Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA, 1991

Characterizing Abstraction Hierarchies for Planning*

Craig A. Knoblock
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213
cak@cs.cmu.edu

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

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

Abstract

The purposes of this paper are threefold. The first is to provide a crisp formalization of ABSTRIPS-style abstraction, since the lack of such formalizations has made it difficult to ascertain the uses and value of this type of abstraction in previous research. Second, we define the *refinement* relationship between solutions at different levels of the abstraction hierarchy. Such definitions are crucial to developing efficient search strategies with this type of hierarchical planning. And third, we provide a restriction on the abstraction mapping that provides a criterion for generating useful abstractions.

Introduction

Ever since Sacerdoti's ABSTRIPS system [Sacerdoti, 1974], researchers have used the technique of eliminating preconditions of operators in order to form abstraction spaces for planning [Christensen, 1990, Tenenberg, 1988, Unruh and Rosenbloom, 1989, Yang and Tenenberg, 1990]. There is some empirical evidence that suggests that such abstraction systems can significantly reduce search, but the selection and evaluation of the individual abstraction hierarchies is confined to the individual systems. As a result, it has been difficult to 1) explicate heuristics that are used implicitly to construct abstraction hierarchies, 2) compare the results between different systems, and 3) uncover and evaluate new heuristics and properties. Even more seriously, the characterization of "good" abstraction hierarchies has remained at an informal level.

This paper begins to fill this gap by formalizing, unifying, and extending the previous work on ABSTRIPS-type planning systems. These systems are characterized by the elimination of precondition constraints on the operators as one ascends the hierarchy. Planning occurs in a *length-first* fashion [Sacerdoti, 1974], by first planning at the most abstract (least constrained) level, and refining this plan at successively lower levels by inserting new plan steps which satisfy the reintroduced preconditions which were eliminated during abstraction By formalizing this refinement process, we are able to provide a set of conditions that can be enforced between the concrete- and abstract-level *representations*. These conditions guarantee certain relationships between the abstract and concrete *solutions* that provide strong constraints on search.

In particular we show that all abstraction hierarchies have the *monotonicity property*, whereby the existence of a concrete-level solution Π implies the existence of an abstract solution that is structurally similar to Π. This property enables one to prune a considerable amount of the search space without loss of completeness. Yet, the monotonicity property fails to characterize the intuition behind "good" abstraction hierarchies. We also identify a restriction of the monotonicity property, called the *ordered monotonicity* property, which holds when *every* refinement of an abstract plan is guaranteed to leave the abstract plan structurally unchanged. This property reduces the search space even further than allowed by the monotonicity property. More importantly, the property provides a qualitative characterization of the intuition behind a "good" abstraction hierarchy. We show that this property can be guaranteed by imposing sufficient syntactic restrictions on the abstraction hierarchy. As a result, it is straightforward to automatically *generate* abstraction hierarchies with the ordered monotonicity property.

With the formalization of abstraction hierarchies, and the characterization of the various properties they possess, it is easy to see the relative merits and drawbacks of different planning systems with abstraction.

*The first author is supported in part by an Air Force Laboratory Graduate Fellowship, and in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. The second author is supported in part by ONR/DARPA under research contract N00014-80-C-0197, and in part by U.S. Army Communication-Electronics Command grant DAAB10-87-D-022. The third author is supported in part by an operating grant from Natural Sciences and Engineering Research Council of Canada number OGP0089686.

We show that many systems can be placed on a *spectrum*; at one end, the abstract level provides only very weak heuristic power, but is applicable to a broad range of problems, while the opposite end of the spectrum is heuristically strong, but narrowly applicable. By developing precise formal characterizations, we hope to provide a context in which researchers can determine which place along the spectrum balances the trade-offs appropriately for problem solving within their domain.

We will first present a formalization of these properties and identify sufficient conditions on the abstraction hierarchies that guarantee the satisfaction of these properties. We then discuss the relationship between several existing abstraction systems in the context of our formal characterization. The definitions and properties presented in this paper are based on earlier work presented in [Knoblock, 1990b, Tenenberg, 1988, Yang and Tenenberg, 1990]. Theorems are presented without proofs, but the proofs can be found in [Knoblock, 1991].

