a pathway from the current state to the subgoal state.

Yet state-based abstraction is powerful in that limited action models (e.g. STRIPS) are not required. In this paper, we demonstrate and evaluate the use of a *partial order* of state-based abstract problem spaces. We believe that the state-based approach is particularly amenable to implementation on real-world mobile robotic platforms because of the ease with which arbitrary action models can be incorporated.

More formally:

We define an ***Abstraction*** *as a mapping from the original search space to an abstract search space, such that every knowledge state in the original search space is mapped to exactly one knowledge state in the abstract search space.*

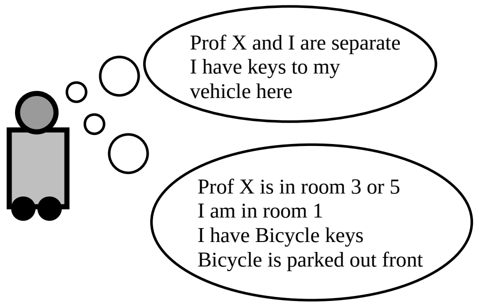
Prof X and I are separate
I have keys to my vehicle here
Prof X is in room 3 or 5
I am in room 1
I have Bicycle keys
Bicycle is parked out front

Figure 1: An example of a robot with state-based abstraction.

This surjective mapping of states is fully general in that any particular abstraction can be formally captured with such a mapping, plus a specification of node connectivity.

Consider Figure 1. For the security robot shown, an abstract space can completely ignore the location of the protagonist and professor, differentiating world state only on the basis of their *relative locations*. Detail is eliminated, not via a straightforward re-discretization of the domain, but by partitioning the set of possible knowledge states in a useful manner.

A more complex example of abstraction in Figure 1 involves vehicle keys. The abstraction is powerful because it groups together individual knowledge states that are superficially quite different. In particular, the state of having bicycle keys with the bicycle out front is indistinguishable, at the abstract level, from the state of having car keys and having the car out front.

Such an abstraction will prove to be useful whenever a solution in the abstract space is a nontrivial 'walk' through a sequence of abstract knowledge states. Of course, the abstract problem space does not obviate the need for ground problem space planning, as it provides only subgoal guidance and not a means to achieve the goal.

In order for such abstractions to be effective during planning, we require a measure of *correctness* from each abstraction. The requisite property is akin to a generalized form of *downward refinement* [5]:

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 nodes in A there is a conditional plan between the set of corresponding nodes in G.*

This definition ensures that whenever there is a path between two abstract knowledge states, then there must be an executable conditional plan from the corresponding *set* of possible initial conditions to the corresponding *set* of goal conditions in the ground problem space.

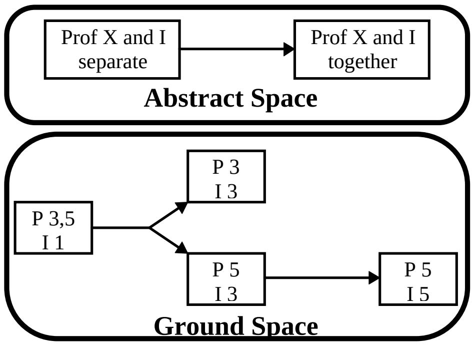

Figure 2: Sound Abstraction example in the domain of Fig. 1. In the abstract problem space, the goal is for the protagonist and Prof. X to be together. In the ground problem space, this subgoal is effected using a conditional plan. The robot checks room 3 and then room 5 if the Professor was not in room 3. The hyperarc denotes a taken action and resulting discrimination between different possible knowledge states based on perceptual values.

Figure 2 demonstrates a partial plan that satisfies the Sound Abstraction constraint. Note the subtle point that a sound abstraction requires the existence of a conditional plan from an initial node that is equivalent to the *union* of the possible knowledge states represented by the source node in an abstract space.

The fact that we employ conditional plans brings up a question that is less commonly addressed: what is the role of perception in abstraction?

Formally, perception is nothing more than a guarantee of run-time information gain. Once this perspective is adopted, the role of perception in abstract problem spaces can be viewed just as in a ground problem space.

