ABTWEAK: Abstracting a Nonlinear, Least Commitment Planner *

Qiang Yang
Computer Science Department
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
qyang@watdragon.waterloo.edu.

Josh D. Tenenberg
Computer Science Department
University of Rochester
Rochester, New York, U.S.A., 14627
josh@cs.rochester.edu

Abstract

We present the system AbTweak, which extends the precondition-elimination abstraction of Abstrips to hierarchical planners using the nonlinear plan representation as defined in Tweak. We show that AbTweak satisfies the *monotonic property*, whereby the existence of a lowest level solution Π implies the existence of a highest level solution that is *structurally similar* to Π. This property enables one to prune a considerable amount of the search space without loss of completeness.

Abstracting Planning Systems

Abstraction in planning systems can be viewed as a mapping from one problem description (at a *concrete* level) to another(at the *abstract* level). There has been a considerable amount of research recently in formalizing intuitions regarding abstraction and the hierarchical problem solving strategies that abstraction gives rise to [Fikes *et al.*, 1972, Knoblock, 1989, Korf, 1985b, Nau, 1987, Sacerdoti, 1974, Tate, 1977, Tenenberg, 1988, Wilkins, 1984, Yang, 1989].

However, there has been little work in extending the formal results from linear STRIPS-like planners to richer temporal planners, such as the nonlinear planners of Sacerdoti [1977] and Chapman [1987]. The advantage of these planners over linear planners is that they allow temporal order and operator instantiations to be only partially specified through the posting of constraints; any fully specified plan consistent with these constraints is guaranteed to solve the given problem.

One particular type of abstraction that we have previously formalized [Tenenberg, 1988], involves the elimination of a subset of the predicates in the language as one ascends the abstraction hierarchy (a generalization of the precondition elimination strategy of Abstrips [Sacerdoti, 1974]). The predicates of the planning system are partitioned, which induces a partition on the preconditions of the operators, and each partition is assigned an integer value, a *criticality*. Each abstract level i is derived from the previous level by eliminating those preconditions having criticality $< i$, otherwise being identical.

We define the system AbTweak, and demonstrate in this paper that precondition-elimination abstraction can be naturally extended to nonlinear least commitment planners, and thus benefit from the advantages of both abstraction and nonlinearity. Most importantly, we show that each AbTweak system has a *monotonically expandable* abstraction space, whereby the existence of a lowest level solution Π implies the existence of a highest level solution that is structurally similar to Π. This property enables one to prune a considerable amount of the search space without loss of completeness. In addition, the abstraction space is monotonic regardless of the criticality assignment, i.e., it does not depend upon obtaining the "right" assignment of criticality values to preconditions.

We first present brief descriptions of Tweak and Abstrips, and then define AbTweak. We demonstrate that AbTweak has the monotonic property, and show how this affects search. All lemmas and theorems are presented without proofs. These proofs can be found in the longer version of this paper [Yang *et al.*, 1991].

*This work was supported in part by an interim research grant to Qiang Yang, from the Faculty of Mathematics at the University of Waterloo, and by grants to Josh D. Tenenberg in part from the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332, under Contract Num. F30602-85-C-0008 which supports the Northeast Artificial Intelligence Consortium (NAIC), in part by ONR/DARPA research contract #N00014-80-C-0197, and in part by US Army Communication-Electronics Command grant #DAAB10-87-D-022.

Nonlinear Planning: TWEAK

Chapman [1987] provides a formalization of a least commitment, nonlinear planner, TWEAK. TWEAK extends STRIPS by allowing for

1. a partial temporal ordering on the operators in a plan,
2. partial constraints on the binding of variables (*codesignations*) of the operators.

A TWEAK plan thus represents a space of STRIPS plans: all totally ordered, fully ground plans that satisfy the ordering and codesignation constraints.

Formally, a TWEAK system is a pair $\Sigma = (L, O)$. L is a restricted language consisting of countably many predicate, constant and variable symbols, but none of the logical connectives (and, therefore, not the quantifiers). The set of *terms* of L is the constants unioned with the variables, and the set of *propositions* is all expressions of the form

$$P(x_1, \ldots, x_n),$$

where P is an n-ary predicate and the x_i are terms. The set of *literals* of L is the set of propositions unioned with the set of negations of propositions.