Abstract Problem Spaces

Problem-Space Language

A problem space is defined by a set of states, and a set of operators that map between states. Problem-space search involves finding a sequence of operators, called a plan, that transforms a given *initial* state into a *goal* state. Formally, a problem space is a triple $\sigma = (L, S, O)$, where L is a first-order language, S is a set of states, and O is a set of operators. Each $S_i \in S$ is a finite set of atomic sentences of L. We apply the closed-world assumption to each S_i, so that implicitly, if Q is an atom of L not in S_i, we take $\neg Q$ to be an element of S_i. To simplify the reading, we will not encumber our notation by including this implicit set of negated atoms in every state definition, leaving it to the reader to make this inclusion.

Each operator α is defined by a corresponding triple $(P_\alpha, D_\alpha, A_\alpha)$, where P_α, the precondition list, is a set of literals (positive or negative atomic sentences) in L, and both the delete list D_α and add list A_α are finite sets of atomic sentences in L. Applying an operator α to a state S_i produces a new state by removing the deleted literals from, and inserting the added literals of α to S_i, in that order. A *plan* $\Pi = \langle \alpha_1, \ldots, \alpha_n \rangle$ is a sequence of operators, which can be applied to a state by executing each operator in order.

A *problem* is a pair (S_0, S_g), where $S_0 \in S$ is the *initial state*, and $S_g \in S$ is the *goal state*. A plan $\Pi = \langle \alpha_1, \ldots, \alpha_n \rangle$ is *correct* relative to an initial state whenever the preconditions of each operator are satisfied in the state in which the operator is applied, *i.e.*, $P_{\alpha_i} \subseteq S_{i-1},\ i = 1, \ldots n$. A plan Π *solves* goal S_g whenever Π is correct and the goal is satisfied in the final state: $S_g \subseteq S_n$. We take $\alpha \prec_\Pi \beta$ to mean that operator α precedes operator β in plan Π. The subscript to $\prec$ will be dropped if the intended plan is unambiguously identified.

Criticalities

Formally, a *k-level abstraction hierarchy* is a quadruple $\Sigma = (L, S, O, crit)$, where L, S, and O are just as in the problem space definition, and *crit* is a function assigning one of the first k non-negative integers to each precondition of each operator. Note that under this definition, the same literal may be assigned different criticalities when it appears as a precondition in different operators. Thus, criticality is a two-place function of both the precondition *and* the operator.

Let α be an operator. We take ${}_iP_\alpha$ to be the set of preconditions of α that have criticality values of at least i:

$$ {}_iP_\alpha = \{p \mid p \in P_\alpha \text{ and } crit(p, \alpha) \geq i.\},$$

and ${}_i\alpha$ is operator α with preconditions ${}_iP_\alpha$, adds A_α, and deletes D_α. Let the set of all such ${}_i\alpha$ be ${}_iO$. This defines a problem space on each level i of abstraction:

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

Example 1 Consider the Tower of Hanoi domain with 3 pegs and 3 disks. Let the three pegs be `P1`, `P2`, and `P3`, and let the disks be `Large`, `Medium` and `Small`. We can represent the location of the disks using literals of the form `OnLarge(x)`, `OnMedium(x)`, and `OnSmall(x)`. Initially, all disks are on `P1`, and in the goal state they are on `P3`. The operators for moving the disks can be represented as shown in Table 1. Note that the add list is denoted by the unnegated literals, and the delete list by the negated literals. One criticality assignment is to assign each `Ispeg` and `OnLarge` predicate to level 2, each `OnMedium` predication to level 1, and each `OnSmall` predication to level 0. Thus, at the highest abstraction level, each operator has only the `Ispeg` and `OnLarge` preconditions. At this highest abstraction level, a plan to solve the problem of getting all pegs to `P3` from an initial state in which all operators are on `P1`, is

$$\langle \texttt{MoveL(P1, P3)}, \texttt{MoveM(P1, P3)}, \texttt{MoveS(P1, P3)} \rangle.$$

Refinement of Abstract Plans

Establishment

Abstract planning is usually done in a top-down manner. An abstract solution is first found on the k^{th} level of abstraction. Then it is refined to account for successive levels of detail. We formalize the notion of refinement by first defining the concept of "operator establishment." Intuitively, an operator α establishes a precondition of another operator β in a plan, if it is the last operator before β that achieves that precondition.