The difference between a ground-level perceptual branch and an abstract perceptual branch involves the role of *input*. In a ground perceptual branch, a single, instantaneous input vector determines the sink node at run-time. In an abstract perceptual branch, a *series* of perceptual input vectors, together with the associated output actions, determine the sink node after execution of a plan fragment.

ABSTRACTION SYSTEMS

We now turn our attention to a more macroscopic issue: that of the entire problem-solving representation. Suppose that we are capable of designing an abstract problem space that is both sound and useful. In other words, the overhead of using the abstract problem space should be more than offset by the computational savings provided by itssubgoaling.

Once this has been accepted, it is no great leap to submit that a *set* of abstract problem spaces can be better than just one. Since ground reality may be extremely rich and complex, several abstraction operations may be required to simplify that complexity into a tenable planning domain.

A popular conclusion is to generalize from a single abstract space to an *Abstraction Hierarchy*, which defines abstractions recursively to create a linearly ordered set of problem spaces ranging from the ground problem space to a most abstract problem space [1,3].

Linear abstractions of this form are useful for the elimination of overpowering detail, particularly if that detail can be ignored, at abstract levels of reasoning, regardless of the initial conditions and goals of the system.

More commonly though, different aspects of detail are irrelevant at different times, depending on the system context and goals. Since such factors certainly change, even within one problem domain, we employ a problem space that allows for parallel abstractions.

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.

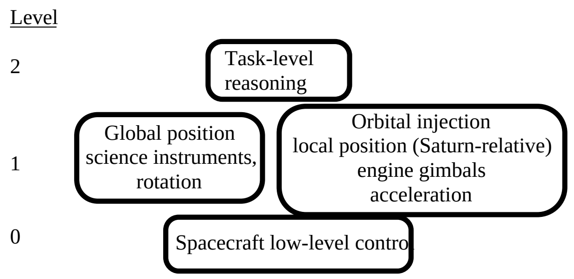
Level
2
1
0
Task-level reasoning
Global position science instruments, rotation
Orbital injection local position (Saturn-relative) engine gimbals acceleration
Spacecraft low-level control

Figure 4: A simple abstraction system for NASA’s Cassini spacecraft. At the ground problem space (Level 0), all spacecraft controls are relevant. However, at Level 1, two alternatives select a particular set of details depending on the spacecraft mode (science versus orbital injection).

We therefore define an *Abstraction System* as a partial order of abstractions:

*An **Abstraction System** is a set of partially ordered search spaces. The partial order must have a unique minimal element, which must bel the ground problem space.*

Figure 4 depicts a portion of the Abstraction System for a spacecraft robot based loosely on the Cassini spacecraft (NASA). Some representational detail as well as available actions may be ignored, depending on the goals and current conditions of the spacecraft.

At Level 1, we present two abstract spaces, both of which are abstractions of the ground space in Level 0. Note that no linear abstraction hierarchy can capture the separation between science experimentation and orbital entry that is captured by the abstraction system of Figure 4.

Creating a partial order on models is not new. For example, [6] uses partial orders of models in the context of model-based reasoning and diagnosis. Here, we apply similar techniques to defining an architecture for abstraction, which we shall then use to interleave planning and execution on a robot platform.

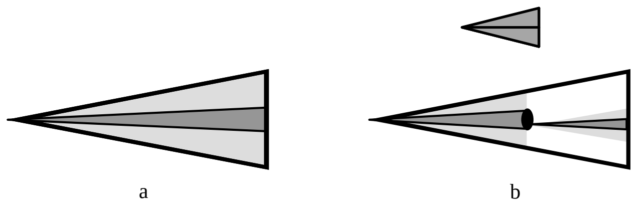

Figure 5: (a) A standard exponential search at the ground level. (b) Using an abstraction system, first an abstract space at Level 1 is searched, then its results are applied to the search in the ground problem space at Level 0. A subgoal resulting from the Level 1 search, denoted by the black oval, enables early search termination and plan execution.

Now that we have presented definitions for Abstraction and for the Abstraction System, we can begin to imagine the potential savings resulting from interleaving with this representation. Suppose that we have a control algorithm for interleaving planning in this problem space with execution.