O is a set of operator templates (referred to simply as operators), defined in terms of preconditions and effects, with the variables in each operator standardized apart. Each precondition or effect is a literal in L. If $a \in O$ is an operator, then P_a is the set of preconditions of a, and E_a is the set of effects of a. An operator a *asserts* literal p if $p \in E_a$, and *denies* p if $\neg p \in E_a$.

Chapman [1987] did not give a formal definition of a TWEAK plan. Because this concept is very important in defining a number of others later in the paper, we formally define it below:

Definition 1 *A plan* Π *is a triple* (A, B, C)*, where*

- $A(\Pi)$ *is a set of operators, defined in terms of preconditions and effects,*
- $B(\Pi)$ *is a partial ordering on* A *("*$\prec$*"),*
- $C(\Pi)$ *is a set of codesignation and non-codesignation constraints of the form* $p \approx q$ *or* $p \not\approx q$*, where* p *and* q *are both either terms or propositions.*

If a, b are operators in $A(\Pi)$, we say $B(\Pi) \vdash (a \prec b)$ if and only if $(a \prec b)$ follows from the transitive closure of $B(\Pi)$. That is, a precedes b under every total ordering that satisfies $B(\Pi)$. If $P(x_1, \ldots, x_n)$ and $P(y_1, \ldots, y_n)$ are propositions, we say

$$C(\Pi) \vdash (P(x_1, \ldots, x_n) \approx P(y_1, \ldots, y_n))$$

if and only if each $(x_i \approx y_i)$ follows from the symmetric and transitive closure of the codesignation constraints of $C(\Pi)$. Likewise for non-codesignation.

With the above definition, we can now restate formally several terminologies used in [Chapman, 1987]. A *complete plan* is a plan where $B(\Pi)$ is a total ordering on $A(\Pi)$, and $C(\Pi)$ is such that every variable in every operator of $A(\Pi)$ codesignates with some constant. A plan *completion* refers to any complete plan that satisfies the partial constraints of a plan.

An input problem is taken to be a pair, $\rho = (I, G)$, where I is the initial state, and G is the goal state, each state consisting of a finite set of literals. A complete plan for a problem implicitly defines a sequence of states: the first element of the sequence is the given initial state, and the $i + 1^{st}$ element of the sequence is the i^{th} state without the literals denied by the i^{th} operator, and including the literals asserted by the i^{th} operator. A proposition is satisfied in a state if it is an element of that state.

For simplicity, the goal G can be represented by a special operator g, where $P_g = E_g = G$. The initial state I can likewise be viewed as a special operator i, with $P_i = \emptyset$ and $E_i = I$. These two operators will be an element of each plan Π, under the constraint that, for every other operator $a \in A(\Pi)$, $(i \prec a)$ and $(a \prec g)$.

A complete plan is *correct* if all preconditions of each operator in the plan are satisfied in the state in which the operator is applied. A complete plan *solves* a problem if it is correct, and the goal is satisfied in the final state. A plan solves a problem if every completion solves the problem; similarly for correctness of plans.

In a partial plan, two terms *necessarily* codesignate, that is, unify, if they codesignate under every completion. Two terms *possibly* codesignate if they codesignate under some completion. Operator a necessarily precedes b if a precedes b under every completion. a possibly precedes b if a precedes b under some completion. We will use $\Box$ and $\Diamond$ to denote necessarily and possibly, and $\approx$, $\not\approx$, and $\prec$ to denote codesignates, non-codesignates and precedes. Necessary and possible precedence, codesignation and noncodesignation can be defined precisely, for plan P, as:

$$\begin{aligned}
\Box(a \prec b) &\iff B(\Pi) \vdash (a \prec b), \\
\Diamond(a \prec b) &\iff \neg\Box\neg(a \prec b) \iff B(\Pi) \nvdash (b \prec a), \\
\Box(p \approx q) &\iff C(\Pi) \vdash (p \approx q), \\
\Diamond(p \approx q) &\iff \neg\Box\neg(p \approx q) \iff C(\Pi) \nvdash (p \not\approx q), \\
\Box(p \not\approx q) &\iff C(\Pi) \vdash (p \not\approx q), \\
\Diamond(p \not\approx q) &\iff \neg\Box\neg(p \not\approx q) \iff C(\Pi) \nvdash (p \approx q).
\end{aligned}$$