Definition 1 *Let* Π *be a correct plan. Let* $\alpha, \beta \in$ operators(Π), $p \in P_\beta, A_\alpha$. *Then* α establishes p for β (establishes(α, β, p)) *if and only if*

Preconditions	*Effects*
MoveL(x,y)	
Ispeg(x), Ispeg(y), ¬OnSmall(x), ¬OnSmall(y), ¬OnMedium(x), ¬OnMedium(y), OnLarge(x)	¬OnLarge(x), OnLarge(y)
MoveM(x,y)	
Ispeg(x), Ispeg(y), ¬OnSmall(x), ¬OnSmall(y), OnMedium(x)	¬OnMedium(x), OnMedium(y)
MoveS(x,y)	
Ispeg(x), Ispeg(y), OnSmall(x)	¬OnSmall(x), OnSmall(y)

Table 1: Operators for the Tower of Hanoi

(1) $\alpha \prec \beta$,
(2) $\forall \alpha' \in$ operators(Π), *if* $\alpha \prec \alpha' \prec \beta$, *then* $p \notin A_{\alpha'}$.

This final condition states that α must be the last operator that precedes β and adds precondition p. Since Π is a correct plan, this implies that there is additionally no operator between α and β that deletes p.

Justification

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

Definition 2 *Let* Π *be a correct plan, and* S_g *a goal.* $\alpha \in$ operators(Π) *is justified with respect to* S_g *if and only if there exists* $u \in A_\alpha$ *such that either*

(1) $u \in S_g$, *and* $\forall \alpha' \in$ operators(Π), *if* $(\alpha \prec_\Pi \alpha')$ *then* $u \notin A_{\alpha'}$, *or*
(2) $\exists \beta \in$ operators(Π) *such that* β *is justified, and* establishes(α, β, u).

If α is justified in Π with respect to S_g, then we say *Justified*(α, Π, S_g). In cases where it is clear which S_g and plan are referred to, we will simply say that α is justified (*Justified*(α)). A plan Π is justified if every operator in it is justified. An unjustified plan Π (one for which *Justified* is false) can be justified by removing all unjustified operators.

In the plan for Example 1, each operator is justified, since each establishes one of the three goal conditions. If we were to prepend **MoveS**$(1, 2)$ onto the beginning of this plan, it would remain correct, but would no longer be justified, since this first operator makes no contribution to the satisfaction of the goal. Although it might seem odd that this plan is correct, note that at the abstract level only the **IsPeg** preconditions of the **MoveS** operator need to be satisfied since the **OnSmall** precondition has a lower criticality. Thus, at the abstract level, one can add the operator **MoveS**$(1, 2)$ *even if the small disk is not on the first peg at the point in which the operator is inserted into the plan.*

If Π is a plan for achieving S_g, then ${}_i\Pi$ is the plan obtained by replacing each operator α in Π by ${}_i\alpha$. Since only the justified operators are needed to satisfy the goals, it can be easily shown that if Π is a plan that solves goal S_g at level i, then its justified version is also a plan that solves S_g. For example, if the **OnMedium** preconditions of each operator are eliminated at level 2, then those plan steps from levels below 2 that only achieve **OnMedium** can be removed. This result, known as the *Upward Solution Property* [Tenenberg, 1988], can be easily extended to a multi-level hierarchy: if Π is a plan that solves S_g at the base level of a k level abstraction hierarchy, then the justified version of ${}_i\Pi$ is also a plan that solves S_g on the i^{th} level, $0 \leq i \leq k-1$. A formal proof can be found in [Tenenberg, 1988].

