A Hierarchical Planner that Generates its Own Hierarchies

Jens Christensen*
Computer Science Department
Stanford University
Stanford, CA 94305
jens@cs.stanford.edu

Abstract

PABLO[1] is a nonlinear planner that reasons hierarchically by generating abstract predicates. PABLO's abstract search spaces are generated automatically using *predicate relaxation*, a new technique for defining hierarchies of abstract predicates. For some domains, this mechanism generates hierarchies that are more useful than those created by previous techniques. Using abstractions can lead to substantial savings in computation time. Furthermore, PABLO can achieve a limited form of reactivity when reasoning with relaxed predicates. These abstractions can be viewed as small reactive plans, and our method as an approach to dynamically combining these into useful nonlinear plans.

Introduction

It has long been known that abstractions can greatly reduce search complexity [Sacerdoti, 1974, Korf, 1987]. Wilkins [Wilkins, 1988] has noted that there has been much confusion in the planning literature about what constitutes an abstraction level. Abstraction usually suggests ignoring details and concentrating on important aspects of a problem.

Most planners, beginning with NOAH [Sacerdoti, 1977, Wilkins, 1988], employ a technique we will label *operator* abstraction. The idea is that more abstract operators are defined in terms of less abstract ones. In a planner that allows operator abstraction, each abstraction level consists of a set of operators (possibly decomposable) and remaining subgoals. A level consisting only of primitive operators with no outstanding subgoals is a complete, executable plan.

In NOAH, these levels were explicitly stored in a procedural net. However, in NOAH and other planners that employ procedural nets, these levels may or may not form a proper abstraction hierarchy. For the levels in a procedural net to be meaningful abstract levels, it is necessary that the operators in the system truly be defined in terms of other, less abstract, operators. Furthermore, operator abstraction in these planners is not automatic, but rather is left to the encoder of the domain, who must decide on how the operators should be abstracted.

There is another abstraction technique encountered in the planning literature wherein abstraction is based not on the operators but rather on the states of the domain. We will label this type of abstraction *state abstraction.*

State abstraction was introduced to planning by ABSTRIPS [Sacerdoti, 1974]. Although ABSTRIPS proved state abstraction to be an effective abstraction mechanism, few planners employ this technique. In this paper, we introduce a new technique for automatically performing state abstraction - *predicate relaxation.* The technique has been implemented in PABLO, an abstract nonlinear planner.

Predicate Relaxation

Predicate relaxation is a method for weakening a predicate so that it holds in more states than it otherwise would. Specifically, we would like to consider a predicate P true, not only in those world states in which it is, in fact, true, but also in those states from which P can be made true by one or more operator applications.

In a domain with m operators, given a predicate P, we define P^n_{rel} as follows:

$$P^0_{rel} = P$$

$$P^n_{rel} = P^{n-1}_{rel} \bigvee_i^m Reg(Op_i, P^{n-1}_{rel})$$

where $Reg(Op_i, P)$ is the regression of predicate P through operator Op_i.

Obviously, $P^{(n+1)}_{rel}$ holds in a superset of the world states where P^n_{rel} holds. Furthermore, if P^n_{rel} holds in state s, there exists a plan which can achieve P in n steps or less, from state s.

*This work was partially supported by NASA Grant NCC2-494 and by Texas Instruments Contract No. 7554900.

[1]PABLO stands for Predicate ABstraction LOgic, and is also the first name of a certain, famous, *abstract* painter.

If P^n_{rel} holds, but Q^n_{rel} does not, one can say that P^n_{rel} is more of a "detail" than Q^n_{rel}, since P can be achieved more easily that Q. Predicate relaxation provides a gradual widening of the states in which a predicate holds. In ABSTRIPS, a predicate can either hold in those states in which it was intended to hold, or, when its criticality value is less than the current threshold, hold in all states of the domain. This change in the semantics of a predicate can be quite sharp. Predicates abstracted with predicate relaxation, however, avoid this semantic cliff, since the set of states in which they hold is gradually enlarged at each relaxation level.