The following definitions introduce simplifying notation.

$$\Diamond(a \prec c \prec b) \iff \Diamond(a \prec c) \text{ and } \Diamond(c \prec b),$$
$$\Box(a \prec c \prec b) \iff \Box(a \prec c) \text{ and } \Box(c \prec b).$$

The Modal Truth Criterion

The *modal truth criterion* (MTC) defines the conditions under which an assertion will be true at a point in a partially ordered plan. Chapman [1987] provides a concise statement of the criterion that is both necessary and sufficient. A problem with his definition is that it is stated in terms of *situations*, which are not well-defined in a partially ordered and instantiated plan. For that reason, we provide a modified version of the MTC, defined in terms of operators in a plan.

Coarsely stated, a proposition p is necessarily true in the state in which operator b is applied if there exists, for every total ordering, some operator a that asserts p (adds a proposition that codesignates with p), and for which no operator between a and b asserts $\neg p$.

Definition 2 *Proposition p is necessarily true in the state in which operator b is applied in plan Π if and only if two conditions hold:*

1. *there is an operator $a \in A(\Pi)$ and $u \in E_a$, such that $\Box(a \prec b)$ and $\Box(p \approx u)$, and*
2. *for every operator $c \in A(\Pi)$ and $q \in E_c$, if $\Diamond(c \prec b)$, and $\Diamond(\neg q \approx p)$, then there is an operator $w \in A(\Pi)$ and $r \in E_w$ such that $\Box(c \prec w \prec b)$ and $C(\Pi) \cup \{(\neg q \approx p)\} \vdash (r \approx p)$.*

This last condition says that $(r \approx p)$ whenever $(\neg q \approx p)$.

AbTweak

In Abstrips, Sacerdoti developed an elegant means for generating abstract problem spaces, by assigning criticality values (an integer between 0 and k, for some small k) to preconditions, and abstracting at level i by eliminating all preconditions having criticality less than i. The formalisms for this system are straightforward, and are provided below when criticalities are assigned to the precondition literals in a Tweak system.

A k level AbTweak system is a triple $\Sigma = (L, O, crit)$, where

(1) L is a Tweak language;
(2) O is an operator set, as in Tweak, and
(3) *crit* is a function:

$$\bigcup_{o \in O} P_o \rightarrow \{0, 1, \ldots, k-1\}.$$

Intuitively, *crit* is an assignment of criticality values to each proposition appearing in the precondition of an operator.

Let a be an operator. We take ${}_iP_a$ to be the set of preconditions of a which have criticality values of at least i:

$${}_iP_a = \{p \mid p \in P_a \textit{ and } crit(p) \geq i.\},$$

and ${}_ia$ is operator a with preconditions ${}_iP_a$ and effects E_a. Let the set of all such ${}_ia$ be ${}_iO$. This defines a TWEAK system on each level i of abstraction:

$${}_i\Sigma = (L, {}_iO).$$

Upward Solution Property

As with Abstrips the strategy for planning with AbTweak is governed by length first search. When a problem is input, planning proceeds first at the most abstract, least constrained level. This plan is then expanded at the next lower level by inserting new operators to satisfy the re-introduced preconditions. Only after all the preconditions are satisfied on the current level does the planner pass the plan to the level below. The primary reason for using this control strategy is for solving the frame problem.

