PLANNING WITH ABSTRACTION

Josh Tenenberg
Department of Computer Science
University of Rochester
Rochester, NY 14620
josh@rochester

Abstract

Intelligent problem solvers for complex domains must have the capability of reasoning abstractly about tasks that they are called upon to solve. The method of abstraction presented here allows one to reason analogically and hierarchically, making both the task of formalizing domain theories easier for the system designer, as well as allowing for increased computational efficiencies. It is believed that reasoning about concepts that share structure is essential to improving the performance of automated planning systems by allowing one to apply previous compuational effort expended in the solution of one problem to a broad range of new problems.

I. Introduction

Most artificial intelligence planning systems explore issues of search and world representation in toy domains. The blocks world is such a domain, with one of its salient and unfortunate characteristics being that all represented objects (blocks) are modeled as being perfectly uniform in physical features. We would like to model a richer domain, where objects bear varying degrees of similarity to one another. For instance, we might wish to model blocks and trunks, which are both stackable but of different sizes and weights, or boxes and bottles, which are both containers but of different shape and material. As a consequence of solving problems in this richer domain, we will want plans to solved problems to be applicable to new problems based upon the similarites of the objects to be manipulated. So, for instance, a plan for stacking one block on top of another will be applicable to a similar trunk stacking in terms of its gross features, but will differ at more detailed levels. We will present a representation for plans of varying degrees of abstraction based upon a hierarchical organization of both objects and actions that provides a qualitative similarity metric for problems posed to the planner. This plan representation has the following property. When a plan is expressed at a high level of abstraction, it will apply to a wide class of problems but with little search information, while, when expressed at lower levels of abstraction it will apply to fewer problems but give increasing amounts of information with which to guide search.

II. Object and Action Abstraction

A common way to represent physical objects is within a taxonomic hierarchy [Hendrix 1979]. A graphic example of part of one such hierarchy is given in figure 1. An arc from node *v* to node *w* indicates a subset relation between these types. Therefore, all objects of type *v* are also objects of type *w*, and inherit all properties provable of type *w*. We will call *w* an abstraction of *v*, and *v* a specialization of *w*. These taxonomies enable us to make assertions about a class of objects that we need not repeat for all of its subclasses. So, for instance, if it is asserted that all supportable objects can be stacked, then it need not be asserted separately that blocks can be stacked, boxes can be stacked, and trays can be stacked. It suffices to assert that blocks, boxes and trays are all supportable objects. This structure is not strictly a tree, which means that each object can be abstracted along several different dimensions, with the effect that every node inherits all of the properties of every other node from which there is a path. For example, a Bottle is both a Container and a Holdable object , since there are paths in the graph from Bottle to both Holdable and Container. Note that this structure admits no exceptions. We prefer instead to weaken those assertions we can make of a class in order to preserve consistency.

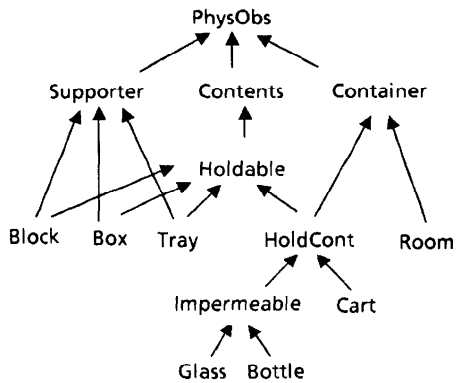


figure 1

We would like to represent actions similarly. Typically [McCarthy and Hayes 1969], actions are represented in terms of those conditions that suffice to hold before the performance of the action (called *preconditions*) that ensure that the desired *effects* will hold after the performance of the action. However, an inherent inefficiency with this is that many actions share preconditions and effects which must be specified separately for each action, providing no means with which to determine which actions are similar and hence replaceable by one another in analogous problems.