PABLO

PABLO is a nonlinear planner that operates in a hierarchy of abstraction spaces. PABLO employs STRIPS-style operators [Fikes and Nilsson, 1971]. A plan consists of a strict partial order of instantiated operators. During planning, PABLO makes use of a modified version of TWEAK's modal truth criterion [Chapman, 1987], extended to handle abstract predicates.

In an initial relaxation phase, PABLO creates the relaxed definitions for the predicates appearing in the goals and in the preconditions of operators. This need only be done once for each domain.

The general idea during planning is that PABLO first consider the most important predicates, and then consider successively less important predicates. This is accomplished by associating each planning level with a relaxation level, and planning with relaxed predicates of that level; when planning at level k, every predicate P in the preconditions of operators and goals is replaced with P^k_{rel}.

When moving down abstraction levels, if newly created subgoals appear in different sections of the plan, PABLO attempts to achieve them independently. The rationale for this being that these predicates were considered "details" at the higher level and presumably do not have global consequences. In cases where this assumption fails, the consequences can, of course, be costly, in terms of computation time. However, in our experience the increased efficiency outweighs this risk.

Example

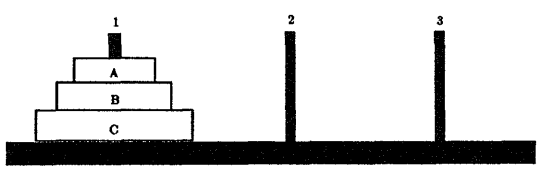

Figure 1: Towers of Hanoi

A well known problem with many inherent abstractions is the Towers of Hanoi problem. The operator given to PABLO is the following:

```
Move(x,y,z)
  P:{Smaller(x,z),Movable(x),On(x,y),
     Clear(x),Clear(z)}
  D:{On(x,y),Clear(z)}
  A:{On(x,z),Clear(y)}
```

See figure 2 for a trace of PABLO solving the 3 disk Towers of Hanoi problem. The plan at the highest level of abstraction consists of Move(C,x,P3), where x is a variable. At this level, all its preconditions are satisfied ($\text{Clear}^2_{\text{rel}}$(C) is satisfied since it can be achieved in two steps).

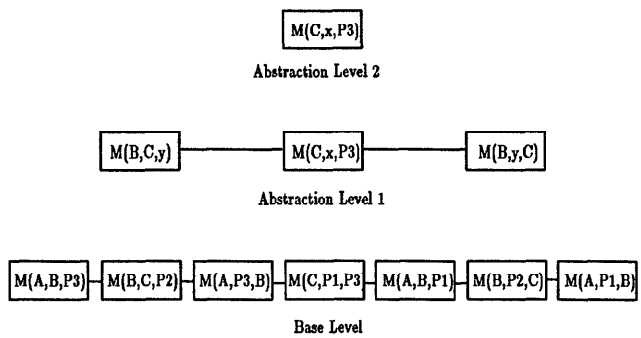

Figure 2: Towers of Hanoi Trace

When we move down to the next abstraction level $\text{Clear}^2_{\text{rel}}$(C) becomes $\text{Clear}^1_{\text{rel}}$(C) which is not satisfied in our initial state, since we cannot clear C in one step. PABLO therefore plans to achieve $\text{Clear}^1_{\text{rel}}$(C) by adding the action Move(B,C,y) to the plan. In doing so, it undoes On^1_{rel}(B, C), which PABLO then plans to reachieve, using the action Move(B,y,C). At this point, the plan at the first level of abstraction is complete since all the first level relaxations of the goals and preconditions are satisfied. Planning is then completed at the base level using the original predicates of the domain.