Figure 5 depicts the potential computational savings of such an approach. Figure 5a depicts a standard search from the initial conditions to the goal at a ground problem space, making no use of abstraction. Figure 5b depicts the case of a two-level abstraction system. Planning is first conducted at Level 1, where a planner discovers a path to the goal. That path enables the introduction of a *subgoal* into the space of Level 0, where that subgoal enables early search termination.

Interleaving comes in at this point, for execution can be effected once a ground plan to the subgoal has been discovered. Once that aspect of execution is complete, planning resumes to the goal conditions and, finally, execution resumes to complete the process. The computational savings provided by abstraction can be denoted visually as the white space in the Level 0 problem space minus the grey area searched in the Level 1 problem space (the overhead of abstraction).

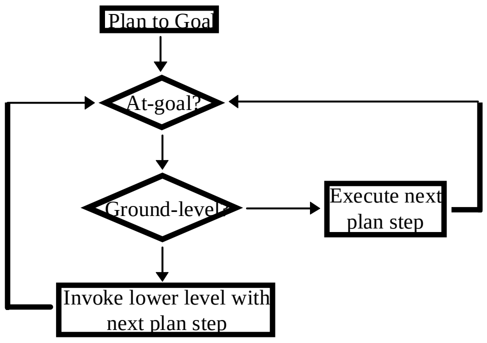

Figure 6: A flowchart description of the control algorithm used in every problem space in the abstraction system.

In order to implement this form of interleaving, planning and reasoning in each problem space needs to satisfy several task requirements:

1 *The planner must plan to the goal within its problem space.*
2 *The planner must be capable of translating and instituting a subgoal provided by any adjacent higher-level problem space.*
3 *The planner must construct multistep plans such that the results of a step can be used to supply an adjacent lower-level problem space with a subgoal.*

The flowchart of Figure 6 describes a planner algorithm that satisfies these requirements for any problem space in the Abstraction System. Note that the notion of the opportunistic refinement of higher-level subgoals applies to every level except the ground problem space; at this level, refinement is replaced by 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 possible lower level? In the systems we have implemented, higher level subgoals have been passed *in parallel* to all adjacent lower-level problem spaces. The worst-case cost of such an operation is proportional to the branching factor of the partial order, whereas the potential for savings is arbitrary, depending upon the existence of one such child problem space that is well-suited to handle that specific subgoal. We were fascinated to discover that, empirically, the overhead of launching the planning process in all child problem spaces appears to be negligible compared to this reward.

Of course, savings are only realized when the partial order is composed of well-designed problem spaces that, together, make the planning problem trivial. The fact that we employ partial orders means that any particular problem space can be valuable even if its representational simplifications are relevant only to one class of subgoals.

IMPLEMENTATION ON BALIN

We now revisit an essential question: can abstraction, in the final analysis, enable computational savings, or does the overhead of using multiple search spaces offset any potential savings?

We believe that abstraction is a win in complex and rich environments, where the autonomous agent is subject to a disparate set of possible inputs, state features, and various types of goals. To demonstrate the usefulness of abstraction in such a situation, we set out to design a problem domain for an indoor navigation robot that would satisfy these criteria.

Table I: Tasks and feature implemented by Balin

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 I summarizes the result of this design effort. Considerations 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 involves Professor happiness—its task is to sing regularly to a set of professors in order to keep them in good spirits. Balin's goal is to achieve overall Professor happiness while meeting its other requirements, including hourly chimes for the author and careful reasoning to either stay clear of "rainy" hallways or retrieve its umbrella.

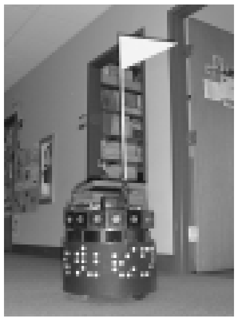

Figure 8: Balin of Smith

Figure 8 is a picture of Balin in its current 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 running Macintosh Common Lisp 2.01. This MCL image implements *every* problem space of the abstraction system (5 problem spaces in all), from the lowest-level *ground problem space* to the highest problem space that is three levels of abstraction removed.