What we will do alternatively is to provide an action taxonomy, by grouping actions into inheritance classes. An example of a partial action hierarchy is given in figure 2. The boxed nodes denote actions and the dotted arcs between actions denote inheritance. As with the object hierarchy, if there is an

This work was supported in part by the National Science Foundation, grants DCR-8405720 and IST-8504726.

inheritance arc from action *v* to action *w*, we say that *v* is a specialization of *w*, and *w* is an abstraction of *v*. The solid arcs from a literal into an action denote necessary preconditions for that action, and the solid arcs from an action to a literal denote effects of that action. Each action inherits all preconditions and effects from every one of its abstractions. So, for instance, *CarriedAloft(x)* is a precondition of *placeIn(x,y)* inherited from *put(x,y)*, and *In(x,y)* is an effect of *placeIn(x,y)* inherited from *contain(x,y)*. As we proceed down this graph from the root node traversing inheritance nodes *backward*, by collecting the preconditions for each action encountered, we are adding increasing constraints on the context in which the action may be performed in order to have the desired effects. At the source nodes, which represent the primitive actions, the union of all of the preconditions on every outgoing path constitute a sufficient set of preconditions. An action can only be applied if its sufficient set of preconditions are all satisfied in the current state. The sufficient set of preconditions for *placeInBox* has been italicized, and is exactly the union of those preconditions for each action type on all paths from *placeInBox* to *contain*. Additional action hierarchies we might have are *remove* with specializations *pourOut* and *liftOut*, and the hierarchy *open*, with specializations *openDoor* and *removeLid*.

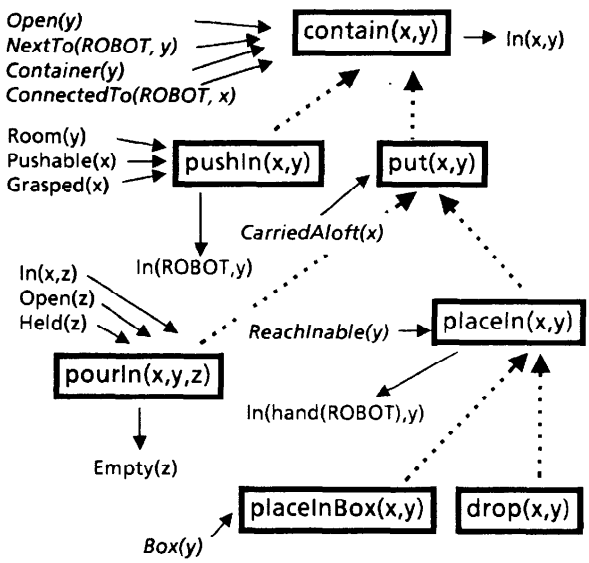


figure 2

III. Plan Abstractions

Planning involves finding a temporally ordered sequence of primitive actions which when applied with respect to the temporal ordering from a given initial state produces a state of the world in which the desired goals hold, and for which the sufficient preconditions for each primitive action must be satisfied by the state in which the action is performed. In this paper, a *total* temporal ordering of actions will be used for simplicity, although the ideas presented here can be extended to more general temporal orderings (partial orderings [Sacerdoti 77], concurrent actions [Allen 84]). Such a totally ordered sequence of actions will be called a *primitive plan*, or simply a *plan*. Finding plans to solve given problems involves searching for a state of the world which satisfies our goals from those states of the world which are possible from the initial state through the performance of one or more actions. To reduce this search, we will make use of plans that have already been found for solving previous problems. In order to use saved plans, the similarity between the previous problem solved by this plan and the current problem we wish to solve must be evaluated. We describe *plan graphs* which are a means for performing this evaluation. These plan graphs are generalizations of triangle tables [Fikes, Hart, and Nilsson 1972].