In this case, PABLO has discovered and made use of the inherent abstractions in the domain. Empirical evidence, presented in [Christensen, 1990] shows a reduction of planning time from exponential in the number of operators, to linear. This conforms to the theoretical analyses in [Korf, 1987, Knoblock, 1990], on the potential computational advantages of using abstraction in planning.

Comparison with ABSTRIPS ABSTRIPS would assign the following criticality values to the predicates in the domain:

```
Move(x,y,z)
      P:{[3]Smaller(x,z),[3]Movable(x),[2]On(x,y),
         [2]Clear(x),[2]Clear(z)}
      D:{On(x,y),Clear(z)}
      A:{On(x,z),Clear(y)}
```

ABSTRIPS creates only one level of abstraction in this domain. When solving the problem, after finishing the abstract level, the plan consists of one action Move(C,x,P3). Although this is of some aid in developing the plan at the base level, it is not as useful as PABLO's hierarchy.

ABSTRIPS's hierarchies are domain-dependent but problem-independent. The number of different criticality values, and therefore the number of abstraction levels, of ABSTRIPS is constrained by the number of different predicates of the domain. For the 4 disk Towers of Hanoi, ABSTRIPS still has only one abstraction level, whereas PABLO generates 3 abstraction levels. In general, for the n disk Towers of Hanoi problem, PABLO generates $n-1$ abstraction levels, whereas ABSTRIPS still creates only one abstraction level.

For other recent approaches to abstraction see [Benjamin, 1990].

Limited Reactivity

There is currently great interest in the designing of systems which provide timely responses in time-stressed situations [Nilsson, 1990, Schoppers, 1987]. Many researches have distanced themselves from traditional planning methods on the grounds that these are often slow and impractical. For example, suppose a planner is given the problem in figure 3 to solve.

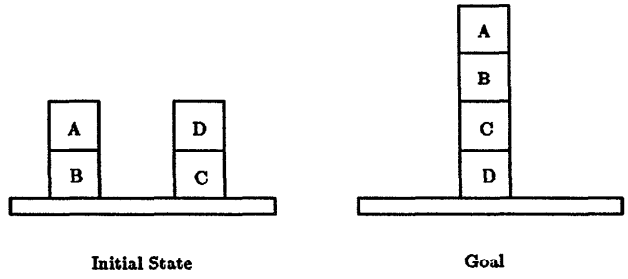

Figure 3: Planning Problem

We are given the following two operators:

PUTON(x,y)	TABLEOPR(x)
P:{Clear(x),Clear(y), On(x,z)}	P:{Clear(x),On(x,y)}
D:{Clear(y),On(x,z)}	D:{On(x,y)}
A:{Clear(z),On(x,y)}	A:{Clear(y),On(x,TABLE)}

Using the classic nonlinear planning method a trace of the plan at various stages of development might look as in figure 4.

One notable feature of this trace is that until the final plan is produced, the classical planner is not aware of any executable actions to perform in the initial state. The actions Puton(B,C) and Puton(C,D) are not directly executable in our initial state. Should the planner be interrupted at any time during planning it would not have a reasonable action to perform. This is

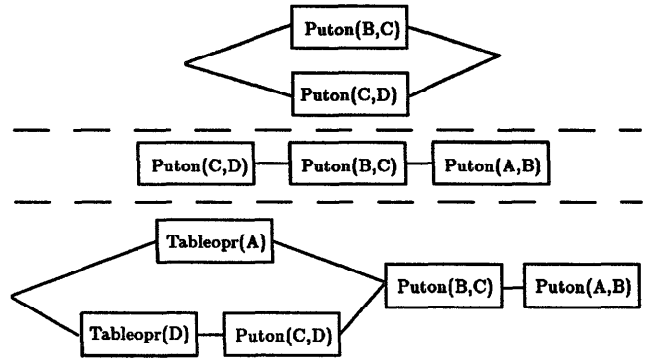

Figure 4: Classic Planning Trace

one reason traditional planning methods have generally been regarded as unsuitable for real-time tasks.