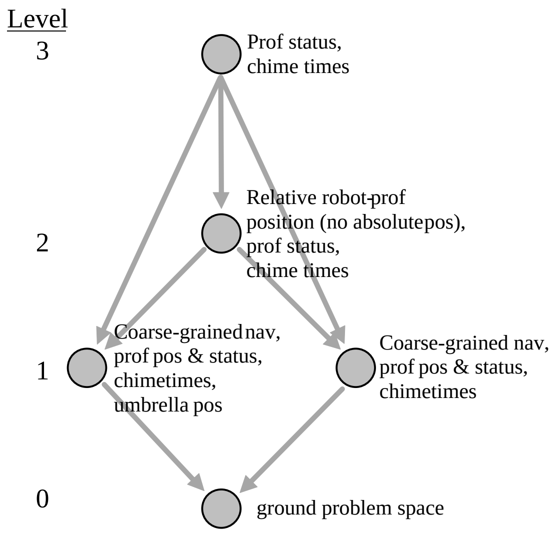
Level
3
2
1
0
Prof status, chime times
Relative robot-prof position (no absolutepos), prof status, chime times
Coarse-grained nav, prof pos & status, chimetimes, umbrella pos
Coarse-grained nav, prof pos & status, chimetimes
ground problem space

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 [7].

As one travels up the 4-deep abstraction system, the complexity of the search spaces decreases rapidly. At Level 1, two alternative search spaces distinguish between a context in which the agent reasons explicitly about the umbrella and one in which the agent ignores the umbrella. The savings in the latter case involve both action branching factor (since we ignore actions having to do with retrieving the umbrella) and with state space size (since we ignore the detail of umbrella position). Of course, the applicability of this latter problem space is more limited, since it will fail to find a solution plan if it is raining or if the agent does not know whether or not it is raining.

The juxtaposition at Level 1 between this "no-umbrella" world and the alternative "with-umbrella" world is truly that of alternative representations; one space is not to be viewed as hierarchically above the other since, for any given initial conditions and goals, one space will be more appropriate than the other.

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 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. 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.

THREE RESULTS

In this section, we present three types of results: empirical results on Balin demonstrate measurable performance improvement due to abstraction; analytical results demonstrate that abstraction saves at the exponent-level. Finally, we note the existence of formal results that enable architectural completeness and soundness to be proven for this model of abstraction.

Empirical Results

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

Table II: Planning time results for Balin 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 II 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 the case of "No abstraction", since it receives no subgoal information from a higher level anyway, even when an abstraction system is in place.

The table demonstrates that ground level planning without abstraction was impossible. Furthermore, planning without abstraction at Level 1 was possible, but just the planning time was actually greater than the entire planning plus execution time in the abstraction-based case.

A final comment about this table regards the relative sizes of planning times in each space. Note that the planning time in successively higher problem spaces decreases quickly. The planning time in these higher problem spaces 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.

Of course, one must remember that abstraction has the potential of leading to less optimal executable plans; it is entirely possible for abstraction to therefore make execution costs sufficiently suboptimal as to outweigh any computational savings brought on by the abstraction. In the case of Balin, the Abstraction System we have designed preserves execution optimality.

Indeed, as the independence and multifaceted nature of a problem domain increases, it becomes easier and easier to design such Abstraction Systems. The nonoptimality danger becomes apparent when seemingly independent problem properties and subgoals are, in fact, dependent.

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, k: $p^k a^k$.

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

Our definition of abstraction and the Abstraction System fits formally within the mathematical framework of state and property-based representations. This connection leads to the appealing characteristic of *provability*. Specifically, it is possible to guarantee *soundness* and *completeness* (convergence in the eyes of control theoreticians) for robot architectures that interleave planning and execution using a slightly more complex implementation of the abstraction system.

> 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, an extension of the flowchart shown in Figure 6.

This formal result has some similarity to formal results provided by other researchers in the field of abstraction. The key difference, however, is that we prove soundness and completeness of the interleaving system, not of a single planning episode.