Using *explanation based generalization* [Mitchell, Keller, and Kedar-Cabelli 1985] techniques, from a primitive plan a plan graph is constructed which embeds the causal structure of the primitive plan such that the purpose of each plan step can be determined. Each action is represented not only as a primitive, but as a path from a primitive to an abstract action taken from an action hierarchy, where the causal structure enables us to determine which hierarchy to choose. Given a new problem, this plan graph is searched for its most specific subgraph whose causal structure is consistent with the new problem. This subgraph represents an abstract plan for the new problem, and will be used as a guide for finding the primitive plan for this problem. If an action in the original plan cannot be applied due to some difference between the old and the new problem, such as a difference in corresponding objects manipulated, (e.g., a ball in one case, a box in the other) we can replace this action by choosing another which is a different specialization of the same abstraction (*pickupBall* replacing *pickupBox*, both of which are specializations of *pickup*). Thus many problems can be solved by performing search within the constraints of the abstract plan we have retrieved for this problem, rather than having to perform an unconstrained global search.

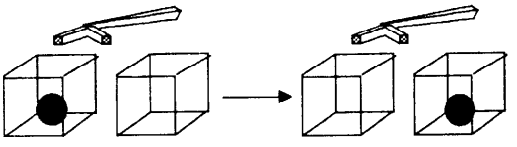
A plan graph of a plan will have nodes for each action in the primitive plan, and nodes with directed arcs for each precondition and effect of these actions. If an effect of an action satisfies a precondition of another, this will appear as an arc from the first action, to its effect, to the second action. These *causal chains* establish the purpose of each action in terms of the overall goal of the plan. We will formally define plan graphs in two stages. The first stage includes only the causal structure, while the second incorporates abstractions.

A plan graph $G = (V,E)$ for primitive plan P is a directed acyclic graph where V and E are defined as follows. The set of vertices is partitioned into two subsets V_p and V_a, precondition nodes and action nodes. Likewise, E is partitioned into two two subsets E_c and E_s, causal edges and specialization edges. For every action in P, there is a node in V_a labeled by its corresponding action. If *p* is an effect of action *a* in P, then there is a corresponding node in V_p labeled *p*, and the edge *(a,p)* is in E_c, and for every action *b* in P that this instance of *p* satisfies there is an edge *(p,b)* in E_c. For example, if action A1 establishes condition K which is a precondition of action A2, then (K,A2) is in E_c if and only if there does not exist action A3 that occurs after A1 but before A2 that *clobbers* K (establishes $\neg$K). Clearly any precondition of each action that is not satisfied by a previous action must be satisfied by the initial state. For every such precondition *p* there is a corresponding node in V_p labeled *p*, and for every action *a* in P for which this instance of *p* is a precondition, there is an edge *(p,a)* in E_c. Each action node in E_a is additionally labeled by a number indicating its temporal order, the n^{th} action labeled by *n*.

This graph will be specified further by the addition of action abstractions, but note that as it stands it is similar to a graph

version of triangle tables [Fikes, Hart, and Nilsson 1972], and fulfills much the same function. We can use the same technique as used in triangle tables for generalizing a plan by replacing all constants in the action and precondition nodes by variables, and redoing the precondition proofs to add constraints on variables in different actions of the plan that should be bound to the same object (see previous reference for details). These constraints will have to be added to the graph as additional preconditions, but are left off in our examples for clarity. The preconditions for this plan graph are the set of source nodes (nodes with no incoming arcs), and the goals of this plan graph are the set of sink nodes (nodes with no outgoing arcs). This graph has the property that any subset of its goals can be achieved from any initial situation in which we can instantiate all of the preconditions by applying each of the actions in order. A plan graph for the problem in figure 3 of moving a ball from one box to another is given in figure 4 (nodes representing preconditions satisfied by the initial state rather than by a previous action are not included in this figure). This graph will be altered to include abstract actions in a straightforward fashion.