The new proposed systems have generally involved compiling large sections of the search space into efficient caches which compute appropriate responses to situations that might be encountered. We will take the liberty of referring to the types of structures produced by these approaches as *reactive plans*. As has been pointed out by various researchers, most notably Ginsberg [Ginsberg, 1989], reactive plans have several drawbacks. Principally, a reactive plan grows exponentially with the size of a domain, which can make them unwieldy for practical purposes.

Besides these reactive planning methods, several other alternatives to classical planning have been suggested. Some favor abandoning backward chaining plan-space search in favor of a forward search of the state space [Washington, 1989]. The advantage of this approach is that an executable action is available as soon as an action has been found to be applicable in the initial state. Unfortunately, until we encounter the final solution during the forward search, we have no guarantee that our current sequence of actions will eventually lead to the goal. Further, we cannot take advantage of the least-commitment implicit in the nonlinear representation of plans. Finally, a forward search, so as not to be completely blind, needs a domain-specific heuristic, thereby reducing domain-independence.

An extension of the above is proposed by Drummond [Drummond, 1989], where *situated control rules* are used to advise an executor on appropriate actions to take. This is a promising idea, but because it too is not goal-directed, its usefulness might be somewhat limited.

Another approach one can take, within the classical planning context, is that of planning down a *left-recursive wedge* of the partial plan in case of an interruption. The idea is to repeatedly expand the left-most outstanding preconditions until an action is encountered with all its preconditions satisfied. In some

circumstances, this approach might be successful. Unfortunately, the time to plan in this manner is possibly unbounded, since interactions might be encountered, necessitating backtracking.

The method we propose retains the power of partially ordered plan representations, but also allows the planner to identify plausible executable actions early in the planning process.

Reactive Reasoning with PABLO

The problem we are addressing is that of providing a plausible executable action should PABLO be interrupted before it has formed a complete plan. Ideally, we would like to provide as long a sequence of executable actions as possible.

Towards this end, we can store, along with each relaxed predicate, the operators through which the predicate was regressed during the relaxation process. Then, during planning, when a relaxed predicate is determined to hold in a situation, the operator through which the predicate was last regressed is automatically identified. For example, this is the first level relaxation of On(x,y).

$On^1_{rel}(x, y)$	
On(x,y)	[]
(y = TABLE) ∧ Clear(x) ∧ ∃ z On(x,z)	[Tableopr(x)]
(y ≠ TABLE) ∧ Clear(x) ∧ Clear(y) ∧ ∃ z On(x,z)	[Puton(x,y)]

The logical expressions are conditions under which the predicate should be determined to hold. The operators through which the predicate was regressed to arrive at the expression are shown in the right side of the table. In this case, since it is a first level relaxation, only one operator is included.

During planning, the relaxation table is examined from top to bottom. When an expression is found that is satisfied in the current state, the relaxed predicate is said to be asserted in that state. We also say the relaxed predicate is *grounded* in this state.

Once PABLO has completed a plan at one level of abstraction, and is working at the next lower level, it can utilize the extra information stored along with the relaxed predicates that hold at the higher level, should it be interrupted. PABLO chooses a plausible action by examining the preconditions of the earliest action(s) of the plan. If one of these actions has all its preconditions satisfied at the base level, the action is obviously executable.

If no such action exists, PABLO can choose from among the leftmost operators associated with the satisfied predicate relaxations that are grounded in the initial situation. All relaxed predicates must be satisfied since the plan was completed at the higher level. Any of the actions collected in this manner are executable.

See figure 5 for a trace of PABLO solving the previous example. It should be noted that PABLO solves the example in 6.6 seconds with the relaxation hierarchy versus 68.5 seconds without it.