RELATED WORK

The majority of research in the area of abstraction has focused on off-line abstraction [2,3,5]. Furthermore, much of this research has involved action-based abstraction. For example, [4] define both *inter-action* and *sequential* abstractions. The former involves abstract actions that represent a disjunction of a set of actions, whereas the latter enables abstract actions that are essentially macro-operators.

As stated earlier, this approach frequently demands that the researcher make simplifying assumptions concerning the action model for the sake of tractability. For instance, Helwig & Haddawy express actions models under a STRIPS assumption, even when describing the preconditions and postconditions of a macro-operator. This type of simplification is significant because, while it enables the abstraction of action, it severely limits applicable and tractable domains. Our work differs in that we do not institute a STRIPS model. Instead, we assume that action models are expressed as arbitrary mappings between knowledge states.

We are able to be this general because we focus upon state-based abstraction. Under the umbrella of state-based abstraction, the representational transformation that we call abstraction is really a way to reason about reachability. Other researchers have delved into state-based abstraction, most notably [3]. However, Knoblock asserts the *ordered monotonicity property*, which limits him to abstractions in which all refinements of an abstract plan leave the abstract literals established. Our approach differs from this in that we consider sound abstractions to be those in which *some* refinement exists. Of course, our weaker definition allows for much more possible abstraction spaces, with the potential penalty of run-time computation (e.g. planning) during the refinement process.

Furthermore, although Knoblock engages in state-based abstraction, he too employes STRIPS type action models, again restricting the set of applicable domains. A much more subtle representational argument involves our definition of state abstraction. Whereas Knoblock drops terms to form *reduced models*, we allow arbitrary clustering of ground level states. Our approach is strictly more general in that an abstraction operation can in fact depend on the value of multiple dependent real-world properties. Dropping literals in one way of eliminating detail; we argue that one may not want to drop a literal altogether, but to abstract across the values of two literals in the problem space.

Interestingly, several results from within the motion planning community can be couched as instances of our state-based abstraction formalism. Lazanas & Latombe [8] introduce the concept of *TC Actions*, which are actions that have a *durative* component, continuing on until an associated termination condition (TC) is detected. Lazanas & Latombe implement this method by introducing geometric subgoals in the robot's environment, then searching between these subgoals. In this more abstract search space, the action branching factor, the perceptual branching factor, the state set size and the solution length are all decreased.

Another motion planning research project, Donald [9] also demonstrates a form of state abstraction through the use of his *critical slice* mechanism, whereby he discretizes a continuous space while preserving soundness and completeness.

Significantly, note that these motion planning roboticists demonstrate instances of state-based abstraction and *do not* employ the STRIPS assumption. In the case of both Donald, for instance, the action model allows for arbitrary function specification. It is no coincidence that those researchers employing more complex robot models cannot use the STRIPS assumptions and, instead, shy away from such simplifications during abstraction.

CONCLUSIONS

We have presented a 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 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.

ACKNOWLEDGEMENTS

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

REFERENCES

1. Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115-135.
2. Holte, R., Mkadmi, T., Zimmer, R. and MacDonald, A. 1996. Speeding Up Problem Solving by Abstraction: A Graph Oriented Approach. *Artificial Intelligence*.
3. Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2).
4. Helwig, J. and Haddawy, P. 1996. An Abstraction-Based Approach to Interleaving Planning and Execution in Partially-Observable Domains. In *Working Notes of the 1996 AAAI Fall Symposium on Plan Execution: Problems and Issues*. November 1996. AAAI.
5. Bacchus and Yang. 1994. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*. 71: 43-100.
6. Nayak, P., Joskowicz, L, Addanki, S. 1992. Automated Model Selection using Context-Dependent Behaviors. In *Proceedings, Tenth National Conference on Artificial Intelligence* . AAAI Press.
7. Nourbakhsh, I. 1997. *Interleaving Planning and Execution for Autonomous Robots*. Kluwer Academic Publishers, Boston.
8. Lazanas, A. and Latombe, J.-C. 1995. Motion planning with uncertainty: a landmark approach. *Artificial Intelligence*, 76(1-2): 287-317.
9. Donald, B. 1989. Error Detection and Recovery in Robotics. *Lecture Notes in Computer Science*, 336. Springer-Verlag.