figure 3



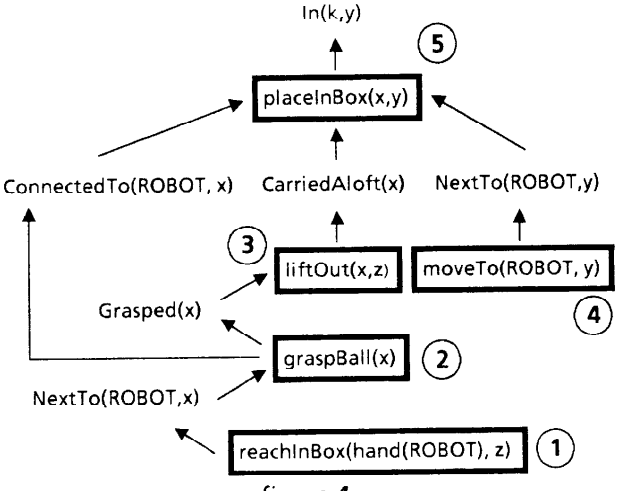


figure 4

Figure 5 is an example of the altered plan from figure 4 (the outlined subgraph of figure 5 will be explained later). This alteration is done as follows. For each primitive action A_0 in V_a, we will add nodes to V_a labeled A_1, A_2, ..., A_n (where *n* may be different for each primitive action) and edges to E_s labeled (A_0, A_1), (A_1, A_2), ..., (A_{n-1}, A_n), where there exists some action hierarchy such that A_i is an abstraction of each A_k for $k<i$, and A_n satisfies at least one effect *p* for which there exists a node in V_p labeled *p* and an edge in E_c labeled (A_0, p). More simply, we add an abstraction *path* from an action hierarchy to the plan graph. We then redirect each precondition arc (p, A_0) to point to the highest abstraction A_i for which there is an arc (p, A_i) in the chosen action hierarchy. In other words, *p* is a precondition of abstraction A_i, but not of any abstraction of A_i. For instance, ***placeInBox*** is replaced by the abstraction sequence ***placeInBox, placeIn, put, contain,*** and the preconditions ***NextTo*** and ***ConnectedTo*** are redirected to ***contain***, while ***CarriedAloft*** is redirected to ***put***. We additionally redirect effect arcs (A_0, p) such that the effects come from the highest abstraction A_i for which there is an arc (A_i, p) in the chosen action hierarchy. In other words, *p* is an effect of abstraction A_i, but not of any abstraction of A_i. Turning again to figure 5, the effect arc into ***ConnectedTo*** is redirected to come from ***attachToAgent,*** since this will be an effect of every specialization of this abstraction, and the effect arc to ***Grasped*** is redirected to come from ***grasp***. We will additionally add temporal numberings to each abstraction on a path from each primitive action (although the examples will only number the highest abstractions for each action).