Implicit in this strategy is the assumption that short plans to solve a given problem are guaranteed to exist at the abstract level which can be successively expanded, and that search strategies exist to find such abstract plans. Our intent is to formally prove this property, and to show how it places some useful constraints on search. The intuition behind the proof is to show that if there exists a lowest (base) level solution to a problem, then this solution will also solve the problem at each higher level of abstraction, since these higher levels do not place any new constraints on the problem. Further, since there are fewer preconditions at the higher levels, one can eliminate from this plan those operators whose purpose at lower levels is solely to satisfy one of the eliminated preconditions, either directly or indirectly.

For instance, consider a plan for getting a box from one room into an adjacent room, in which the robot picks up the box, goes to the door, sets the box down, opens the door, picks up the box, and goes through the doorway. Suppose that the status of the door – whether it is open or closed – is ignored at the abstract level. In this case, since opening the door is no longer considered as a precondition, the intermediate steps of setting down and re-picking up the box are no longer necessary; their sole purpose was to free the agent's hands for the door opening. Thus, the abstract level plan is simpler than the concrete level plan.

Ascending Preserves Correctness

For notational simplicity, if Π is a plan on the base level, then for $i = 1, 2, \ldots, k-1$, let ${}_i\Pi$ represent the plan formed by replacing every occurrence of a (except i and g) in Π by ${}_ia$. As defined above, a plan Π is correct if and only if $\forall a \in A(\Pi), \forall q \in P_a$, q is necessarily true in the state in which a is applied. Removing a precondition of an operator while holding the plan fixed does not affect the necessary truth of any condition. Thus, after removing a precondition of an operator in Π, the resulting plan Π' is still correct. However, we can establish a stronger property. Namely, if Π is a plan correct at the base level, then a plan ${}_i\Pi'$ is also a correct plan on level i, where ${}_i\Pi'$ is simpler than Π in that it is ${}_i\Pi$ with possibly one or more operators removed. Thus, $A({}_i\Pi')$ is possibly smaller than $A(\Pi)$. Moreover, the constraints in ${}_i\Pi'$ are $B(\Pi)$ and $C(\Pi)$ with possibly one or more constraints removed. Thus, the plan ${}_i\Pi'$ is less constrained than Π. This will be shown by specifying the *precondition establishment* structure of the plan, that is, which operators satisfy preconditions of other operators, either directly or indirectly.

Definition 3 *Let Π be a correct plan. Let a and b be operators in $A(\Pi)$, p be a precondition of b, and u be an effect of a. Then a* establishes *p for b with u (*Establishes(a, b, p, u)*) if and only if*

1. $\Box(a \prec b)$,
2. $\Box(u \approx p)$, *and*
3. $\forall a' \in A(\Pi), \forall u' \in E_{a'}$, *if* $\Box(a \prec a' \prec b)$, *then* $\neg\Box(u' \approx p)$.

This final condition states that a must be the last such operator that necessarily precedes b which necessarily asserts precondition p.

Given this definition, it can be proven that every precondition in every operator of a correct plan has an establisher.

Lemma 4 *Let Π be a correct plan, $b \in A(\Pi)$, and $p \in P_b$. $\exists a \in A(\Pi), \exists u \in E_a$ such that* Establishes(a, b, p, u).

Informally, a *clobberer* is an operator which possibly precedes and possibly denies the precondition of another operator in the plan. A *white knight* is another operator which necessarily re-establishes this clobbered precondition.

Definition 5 *c is a clobberer of b, (*CB(c, b, p, q)*) if and only if*

(1) $p \in P_b$,
(2) $q \in E_c$,
(3) $\Diamond(\neg q \approx p)$,
(4) $\exists a, u$ such that Establishes(a, b, p, u),
(5) $\Diamond(a \prec c \prec b)$.

Definition 6 *w is a white knight for b, (*WK(w, b, c, p, q, r)*), if and only if*

(1) CB(c, b, p, q),
(2) $r \in E_w$,
(3) $\Box(c \prec w \prec b)$, and
(4) $C(\Pi) \cup \{(\neg q \approx p)\} \vdash (r \approx p)$.

An operator or constraint in a plan is justified if it is subservient, directly or indirectly, to the satisfaction of the goal.