Refinement

With the notion of justification, we can now define the "refinement" of an abstract plan. Intuitively, a plan Π is a refinement of an abstract plan Π', if all operators and their ordering relations in Π' are preserved in Π, and the new operators have been inserted for the purpose of satisfying one of the re-introduced preconditions.

Definition 3 *A plan* Π *at level* $i-1$ *is a* refinement *of an abstract plan* Π' *at level* i, *if*

1. Π *is justified at level* $i-1$, *and*
2. *there is a 1-1 function* c *(a* correspondence function*) mapping each operator of* Π' *into* Π, *such that*
   (a) $\forall \alpha \in$ operators(Π'), ${}_ic(\alpha) = \alpha$,
   (b) if $\alpha \prec_{\Pi'} \beta$, *then* $c(\alpha) \prec_\Pi c(\beta)$,
   (c) $\forall \gamma \in$ *operators*(Π), $\forall \alpha \in$ operators(Π'), *if* $c(\alpha) \neq \gamma$, *then* $\exists \beta \in$ operators(Π) *with precondition* p *such that* Justified(γ, Π, p) *and* crit$(p) = i-1$.

If Π is a refinement of Π', then we say that Π' refines to Π. This formal definition captures the notion of plan refinements used in many different planners, including ABSTRIPS [Sacerdoti, 1974], SIPE [Wilkins, 1984], ALPINE [Knoblock, 1991], and ABTWEAK [Yang and Tenenberg, 1990].

Example 2 In our previous Tower of Hanoi example, the following plan at level 2,

$$\langle \texttt{MoveL}(\texttt{P}_1, \texttt{P}_3), \texttt{MoveM}(\texttt{P}_1, \texttt{P}_3), \texttt{MoveS}(\texttt{P}_1, \texttt{P}_3) \rangle$$

refines to the following plan at level 1, when we reintroduce all `OnMedium` preconditions:

$\langle$`MoveM(P1,P2)`, `MoveL(P1,P3)`, `MoveM(P2,P1)`, `MoveM(P1,P3)`, `MoveS(P1,P3)`$\rangle$.

To verify that it satisfies the refinement definition, note that it solves the problem at level 1, the ordering of the abstract operators is preserved in the refinement, and the inserted operators at level 1, (the new `MoveM` operators), establish the preconditions ¬`OnMedium(P1)` for `MoveL(P1,P3)`, and `OnMedium(P1)` for `MoveM(P1,P3)`, respectively, each of these preconditions having criticality 1.

The Monotonicity Property

The refinement of an abstract plan places almost no constraint on search. This section defines the monotonicity property, which relates the plans at successive levels of abstraction in terms of the establishment relations. First, we define a *monotonic refinement* of an abstract plan, which in turn is used to define a monotonic abstraction hierarchy.

Definition 4 *Let Π' be an abstract plan that solves $\rho = (S_0, S_g)$ at level i, $i > 0$ and is justified relative to S_g. A level $i-1$ plan Π is a* monotonic refinement *of a level i plan Π' if and only if*

(1) Π is a refinement of Π',
(2) Π solves ρ at level $i-1$, and
(3) Justified$({}_i\Pi) = \Pi'$.

This last condition states that the refined plan when justified at the abstract level is equal to the abstract plan. That is, plan refinement does not result in the introduction of any new abstract establishments. The refined plan from Example 2 is a monotonic refinement since the inserted `MoveM` operators satisfy a criticality 1 precondition, which can be eliminated at level 2.

Definition 5 *A k-level abstraction hierarchy is monotonic, if, for every problem $\rho = (S_0, S_g)$ solvable at the concrete (0^{th}) level, there exists a sequence of plans $\Pi_{k-1}, \ldots, \Pi_0$ such that Π_{k-1} is a justified plan for solving ρ at level $k-1$, and for $0 < i < k$, Π_{i-1} is a monotonic refinement of Π_i.*

An important feature of the monotonicity property lies in its generality:

Theorem 1 *Every abstraction hierarchy is monotonic.*

It can be shown that if Π is a monotonic refinement of Π', then all of the establishment relations *establishes*(α, β, p) in Π' hold in plan Π. Thus, one can backtrack on any refinement that is not monotonic without losing completeness.