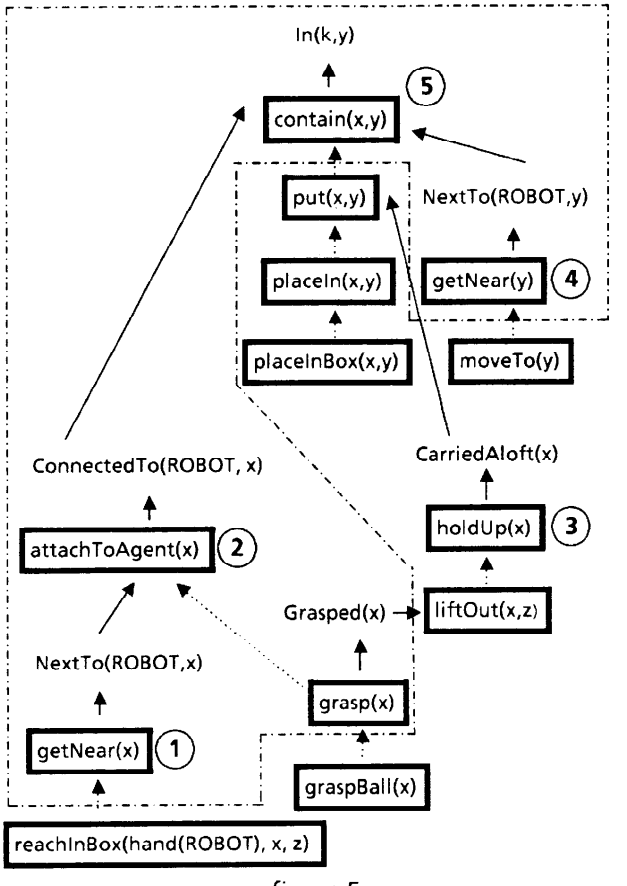


figure 5

The primitive action nodes of this plan graph indicate the primitive plan that solves the problem for which the plan was constructed. The distance between an action node and one of the goals of the entire plan graph along its shortest causal chain is a rough measure of the significance of the action to the overall plan. The shorter the distance, the more likely this action or an abstraction of it will be required in a similar problem; the greater the distance, the less likely this action will be useful in a similar problem. This plan can thus be abstracted by one or both of the following: removing causal chains from one or more precondition nodes, and removing specialization paths from one or more action nodes. Each resultant ***partial plan graph*** represents a plan with some of the detail unspecified.

More formally, a partial plan graph P of plan graph P' is any subset of the nodes and arcs of P' such that no source nodes are action nodes, at least one sink node (goal) of P' is in P, and these will be the only sink nodes in P, and for every node in P, there exists at least one path from this node to a sink node (unless that node is itself a sink node). Additionally, if *b* is an action node, then every node *p* for which there exists an arc (*p,b*) in P' will be added to P along with this arc. We will additionally "mark" each source node in P that was also a source node in P'. This mark indicates that this precondition is satisfied by the initial state of the original problem, as opposed to being satisfied by the performance of a previous action. The reason for marking these nodes will be explained later. From this definition, there will be several partial plan graphs that can be constructed from a given plan graph. The subgraph outlined by the dotted line in figure 5 is one exapmle. As before, the preconditions of a partial plan graph are the formulas attached to the source nodes (not included in the given figures), while the goals of each partial plan graph are the formulas attached to the sink nodes.

Figure 7 is the plan graph for a plan to solve the problem from figure 6. Here a box must be moved between rooms. In both this problem and that of figure 3, the goal is to move an object from one container to another. This draws analogies between rooms and boxes, which are both containers according to our object hierarchy, and between placing objects in boxes and pushing objects into rooms, which are both containment actions, according to our action hierarchy. At an abstract level, the plan of attaching the object to the agent, and moving the agent from one container to the other suffices for both problems, and in fact this is the abstract plan represented by the identical partial plan graph that is outlined by the dotted line in both figures 5 and 7. So although the problems that these graphs solve are different, at this level of abstraction they are identical.

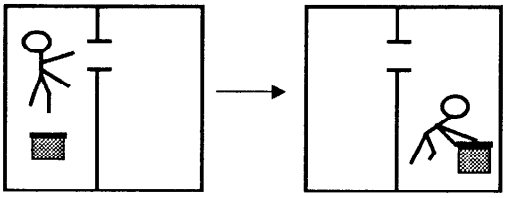


figure 6

We can generalize from this in that for any partial plan graph P of plan graph P', there will exist a set Π of plan graphs for which P will be a partial plan graph of each of them. That is, P will describe each primitive plan of each element from this set at some level of abstraction. We will use the symbol Π_P to denote the largest such set. For instance, if we label the outlined partial plan graph of figure 5 *K*, then the graphs of figures 5 and 7 are in Π_K. We will say that the primitive plan of each member of Π_P is an *expansion* of the partial plan graph P. The more general P is, that is, the smaller a subgraph of P' it is and hence the more abstract each of its constituent actions are and the smaller its causal chains, the larger will be the cardinality of Π_P. We will say that partial plan graph P *solves* problem Q if and only if there exists an element of Π_P whose primitive plan solves Q for some instantiation of all of its variables by ground terms.