Definition 7 *Let Π be a plan, and i, g be the special operators for the initial and goal states. Then in plan Π,*

Initial/Goal justification *i and g are justified,*

Establishment justification *If b is justified, and $\exists a, \exists u \in E_a, \exists p \in P_b$ such that* Establishes(a, b, p, u), *then*

(1) a is justified, (2) $(a \prec b)$ is justified, (3) $(u \approx p)$ is justified.

White knight justification
If WK(w, b, c, p, q, r) *and b and c are justified, then w, $(c \prec w)$, and $(w \prec b)$ are justified. Moreover, let D be a minimal set of codesignation constraints such that $D \cup \{(\neg q \approx p)\} \vdash (r \approx p)$. Then every codesignation constraint in D is also justified.*

Separation justification *If c and b are justified, and $\exists p \in P_b, \exists q \in E_c$ such that $\Box(p \not\approx \neg q)$, then $(p \not\approx \neg q)$ is justified.*

Precedence Justification *If b and c are justified and $\Box(b \prec c)$, then $b \prec c$ is also justified.*

Nothing else is justified.

The justification of plan Π, Jus(Π), is the set of operators, precedence and codesignation constraints of Π that are justified. It is obvious that the justified version of a plan is simpler than the plan itself, in the sense that the set of operators, precedence and codesignation constraints of the justified plan are a subset of those in the unjustified plan.

Lemma 8 *If Π is a correct plan that solves goal G, then* Jus(Π) *also is a correct plan that solves G.*

The following theorem establishes the *Upward Solution Property*: if there is a solution to a problem at the base level, then the justified version of that solution at each higher level of abstraction is correct, and also solves the problem on that level. More formally,

Theorem 9 *If* Π *is a correct plan that solves* G *at the base level, then the justified version of* ${}_i\Pi$ *is also a correct plan that solves* G *on the* i^{th} *level,* $0 \leq i \leq k-1$.

Monotonic Expansion

The Upward Solution Property guarantees the existence of an abstract level solution to a problem, whenever there exists a lowest level solution. Length-first search, on the other hand, proceeds from the highest level to the lowest. Since the converse of the Upward Solution Property does not hold, one cannot be sure that an arbitrary solution obtained at the abstract level is one which can be expanded into a low level solution. It is therefore important to uncover constraints that will be helpful in plan expansion. The monotonic property is one such constraint, and was first defined by Knoblock, for linear ABSTRIPS systems [Knoblock, 1989]. We will define it here (in a slightly different form than Knoblock).

Definition 10 *Let* Π' *be a level* i *plan, and* Π *a level* $i-1$ *plan.* $c : A(\Pi') \mapsto A(\Pi)$ *is a correspondence function if and only if*

1. c *is* $1-1$ *and* into, *and*

2. $\forall a \in A(\Pi'),\ {}_i(c(a)) = a$.

Definition 11 *Let* Π' *be an abstract plan that solves* ρ *at level* i, $i > 0$. Π' monotonically expands *to level* $i-1$ *plan* Π *if and only if*

1. Π *solves* ρ *at level* $i-1$, *and*

2. *there exists a correspondence function* $c : \Pi' \mapsto \Pi$ *such that* $\forall a, b, p, u$ *if* Establishes(a, b, p, u) *in* Π' *then* Establishes$(c(a), c(b), p, u)$ *in* Π.

Definition 12 *A* k*-level* ABTWEAK *system is monotonic, if and only if, for every problem* ρ *solvable at the concrete* (0^{th}) *level, there exists a sequence of plans* $\Pi_{k-1}, \ldots, \Pi_0$ *such that* Π_{k-1} *solves* ρ *at level* $k-1$, *and for* $0 < i < k$, Π_i *monotonically expands to* Π_{i-1}.

The following Lemma can now be proven:

Lemma 13 *Every* ABTWEAK *system of* k *levels, for any* k, *is monotonic.*

Search Control

In this section, we explore the implications of the Monotonic Property on search control in ABTWEAK. We will discuss global completeness of ABTWEAK as an abstraction system, and show that ABTWEAK can backtrack on violations of higher-level establishment relations and unresolvable conflicts.