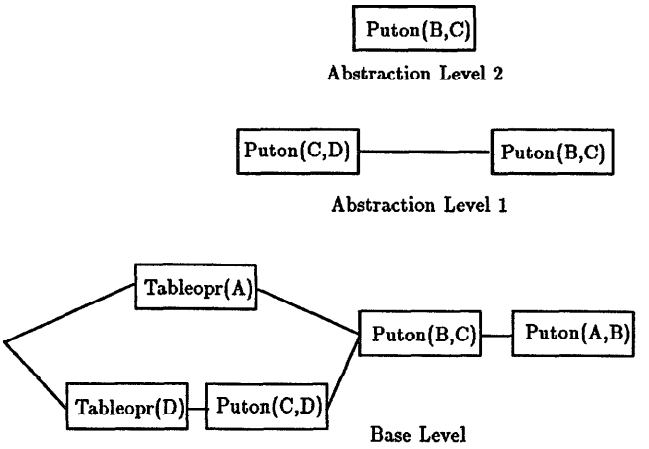

Figure 5: Pablo's Planning Trace

After completing planning at the second level of abstraction the plan consists of one operator: Puton(B,C). This is because all preconditions of Puton(B,C) are satisfied at this level of abstraction, and because the remaining goals, $On^2_{rel}(A, B)$ and $On^2_{rel}(C, D)$ are also satisfied.

As PABLO moves down to the first abstraction level, the goal $On^1_{rel}(C, D)$ is no longer satisfied since it requires two steps to accomplish. PABLO grows the plan by adding the operator Puton(C,D) to achieve this goal. Notice that at the first level of abstraction all preconditions to Puton(C,D) hold, since D is clear and block C can be cleared in one step. PABLO then completes the plan at the base level.

Now, suppose PABLO is interrupted after it has completed planning at abstraction level 2. At this level, there are three predicates that hold abstractly, i.e., the components of their relaxed definitions that are satisfied have non-null operator lists associated with them. These are $Clear^2_{rel}(B)$, $Clear^2_{rel}(C)$, and $On^2_{rel}(C, D)$. To see this, examine the second level predicate relaxation of Clear(x).

$Clear^2_{rel}(x)$	
Clear(x)	[]
∃ y On(y,x) ∧ Clear(y)	[Tableopr(y)]
∃ y,z On(y,z) ∧ Clear(y) ∧ On(z,x)	[Tableopr(y),Tableopr(z)]

The above table has been simplified by removing subsumed expressions. For instance, the result of regressing Clear(x) through Puton(y,z) is

$$\exists y, z\ On(y, x) \wedge Clear(y) \wedge Clear(z)$$

This expression is not included in the table since it is subsumed by the regression of Clear(x) through Tableopr(y).

When $Clear^2_{rel}(x)$ is instantiated in our plan, with variable x bound to C, $Clear^2_{rel}(C)$ becomes:

$Clear^2_{rel}(C)$	
Clear(C)	[]
On(D,C) ∧ Clear(D)	**[Tableopr(D)]**
∃ y,z On(y,z) ∧ Clear(y) ∧ On(z,C)	[Tableopr(y),Tableopr(z)]

As can be seen from the predicate relaxation definition, $Clear^2_{rel}(C)$ holds because On(D,C) ∧ Clear(D), and the leftmost regressed operator associated with this expression is Tableopr(D). In brief, the three relaxed predicates hold in the initial state for the following reasons:

$Clear^2_{rel}(C)$	On(D,C) ∧ Clear(D)	[Tableopr(D)]
$Clear^2_{rel}(B)$	On(A,B) ∧ Clear(A)	[Tableopr(A)]
$On^2_{rel}(C,D)$	Clear(D) ∧ On(D,C)	[Tableopr(D), Puton(C,D)]

The above relaxed predicates are grounded in the initial state. If PABLO is interrupted after having completed planning at abstraction level 2, it can choose from among the identified action sequences that are executable in the initial state. In this case, it can propose executing Tableopr(D), Tableopr(A), or Tableopr(D) - Puton(C,D). This can be determined as soon as the first level of abstraction has been completed, early in the planning process. In this example, it is done after only 15% of the total planning time.