Given a partial plan graph P and a problem instance Q that P solves, we can find an expansion of P that solves Q by only searching for specializations of the abstract actions of P without

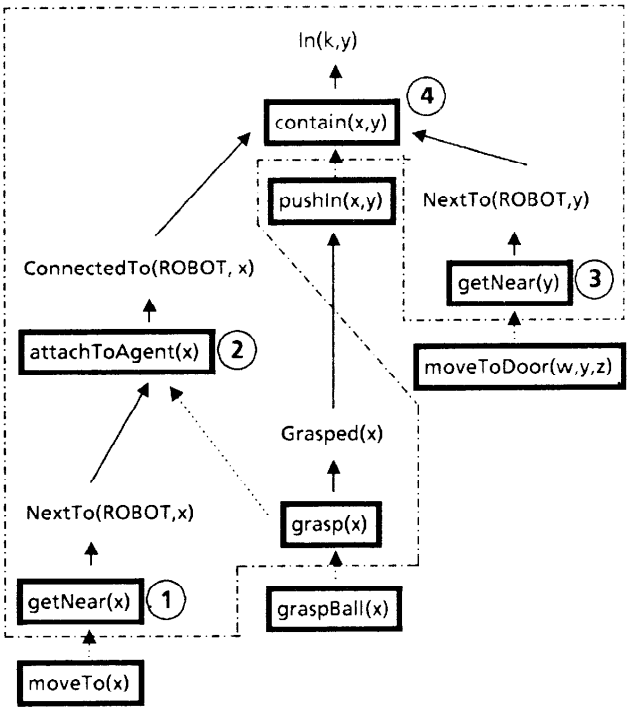


figure 7

having to backtrack through the actions of P itself. P thus serves as an abstract guide to solving Q. So, for instance, given the partial plan graph outlined in figure 5, we can find the remainder of the primitive plan (those actions not inside the dotted line) that solves the problem from figure 6 by only having to do local search. By this, we mean that for any non-primitive action in this partial plan graph, (such as *contain*, in figure 7), we follow arcs backward through that abstract action's specialization tree (figure 2 in this case) until we find a primitive action whose preconditions are all satisfied by the state in which it is executed (*pushIn*, in this example). If no such primitive exists, then additional primitives must be inserted in this plan to establish the sufficient preconditions for *some* specialization, where these inserted actions do not clobber preconditions of any of the already established succeeding actions.

Unfortunately, we cannot in general know if a given partial plan solves a given problem instance unless we perform the possibly unbounded local search for the primitive plan that verifies this. It may not be possible to find specializations of each extant abstract action without reordering some of the actions, and therefore backtracking through and altering the partial plan graph itself. Although we are not guaranteed certainty, we can still use the plan graphs as a heuristic for search. We will define a partial plan graph P as being *applicable* to a problem Q if and only if the goals of Q are a subset of the goals of P, and the marked preconditions of P are a subset of the conditions that hold in the initial state of Q. Recall that we marked all of those preconditions in a partial plan graph that were satisfied by the initial state of the original problem. Applicability thus means that the current initial state satisfies the same preconditions *at this level of abstraction* as the original initial state.

Suppose we wish to find a primitive plan for problem Q consisting of an initial state and a set of goals (for simplicity we

will assume that this goal is a single literal). Additionally suppose that the goals of plan graph P are the same as those of Q. We will attempt to find the most specialized partial plan graph P' of P for which an expansion exists that will solve Q, even though it is possible that no such P' exists. We will do this by traversing P *backward* from its goal node through the causal and specialization arcs, considering increasingly larger partial plans of P. We will continue this traversal as long as the partial plan represented by all of the paths pursued is still applicable to Q, stopping when we can no longer traverse any arc and still have applicability of the current partial plan to problem Q. The size of the partial plan that we have constructed is thus a qualitative measure of similarity between the original problem and the current one. If there are only insignificant differences, the partial plan may be equivalent to the entire plan graph. If the differences between the problems are large, this may result in a graph of only a few actions expressed at high levels of abstraction. But given the exponential nature of searching through combinatorial spaces, knowing the temporal ordering of even a few of the action abstractions that will eventually appear specialized in our plan may help significantly.

IV. Previous Research

Abstraction in planning is typically viewed in terms of *decompositional abstraction* as used in NOAH-like planners [Sacerdoti 1977]. In these planners, action A is an abstraction of actions B,C,D if the latter actions are each steps in the performance of action A. This type of abstraction is thus orthogonal to inheritance abstraction presented here.