Completeness of ABTWEAK

Search for a plan with ABTWEAK proceeds in a *length-wise* fashion, by first finding a plan at the most abstract level, and then, for each lower level i, expanding the plan Π from level $i+1$ by inserting operators into Π, or imposing new constraints to satisfy the re-introduced preconditions in Π. Thus, ABTWEAK searches for a correct concrete-level plan in a space of abstract plans. In this search space, if a plan Π is not correct yet on an abstract level, then the set of state-space operators applicable to Π is the set of plan modification operations in TWEAK. On the other hand, if a plan Π is correct on level $i > 0$, then the state-space operator is simply plan expansion, which inserts all $i-1$ level preconditions to each operator in Π.

In this section, we discuss the global search control strategy for ABTWEAK. Before describing it in detail, we first explain what seems to be an obvious choice for search control, and why it is not used for ABTWEAK.

We first define what we mean by completeness, and monotonic completeness:

Definition 14 *A control strategy is* complete *if whenever there is a solution at the concrete level, the strategy will terminate by finding a solution.*

Definition 15 *A control strategy for a* k*-level* ABTWEAK *system is* monotonically complete *if and only if for every problem* ρ *solvable at the concrete* (0^{th}) *level, the strategy outputs a sequence of plans* $\Pi_{k-1}, \ldots, \Pi_0$ *such that* Π_{k-1} *solves* ρ *at level* $k-1$, *and for* $0 < i < k$, Π_i *monotonically expands to* Π_{i-1}.

Our aim is to explore control strategies that are monotonically complete.

An intuitively obvious choice of control is to use a separate TWEAK for control on each level of abstraction, similar to the way ABSTRIPS uses STRIPS. This is especially appealing, since it is not difficult to specify complete control strategies for TWEAK, either using a complete state-space search procedure such as A*, or breadth-first search, or the procedure provided by Chapman [1987]. Using this approach, if a plan is formed on abstraction level i, then it is passed down to the level below. At level $i-1$, all the conditions of criticalities no less than $i-1$ are planned for. The process continues, until either a correct plan is formed at the base level, or it is found that a plan cannot be made correct at a level. Then the planner backtracks to the level immediately above the current one, and tries to find an alternative solution.

The fact that TWEAK is complete may lead one into believing that the above control structure is also monotonically complete. Unfortunately, this is not the case in general. The reason is that any search strategy

for TWEAK will be semi-decidable, in the sense that if there is no solution, it is not guaranteed to terminate. Suppose that a plan Π is found on level $i+1$ that is not monotonically expandable, and is passed to the level i below. Then it is possible for TWEAK to run forever, without knowing it should backtrack to the level above. Incompleteness may result since there may exist a correct solution at the concrete level, and Π cannot be expanded to that solution.

Thus, although on each level of abstraction completeness is guaranteed separately, it is not ensured monotonically. A complete search strategy will be obliged to do a "diagonalizing enumeration," that is, it cannot simply pick an abstract plan, and attempt to specialize this plan further without regard to the remaining abstract solutions, but must instead do only a quanta of planning steps, and go to the next abstract solution. But, it cannot be simple minded about this either, since there may be an infinite number of abstract solutions. So, it must do some quanta on the next abstraction, and then return to the first one. That is, the enumeration must "diagonalize in two dimensions."

The above argument suggests a monotonically complete control procedure, in the sense that any state in ABTWEAK's search space may be selected next according to a complete search control strategy. Recall that ABTWEAK's search space operations include not only the plan modifications of TWEAK, but also the plan expansions. Thus, if a path exists in the original state space from the initial state to a goal state, one such path will eventually be found. Any complete search strategy will suffice for the purpose: breadth-first, A^* [Nilsson, 1980], depth-first iterative deepening[Korf, 1985a], etc.