If necessary, PABLO can construct a substantial portion of the plan, even at this early stage. First, the lists of operators associated with the abstractly satisfied predicates are collected. A plan is built by repeatedly appending applicable sequences of operators. When no more applicable sequences exist, if there is some action in the original plan that is applicable in the last state of this provisional plan, the action is appended to it. We continue appending to the provisional plan in this manner until we have no more sequences of operators or actions in the plan from which to choose. See figure 6 for the incomplete plan constructed using this technique. The full algorithm can be found in [Christensen, 1990].

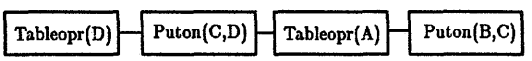

Figure 6: Provisional Plan

The plan is almost complete, the only remaining action is Puton(A,B). Once PABLO commits to the first portion of the plan, developing the remaining portion can be considerably easier.

Here, there is little interaction among the executable alternatives. However, there are obviously cases where such interactions exist. Of course, the only way to discover and resolve conflicts based on such interactions is to continue planning. Until a complete plan is produced, we cannot guarantee that the optimal action will be chosen by PABLO (or any other planner), should it be interrupted. This technique, as opposed to the traditional planning algorithm, produces viable alternatives early on in the planning process. Given more time, PABLO will complete plans at succeedingly lower levels, thereby resolving conflicts not discovered at higher levels, and so producing more reliable answers. Our method, in effect, provides a primitive *anytime* algorithm for planning [Dean and Boddy, 1988].

Unlike a forward search of the state space, PABLO can take advantage of the least-commitment implicit in nonlinear plans. Furthermore, when it is interrupted, its choice for a plausible executable action is derived from a complete abstract plan, which provides a global constraint on this action. An interrupted forward state-space search on the other hand, can only provide local constraints on its choice of executable actions.

Unlike the technique of continuing planning down the leftmost wedge of the plan after an interruption, our approach requires only a bounded computation time to produce an executable action after an interruption. To see this, note that the executable actions are automatically identified after planning at the highest level of abstraction has been completed.

Reactive Plans

The trace of the relaxation of a predicate is in effect a reactive plan for achieving that predicate. See figure 7 for an illustration of the definition of the $On^2_{rel}(x,y)$ predicate as a reactive plan. Notice that some predicates in the reactive plan are not further regressed. This is because these are preconditions to the operator that we do not wish to plan to achieve, but rather just check that they hold. These predicates are specified in the operator definitions given to PABLO. During planning, when a relaxed predicate is determined to hold, the path through the reactive plan that will lead to the establishment of the predicate is automatically identified.

Our technique is a method for handling these small reactive plans. We believe that this is a more promising approach to reactivity than constructing large, unwieldy reactive plans which risk succumbing to space restrictions very quickly. Each individual plan is restricted in size and can be used by the planner on different instantiations of the same predicate.

Each reactive plan in our system has a clear purpose, namely to achieve a particular predicate. Unlike other reactive planning techniques which must construct a new reactive plan for each combination of goals encountered (modulo some parameters to the reactive plan), PABLO can reutilize the reactive plan definitions for any goals specified in the domain.