Example 3 As another example, consider a simple domain where a robot can move between two rooms, $Room_1$ and $Room_2$, connected by a door, $Door_{12}$, that can be opened or closed. We can build an abstraction hierarchy by eliminating all preconditions involving whether the door is open or closed. Given the goal of getting the robot into $Room_1$ and closing $Door_{12}$ the system might construct an abstract plan that moved the robot into $Room_2$ and closed the door, and then moved the robot through the closed door into $Room_1$. This plan could be refined by inserting steps to open $Door_{12}$ in order to get into $Room_1$. But this is not a monotonic refinement since it violates a condition established in the abstract plan. In planning terms, the establishing literals from the level above are *protected* during plan refinement. Instead of forging ahead with this refinement, the system can backtrack and change the abstract plan to close the door once the robot is inside $Room_1$, which produces the correct solution.

The Ordered Monotonicity Property

Partitioned Abstraction Hierarchies

We consider here a more restrictive type of abstraction hierarchy, which we call a *partitioned* hierarchy. Partitioned abstraction hierarchies are obtained from criticality functions constrained to assign the same criticality to all precondition literals having the same predicate. More formally, a k-level *partitioned* abstraction hierarchy is a k-level hierarchy $\Sigma = (L, S, O, crit)$, such that $\forall \alpha, \beta \in O,\ l_1 \in O_\alpha,\ l_2 \in O_\beta,\ crit(l_1, \alpha) = crit(l_2, \beta)$ if l_1 and l_2 have the same predicate. Since the criticality of a literal l does not depend on the operator, it will simply be denoted as $crit(l)$.

Ordered Abstraction Hierarchies

Now we consider a property that ensures that literals added at an abstract level are never violated by any operators added at lower levels in the refinement process. A refinement of an abstract plan that satisfies this property is called an *ordered refinement.*

Definition 6 *Let Π' be a justified plan that solves $\rho = (S_0, S_g)$ at level i, $i > 0$. A level $i-1$ plan Π is an* ordered refinement *of a level i plan Π' if and only if*

(1) Π is a monotonic refinement of Π', and
(2) $\forall \alpha \in$ operators(Π), if ${}_i\alpha \notin$ operators(Π'), then α does not add or delete any literal l with crit$(l) > i-1$.

The second condition states that in plan Π, only the operators that correspond to the operators in the abstract plan Π' are allowed to modify literals with criticality values higher than $i-1$.

We now define the *ordered monotonicity property* for an abstraction hierarchy:

Definition 7 *An abstraction hierarchy has the* ordered monotonicity property[1] *if, for all levels $i, i-1$*

[1] This definition of ordered monotonicity is slightly more restrictive than the ordered monotonicity property that was informally described in [Knoblock, 1990a].

($0 < i < k$), for every problem ρ, if Π' is a justified plan that solves ρ at level i, then every refinement of Π' at level $i-1$ is an ordered refinement.

An abstraction hierarchy is said to be *ordered* if it has the ordered monotonicity property. Note that this property does not say that every abstract solution *can be* refined; it only states that every refinement of an abstract plan must be ordered.

An ordered hierarchy has at least two important implications. First, it guarantees that every possible refinement of an abstract plan will leave the conditions established in the abstract plan unchanged. In contrast, the monotonicity property requires explicit protection of these conditions. By ensuring that every refinement is ordered, the ordered monotonicity property guarantees that no violation of the monotonic property will ever occur during plan generation. Thus, there is no need to spend an additional computational resource in checking for such violations.