ABSTRIPS [Sacerdoti 1974], although using different techniques, shares some important similarities. ABSTRIPS is an iterative planner, where increasingly large subsets of preconditions of each action are considered at each successive iteration. The developed plan at each level is then used to guide search at more detailed levels, where the satisfaction of emergent preconditions is attempted locally, similar to what is done in this paper.

Of even greater similarity, but within a different domain, is the work presented in [Plaisted 1981], who uses abstraction within a theorem prover. He details how a desired proof over a set of clauses can be obtained by first mapping the clause set to a set of abstract clauses, obtaining a proof in this (hopefully simpler) space, and then using this proof as a guide in finding the proof in the original, detailed space. His mapping process and abstract proof are similar to our search for an abstract plan within our saved plan space - but rather than constructing an abstract plan for each new problem, we attempt to appropriate one from a previously solved problem.

V. Conclusion

The primary motivation for using abstraction was so that search for solutions to new problems can be improved by using solutions to old problems. We believe that this approach can be used to these ends in a domain in which objects are distinguishable at various levels of detail. We will try matching abstract plans to problems that have the same goals. Any such new problem whose initial state does not contain *all* of the preconditions of the original initial state will thus not match the abstract plan at every level, but will likely do so at some level. The partial plan graph still provides two important functions in this case. First, it ignores "unimportant" preconditions at the most general levels, where the importance of a precondition is determined by the height at which it appears in the action hierarchy. Second, the search space of the new problem can be explored along those paths that do not match the original problem, while attempting to leave intact those paths that do match.

We must point out that the abstraction described in this paper has not been implemented for even a small domain. In fact, one of the obstacles to doing such an implementation is that one may likely only see benefits in a large domain. Thus, there will be little point to use this method as a representation for the vanilla blocks world. An additional issue is in the choice of problems that the system will encounter. One can always construct problem sequences given as input to the problem solving system such that the abstractions in the model will optimize performance. By the same token, one can always construct problem sequences where the abstractions will give quite poor performance. The ultimate test of a set of abstractions will therefore be empirical in that they must be cost-effective (in terms of some resource measure) only as compared with other problem solvers (human or machine) for a given domain. We can make no such claims for the *particular* abstractions of the limited physical world domain illustrated in this paper. The importance of this work is in how we can structure knowledge for solving problems in domains that are far richer than the ones in which the current generation of planners have approached. It is believed that inheritance abstraction will be a powerful technique in this endeavor.

Special thanks to my advisor, Dana Ballard, whose energy, knowledge, piercing insights and trust have made it all worthwhile, to Leo Hartman, who always seems to have an answer when an answer is needed, and to Jay Weber, who will hopefully solve the questions of how we go about *constructing* abstraction hierarchies.

References

[Allen 84]
Allen, J.F., "Towards a General Theory of Action and Time", Artificial Intelligence **23**:123 - 154, 1984.

[Fikes, Hart, and Nilsson 1972]
Fikes, R., Hart, P. , and Nilsson, N., "Learning and executing generalized robot plans", *Artificial Intelligence* **3**:251 - 288, 1972

[Hendrix 1979]
Hendrix, G.C., "Encoding Knowledge in Partitioned Networks" in, Associative Networks, ed. Findler, N.V. 1979

[McCarthy and Hayes 1969]
McCarthy, J., and Hayes, P., "Some philosophical problems forom the standpoint of artificial intelligence", In B.Meltzer and D. Michie (editors), *Machine Intelligence 4,* 1969.

[Mitchell, Keller and Kedar-Cabelli 1985]
Mitchell, T., Keller, R. and Kedar-Cabelli, S., "Explanation Based Generalization: A Unifying View", Rutgers Computer Science Dept. ML-TR-2, 1985.

[Plaisted 1981]
Plaisted, D., "Theorem Proving with Abstraction", *Artificial Intelligence* **16**:47-108, 1981

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

[Sacerdoti 1977]
Sacerdoti, E. A structure for plans and behavior. American Elsevier Publishing Company, New York, 1977