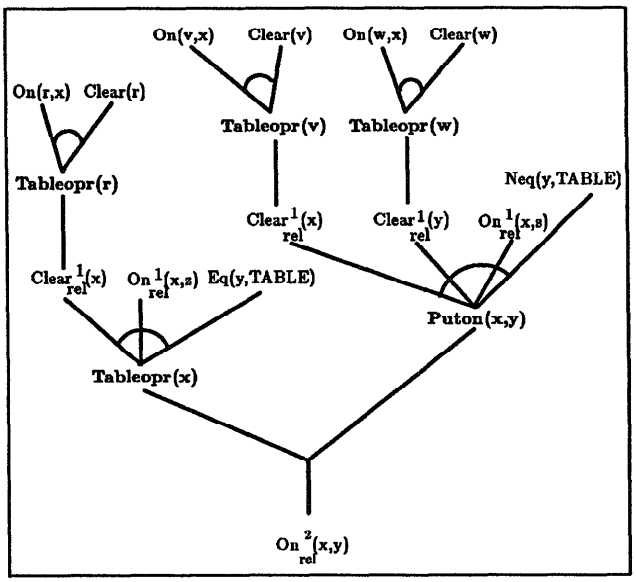

Figure 7: Reactive plan for $On^2_{rel}(x, y)$

If our domain is large enough we risk creating abstraction definitions that are too large, although they will always be considerably smaller than reactive plans created for entire domains, since we are only considering reactive plans for individual predicates.

With PABLO, we can extend the planning method to restricted reactive plans, e.g., allow only commonly encountered conditions in the relaxed definitions. Although this reduces the number of abstractions identified at the higher levels, each predicate can be more quickly identified to hold abstractly. PABLO is robust in the sense that if a predicate is not deemed to hold abstractly, it can plan to achieve it. This is something systems which rely solely on reactive plans cannot do.

Conclusion

We have presented PABLO, a nonlinear hierarchical planner that automatically generates abstraction spaces using predicate relaxation. PABLO is able to solve some problems, e.g., Towers of Hanoi, making full use of the abstractions inherent in the domain. Furthermore, PABLO achieves a limited form of reactivity in that it can produce a sequence of executable actions should it be interrupted before the final plan has been completed.

Acknowledgements

I wish to thank my adviser Nils Nilsson for many helpful discussions related to this work. Matt Ginsberg, Craig Knoblock, Andrew Kosoresow, Alon Levy, Karen Myers, Eunok Paek, Yoav Shoham, Rich Washington, David Wilkins, and the Principia group at Stanford have also made many helpful suggestions.

References

[Benjamin, 1990] Benjamin, D.P., (Ed.), *Change of Representation and Inductive Bias*, Boston, MA: Kluwer, 1990.

[Chapman, 1987] Chapman, D., "Planning for Conjunctive Goals," *Artificial Intelligence*, v 32: 333-378, July 1987.

[Christensen, 1990] Christensen, J., "Abstraction in Planning," forthcoming PhD Dissertation, Stanford University, 1990.

[Dean and Boddy, 1988] Dean, T. and Boddy M., "An Analysis of Time-dependent Planning," in *Proceedings AAAI-88*, pages 49-54, 1988.

[Drummond, 1989] Drummond, M., "Situated Control Rules," *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*," Toronto, Canada, 1989.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J., "STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, 2(3-4): 189-208, 1971.

[Ginsberg, 1989] Ginsberg, M., "Universal Planning: an (Almost) Universally Bad Idea," in *AI Magazine*, vol. 10, no. 4, winter 1989.

[Knoblock, 1990] Knoblock, C., "A Theory of Abstraction for Hierarchical Planning," in D.P. Benjamin (Ed.), *Change of Representation and Inductive Bias*, Boston, MA: Kluwer, 1990.

[Korf, 1987] Korf, R. E., "Planning as Search: A Quantitative Approach," *Artificial Intelligence*, 33(1): 65-88, 1987.

[Nilsson, 1990] Nilsson, N. J., Moore, R., Torrance, M. C., "ACTNET: An Action-Network Language and its Interpreter (A Preliminary Report)," forthcoming.

[Sacerdoti, 1974] Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, v 5: 115-135, 1974.

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

[Schoppers, 1987] Schoppers, M. J., "Universal Plans for Reactive Robots in Unpredictable Environments," in *Proceedings AAAI-87*, 1987.

[Washington, 1989] Washington, R., "Abstraction Planning in Real Time," Ph.D. Thesis Proposal, unpublished, 1989.

[Wilkins, 1988] Wilkins, D. E., *Practical Planning*, Morgan Kaufman, San Mateo, California, 1988.