Second, the ordered monotonicity property captures an important intuition behind "good" abstraction hierarchies. An ideal abstraction hierarchy decomposes a complex problem into parts with different levels of difficulties. Once a solution is found for solving the most difficult parts, one solves the detailed, less difficult parts by inserting steps into the abstract solution. During the refinement process, the ordered monotonicity property guarantees that the problem solver will naturally avoid the parts of the search space relevant to the problems already solved in more abstract spaces.

Unlike the monotonicity property, the ordered monotonicity property is not satisfied by all abstraction hierarchies. It is therefore important to explore conditions under which a hierarchy satisfies this property. The following is a set of conditions which are sufficient but not necessary to guarantee the ordered monotonicity property. (A set of less restrictive problem-specific conditions that are also sufficient to guarantee the ordered monotonicity property is described in [Knoblock, 1991].)

Restriction 1 *Let O be the set of operators in a domain. $\forall \alpha \in O, \forall p \in P_\alpha$ and $\forall e_1, e_2 \in A_\alpha \bigcup D_\alpha$,*

(1) $\mathrm{crit}(e_1) = \mathrm{crit}(e_2)$, *and*
(2) $\mathrm{crit}(e_1) \geq \mathrm{crit}(p)$.

This restriction is called the *Ordered Restriction*. Stated simply, all effects of an operator have the same criticality, and have criticality at least as great as the operator's preconditions.

Theorem 2 *Any partitioned abstraction hierarchy satisfying Restriction 1 is an ordered hierarchy.*

The reason why this theorem holds can be explained informally as follows. Restriction 1 partitions the operators into disjoint classes according to their effects. It also imposes a partial ordering on the classes. The ordering has the following property: if an operator class A precedes B in that ordering, then an operator β in B does not change any condition e in the effects of any operator α in A. This implies that if all predicates that are effects of operators in A have a higher criticality than the effects of any of the operators in B. Thus, the ordered monotonicity must hold.

Example 4 The criticality assignment that has been used throughout in our Tower of Hanoi problem satisfies Restriction 1, and hence results in an ordered hierarchy. Consider our earlier level 2 plan:

$$\langle \texttt{MoveL}(\texttt{P}_1, \texttt{P}_3), \texttt{MoveM}(\texttt{P}_1, \texttt{P}_3), \texttt{MoveS}(\texttt{P}_1, \texttt{P}_3) \rangle.$$

To see that *every* refinement at each lower level is an ordered refinement, note the following. By definition, for every refinement, every added operator must be justified by a criticality 1 literal. Since the `MoveL` operator only satisfies the `OnLarge` preconditions, it cannot be justified with respect to any criticality 1 or 0 literal, and thus no `MoveL` operators can occur in any refinement. But then, this satisfies the ordered monotonicity property, since the only added operators, `MoveS` and `MoveM`, never add or delete literals having criticality 2. The argument for the level 0 refinement with respect to the `OnMedium` literals and the `MoveM` operator is analogous.

On the other hand, consider the criticality function assigning 2 to `OnSmall`, 1 to `OnMedium`, and 0 to `Ispeg` and `OnLarge`. A level 2 solution to the same problem is:

$$\langle \texttt{MoveS}(\texttt{P}_1, \texttt{P}_3), \texttt{MoveM}(\texttt{P}_1, \texttt{P}_3), \texttt{MoveL}(\texttt{P}_1, \texttt{P}_3) \rangle.$$

Any level 1 refinement to this plan will require the addition of `MoveS` operators in order to establish the `MoveM` and `MoveL` operators. Since `OnSmall` is a criticality 2 literal, this violates the ordered monotonic property.

A Spectrum of Abstraction Hierarchies

The heuristic power of an abstraction hierarchy depends on the constraints placed on the assignment of criticalities to the preconditions. Above, we provided a formal discussion of refinement in abstract planning, the monotonicity property and the ordered monotonicity property. One purpose of the formal characterization is to facilitate the comparison between different planning systems. Here, we will demonstrate how some of the existing systems can be classified under the same formalism.