Backtracking on Protection Violations

The monotonic property provides a powerful heuristic for guiding the search in ABTWEAK. It can be considered as a criteria for backtracking that does not sacrifice completeness. More specifically, one can backtrack on precondition-establishment violations, that is, if for some operators a and b, and literals p and u, Establishes(a, b, p, u) in a plan at abstraction level i, then at level $i-1$, if the only choices left are to insert an operator that possibly asserts $\neg p$, then ABTWEAK can backtrack without losing monotonic completeness. Thus, the causal relation between preconditions and effects should be preserved when going down abstraction levels. This effectively imposes a strong constraint on how an abstract plan should be refined at a lower level.

Backtracking on Incompleteable Plans

Sometimes no solution can be found at a particular level of abstraction. In that case, one would like to know whether a solution exists at the base level. For ABTWEAK, it follows from the Upward-Solution Property that if there is no solution at one level, then no solution exists at all at any lower levels of abstraction.

A related problem is whether to backtrack from an incompleteable plan. A plan Π is said to be *incompleteable* if no correct completion of Π exists, and no operators and constraints can be inserted to obtain a correct completion. One way for Π to be incompleteable is that it contains a set of clobberers of the operators in Π, and that no white knights and constraints exist to remove all of the clobbering. This situation corresponds to what is commonly known as the "unresolvable conflicts" in nonlinear planning. It can be proven that ABTWEAK can backtrack from an incompleteable plan without losing completeness[Yang *et al.*, 1991].

Conclusion

This research has been aimed at formalizing domain-independent, nonlinear planning systems that plan in hierarchies of abstraction levels. The resultant planner, ABTWEAK, extends the precondition-elimination methods in ABSTRIPS for building abstraction hierarchies, and allows for least-commitment representations of plans in TWEAK. We have shown that ABTWEAK satisfies the monotonic property, that is, as planning descends from top to concrete levels of abstraction, the precondition establishment structure of a plan need not be changed. This, to a large extent, formalizes our intuition for using abstraction in planning: that it is generally more efficient to use an abstract solution to guide search at lower levels of abstractions than without abstraction. In addition, we have demonstrated that a simplistic application of a control strategy for a single-level problem solver to each level of the abstraction hierarchy will not in general provide a complete multiple-level system. We also discussed how to ensure the monotonic completeness for ABTWEAK systems.

We believe that ABTWEAK also offers computational advantages over some of the existing hierarchical planning systems. However, to provide concrete evidence for this claim, it might take a considerable amount of experimentation. Indeed, our ongoing work is to implement ABTWEAK and make such computational comparisons.

Acknowledgements

We thank Craig Knoblock for many useful comments.

References

[Chapman, 1987] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[Fikes *et al.*, 1972] Richard Fikes, Peter Hart, and Nils Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Knoblock, 1989] Craig A. Knoblock. A theory of abstraction for hierarchical planning. In Paul Benjamin, editor, *Proceedings of the Workshop on Change of Representation and Inductive Bias*, Boston, MA, 1989. Kluwer.

[Korf, 1985a] Richard Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[Korf, 1985b] Richard Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1985.

[Nau, 1987] Dana Nau. Hierarchical abstraction for process planning. In *Proceedings of Second International Conference in Applications of Artificial Intelligence in Engineering*, 1987.

[Nilsson, 1980] Nils Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1980.

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

[Sacerdoti, 1977] Earl Sacerdoti. *A Structure for Plans and Behavior*. American Elsevier, 1977.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 888–893, San Mateo, CA, 1977. Morgan Kaufmann Publishers, Inc.

[Tenenberg, 1988] Josh Tenenberg. *Abstraction in Planning*. PhD thesis, University of Rochester, Dept. of Computer Science, Rochester, NY, May 1988.

[Wilkins, 1984] David Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22, 1984.

[Yang *et al.*, 1991] Qiang Yang, Josh Tenenberg, and Steve Woods. Abstraction in nonlinear planning. Technical Report CS 91-65, University of Waterloo, Department of Computer Science, Waterloo, Ontario, Canada N2L 3G1, 1991.

[Yang, 1989] Qiang Yang. *Improving the Efficiency of Planning*. PhD thesis, University of Maryland, 1989.