Our analysis reveals that various systems and their associated abstraction hierarchies can be placed on a spectrum formed by the formal properties and restrictions, where each restriction includes the previous ones. Thus, at each place in the spectrum, the given restriction ensures not only its associated heuristic property, but all of the previous ones as well. Table 2 lists each restriction with the corresponding properties and examples of systems whose abstraction hierarchies satisfy the restrictions. Below, we briefly discuss each in turn.

Restriction	Properties	Examples
Precondition Elimination	Upward Solution	ABSTRIPS
		SOAR
	Monotonicity	ABTWEAK
		PABLO
Partitioned		LAWALY
		SIPE
Ordered	Ordered Monotonicity	ALPINE

Table 2: Restrictions and Properties of Abstraction Hierarchies

At the first point in the spectrum, criticalities are assigned without constraint to the preconditions of the operators. A literal that occurs as a precondition in two different operators can be assigned different criticalities in each operator. ABSTRIPS [Sacerdoti, 1974], ABTWEAK [Yang and Tenenberg, 1990], SOAR [Unruh and Rosenbloom, 1989], and PABLO [Christensen, 1990] are examples of systems in this class. Any system in this class will have both the upward solution property and the monotonicity property. Although the monotonicity property applies to any system in this class, only ABTWEAK explicitly uses this property in its search strategy.

At the second point in the spectrum the following constraint is enforced on the assignment of the criticalities: literals of the same predicate must have the same criticality. This corresponds to *partitioned* abstraction hierarchies. LAWALY [Siklossy and Dreussi, 1973] and SIPE [Wilkins, 1984] are both examples of systems in this class. The additional partitioning constraint does not provide any additional known formal properties.

At the third point on the spectrum lie the ordered hierarchies. Recall that these hierarchies impose an order on a partitioned hierarchy such that every refinement of an abstract plan will leave the literals in a more abstract space unchanged. This property is used in the ALPINE system [Knoblock, 1990a], which automatically generates abstraction hierarchies that have this property.

Conclusion and Future Directions

This paper presents a formalism for studying abstraction in planning. It explores the properties of abstraction hierarchies that are generated by gradually restricting the assignment of criticality values to preconditions of operators. These properties can be applied to both the construction and use of abstraction hierarchies for planning. The monotonicity property holds for every abstraction hierarchy and can be used to prune the search space without sacrificing completeness. The ABTWEAK planner [Yang and Tenenberg, 1990] exploits this property (within a nonlinear, least-commitment planner) to constrain the search space. The ordered monotonicity property and the associated restriction can be used to *generate* abstraction hierarchies from a set of operators. For example, to build ordered hierarchies, one can impose an ordering relation upon the literals in a domain, based on Restriction 1. If the resulting relation is partially ordered, then any total ordering of the relation gives a criticality assignment to literals that satisfies the ordered monotonicity property. Algorithms for automatically generating abstraction hierarchies based on the ordered monotonicity property are presented in [Knoblock, 1991].

References

[Christensen, 1990] Jens Christensen. A hierarchical planner that generates its own abstraction hierarchies. In *Proceedings of Eighth National Conference on Artificial Intelligence*, pages 1004–1009, 1990.

[Knoblock, 1990a] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of Eighth National Conference on Artificial Intelligence*, pages 923–928, 1990.

[Knoblock, 1990b] Craig A. Knoblock. A theory of abstraction for hierarchical planning. In D. Paul Benjamin, editor, *Change of Representation and Inductive Bias*, pages 81–104. Kluwer, Boston, MA, 1990.

[Knoblock, 1991] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Tech. Report CMU-CS-91-120.

[Sacerdoti, 1974] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[Siklossy and Dreussi, 1973] L. Siklossy and J. Dreussi. An efficient robot planner which generates its own procedures. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 423–430, 1973.

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

[Unruh and Rosenbloom, 1989] Amy Unruh and Paul S. Rosenbloom. Abstraction in problem solving and learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 681–687, 1989.

[Wilkins, 1984] David Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22(3):269–301, 1984.

[Yang and Tenenberg, 1990] Qiang Yang and Josh D. Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. In *Proceedings of Eighth National Conference on Artificial Intelligence*, pages 204–209, 1990.